Deep Learning - RNN
深度学习 - RNN

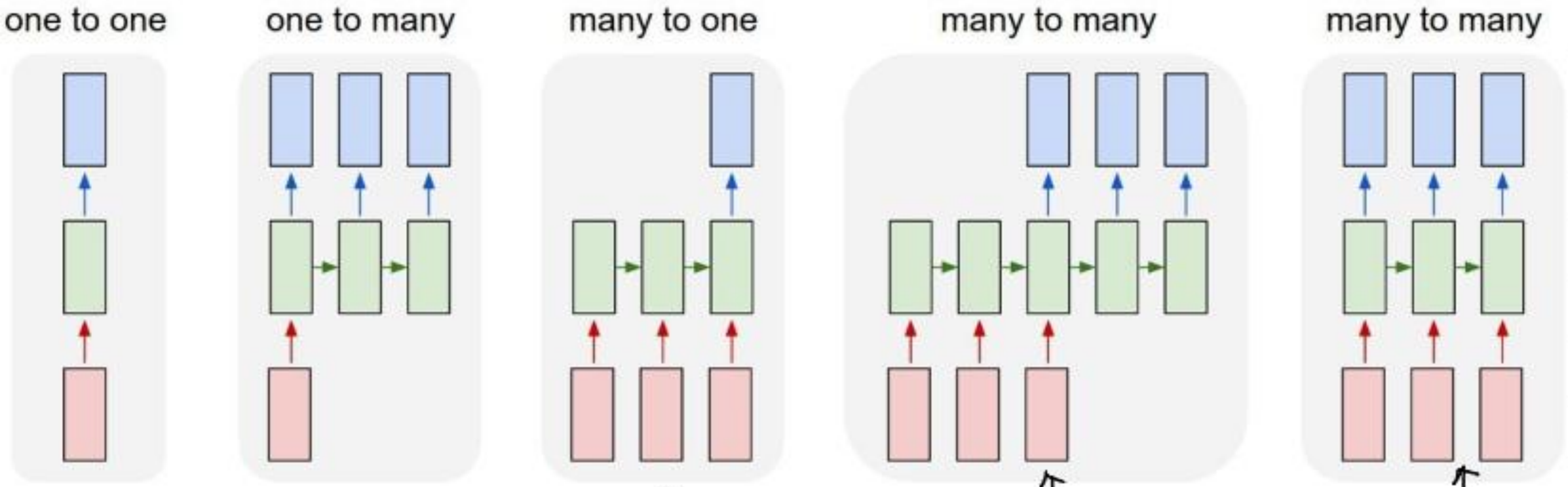
Jian Li

IIIS, Tsinghua

Recurrent Neural Networks (RNN)
递归神经网络

CNN: parameter sharing in space
CNN：在空间上共享参数
RNN: parameter sharing in time (suitable for sequences, in particular sequences with variable lengths)
RNN：在时间上共享参数（适合于序列，尤其是变长序列）

Basics



e.g. **Image Captioning**
image -> sequence of words
图像→序列

machine translation
机器翻译

Video classifiation
视频分类

Sequence of words → Sentiment
序列→情感

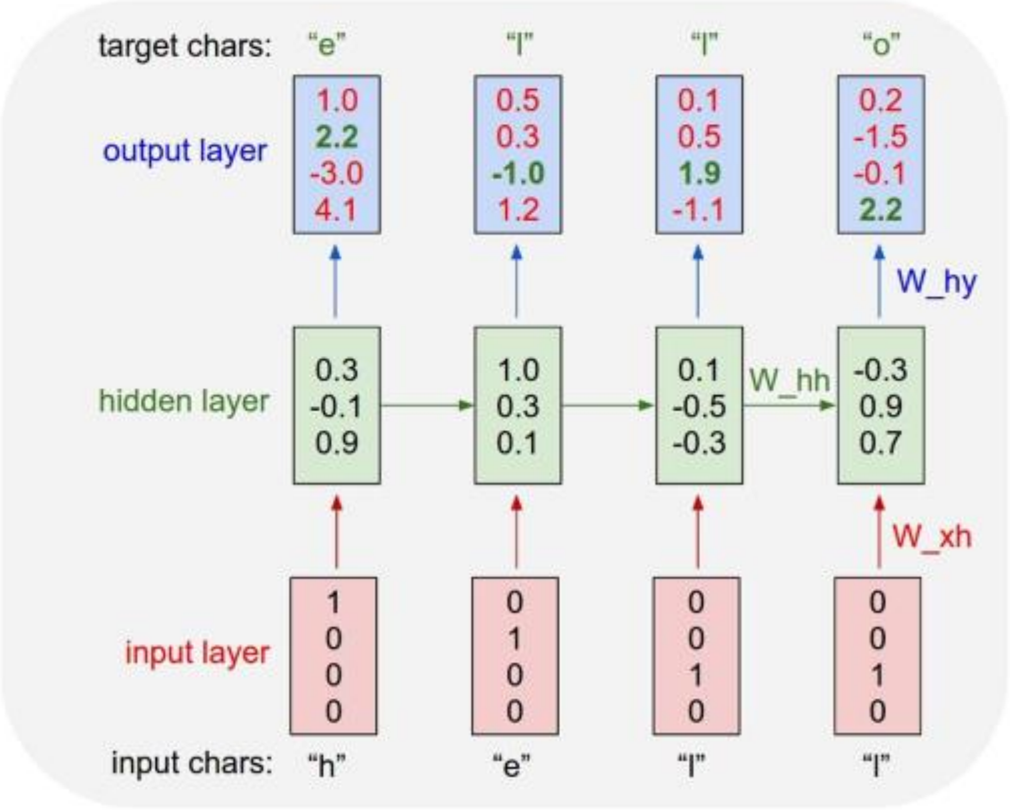
Basics

Character-level language model example

字符级语言模型

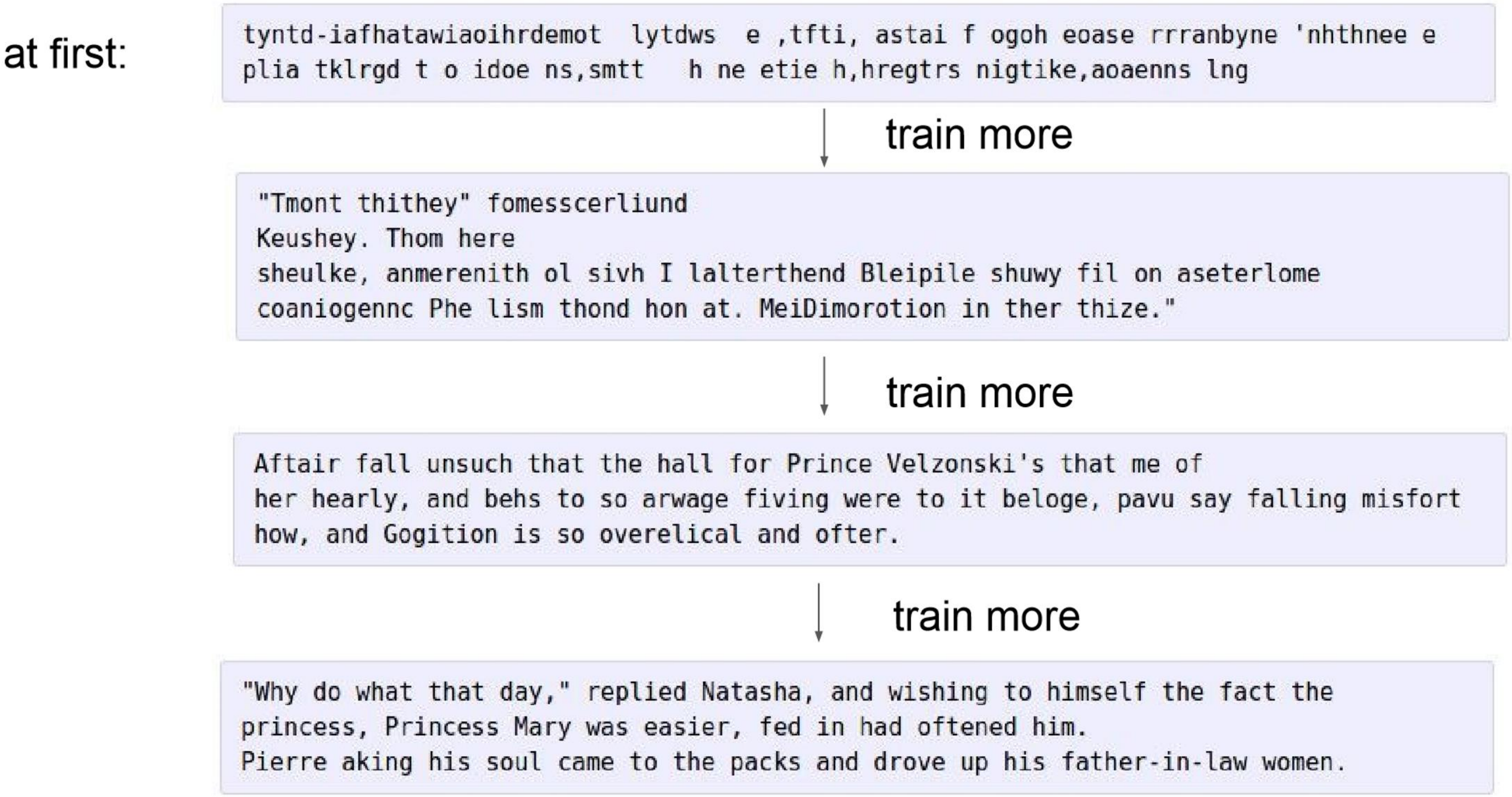
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



- Learns time dependency gradually:

逐渐学习时间的相关性



at first:

```
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

```
"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

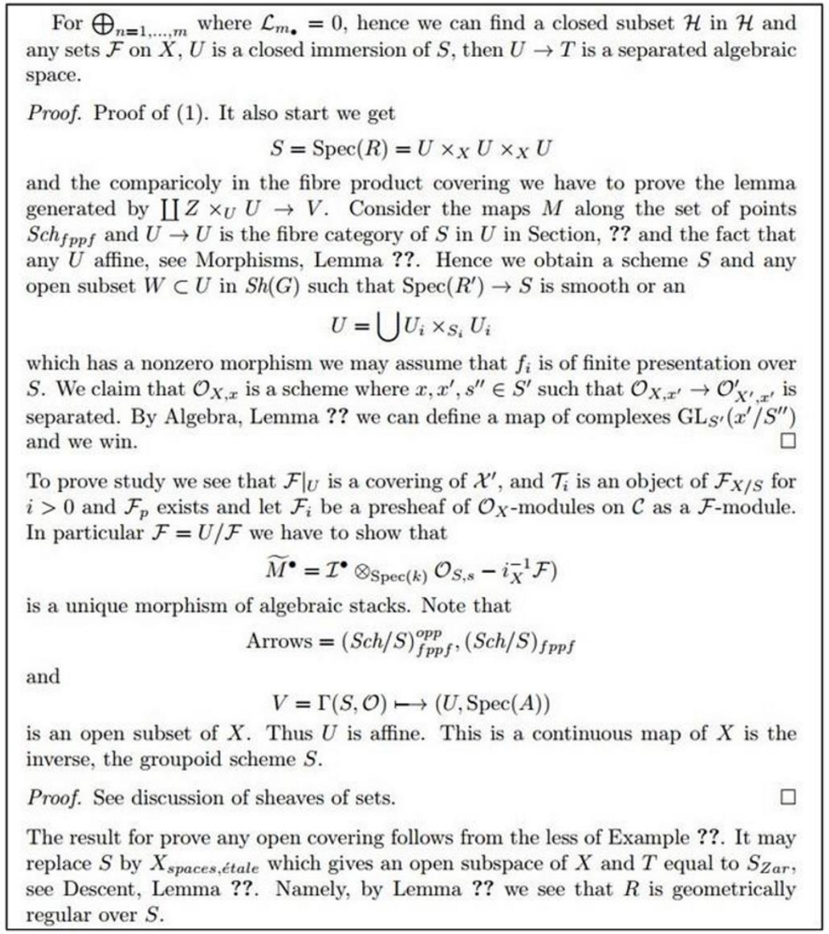
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"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.
```

“Proof” generated by RNN

- Training data – an algebraic geometry book 用一本代数几何书做训练数据



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

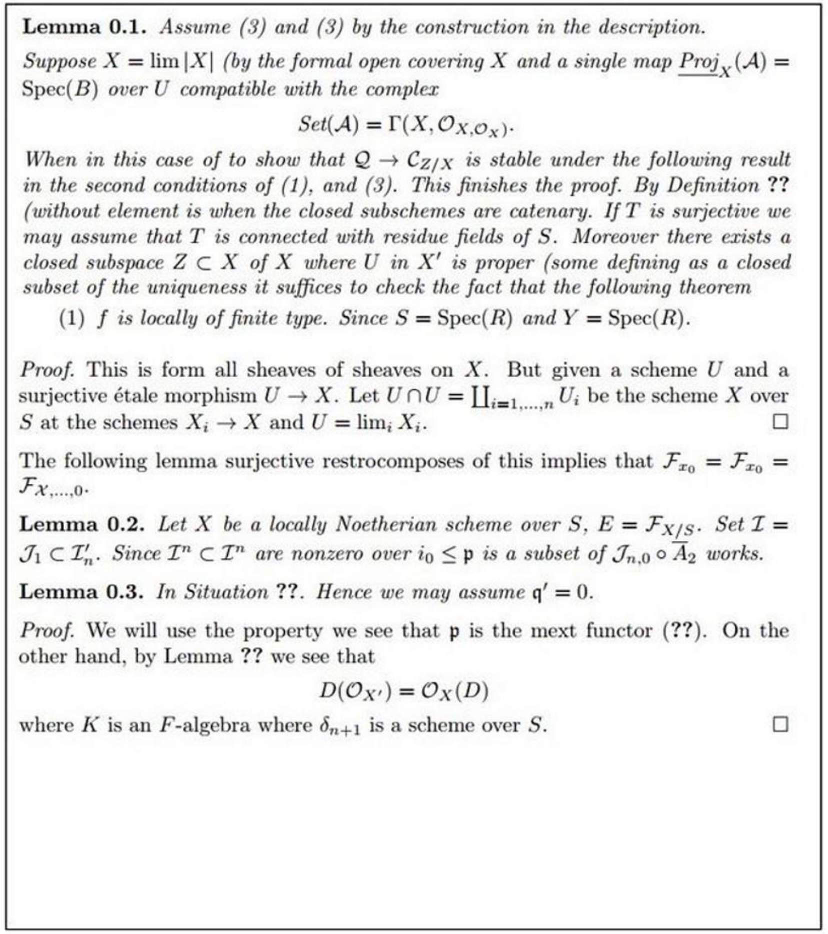
and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.



Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) =$ $\mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition **??** *(without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) *f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

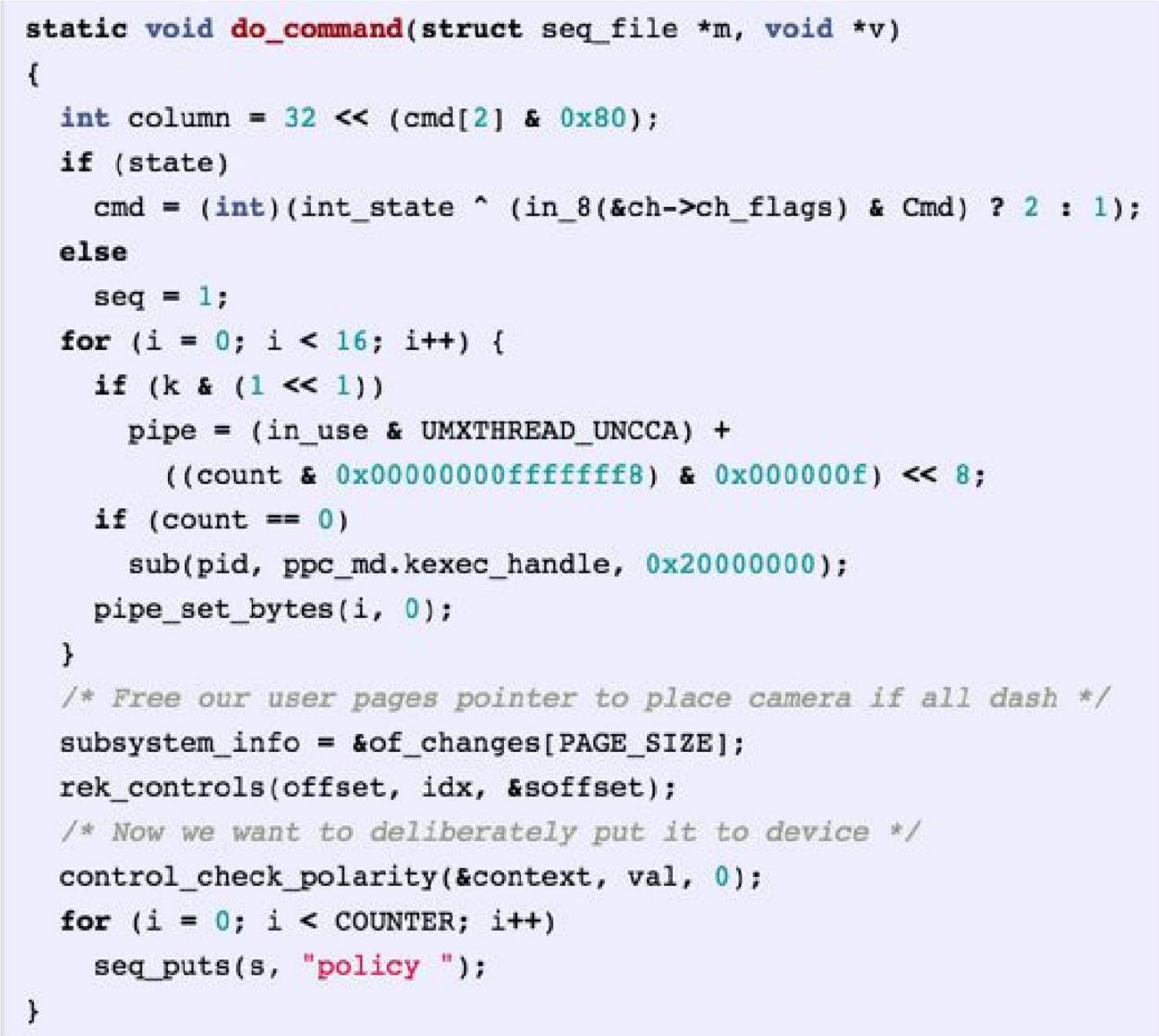
Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation* **??**. *Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □



```c
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 ");
}
```

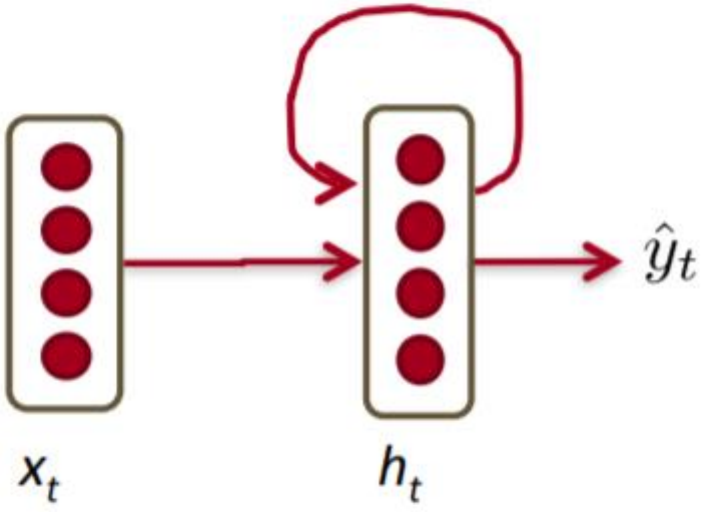
Generated C code

RNN生成的C代码

RNN

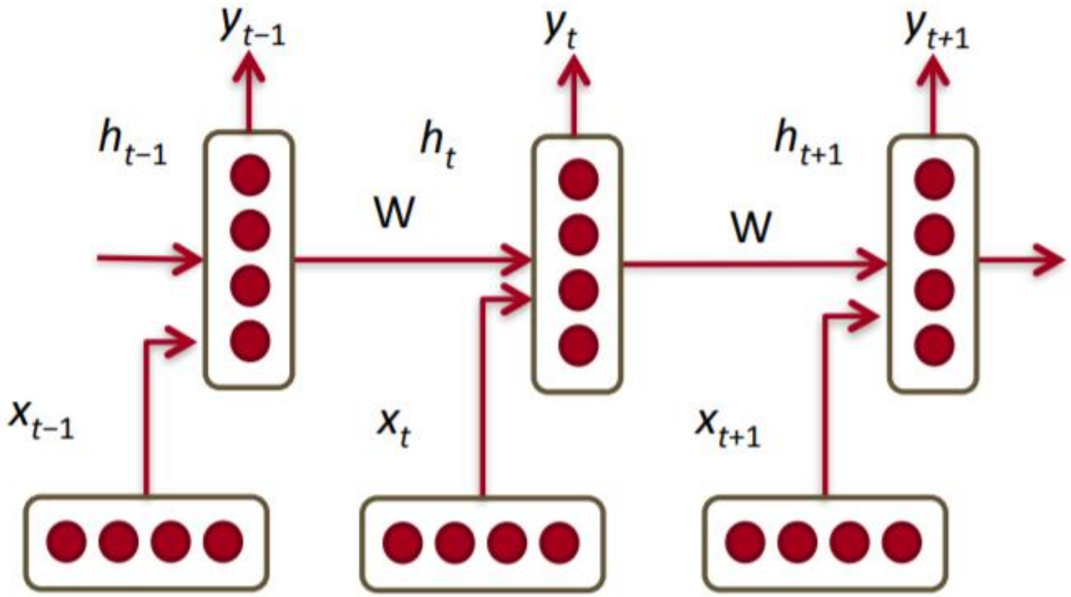
- Self feedback loop

  RNN自反馈循环



- Unroll/Unfold a RNN in time

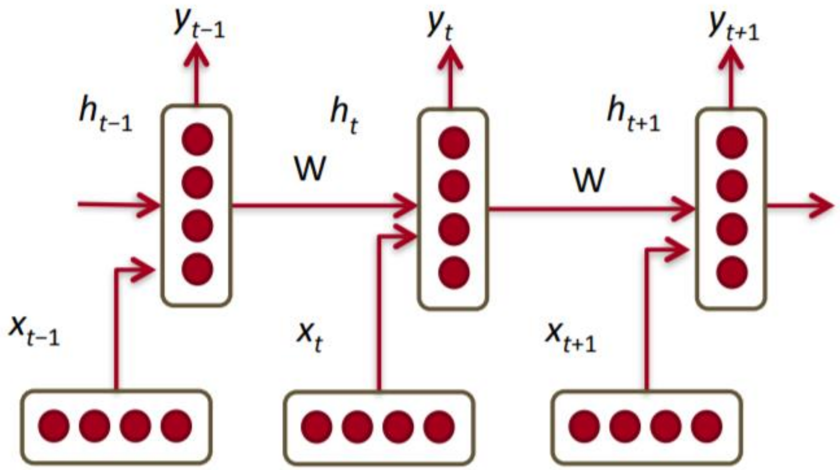
  RNN按时间展开



RNN

- 输入Input: (一组向量 a list of vectors) $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- 在每一个时刻 In each time step:

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\ \hat{y}_t &= \operatorname{softmax}\left(W^{(S)} h_t\right) \end{aligned}$$



$W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$ $W^{(hx)} \in \mathbb{R}^{D_h \times d}$ $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$

Key idea: we use the same set of W weights at all time steps!
主要思想：在所有时间用同样的权重W

双向RNN Bi-directional RNN

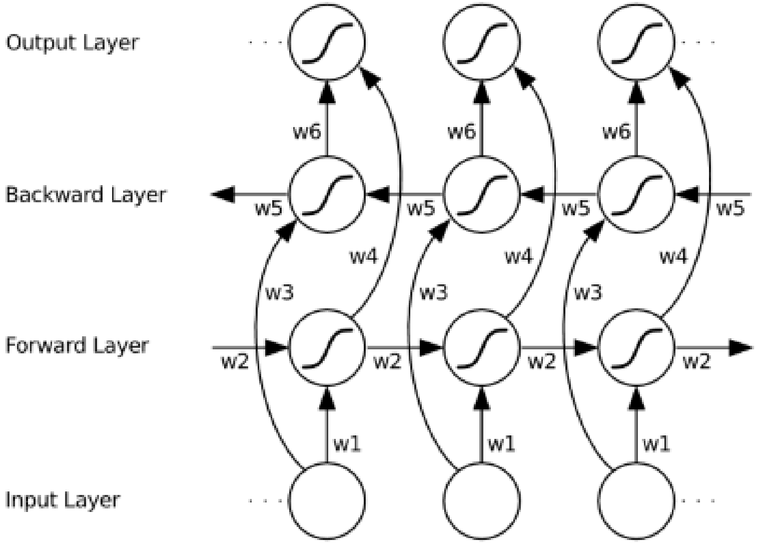
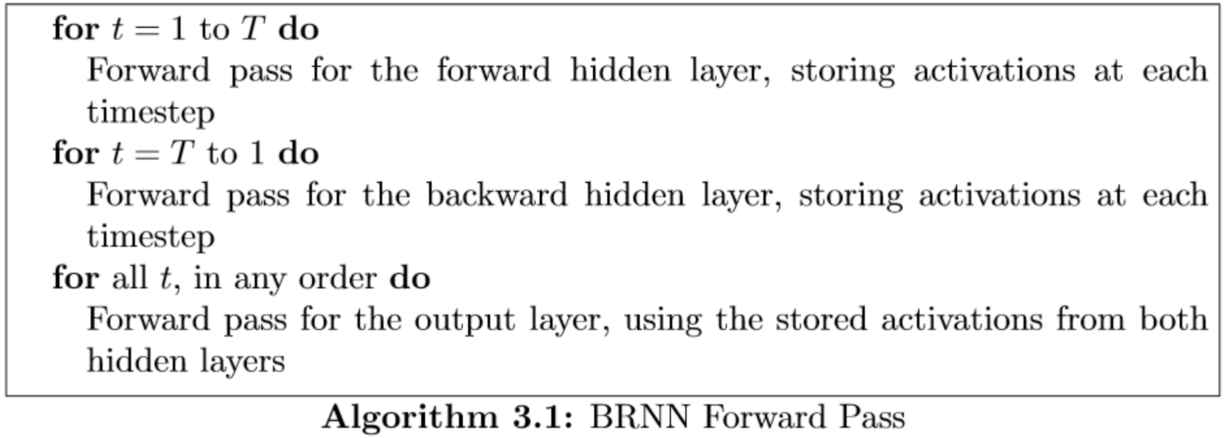
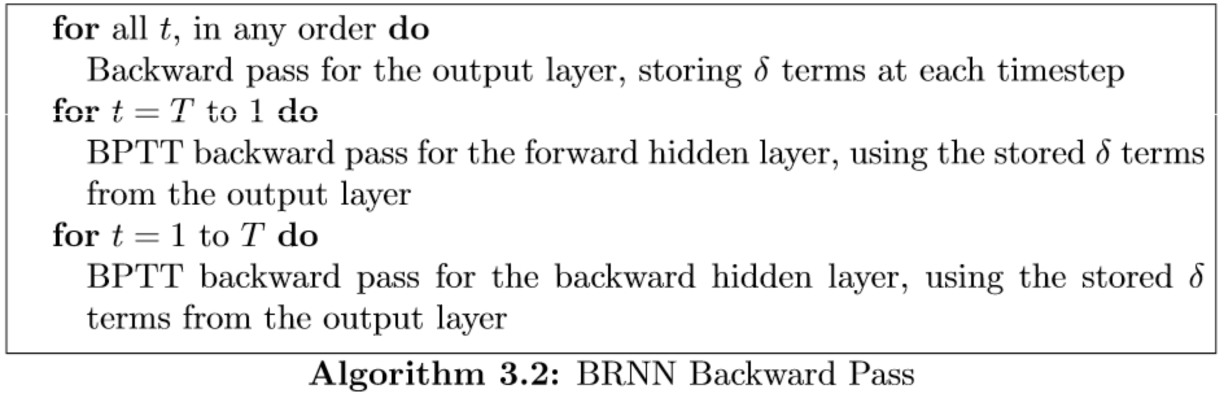


Figure 3.5: **An unfolded bidirectional network.** Six distinct sets of weights are reused at every timestep, corresponding to the input-to-hidden, hidden-to-hidden and hidden-to-output connections of the two hidden layers. Note that no information flows between the forward and backward hidden layers; this ensures that the unfolded graph is acyclic.



```
for $t = 1$ to $T$ do
    Forward pass for the forward hidden layer, storing activations at each timestep
for $t = T$ to 1 do
    Forward pass for the backward hidden layer, storing activations at each timestep
for all $t$, in any order do
    Forward pass for the output layer, using the stored activations from both hidden layers
```

Algorithm 3.1: BRNN Forward Pass

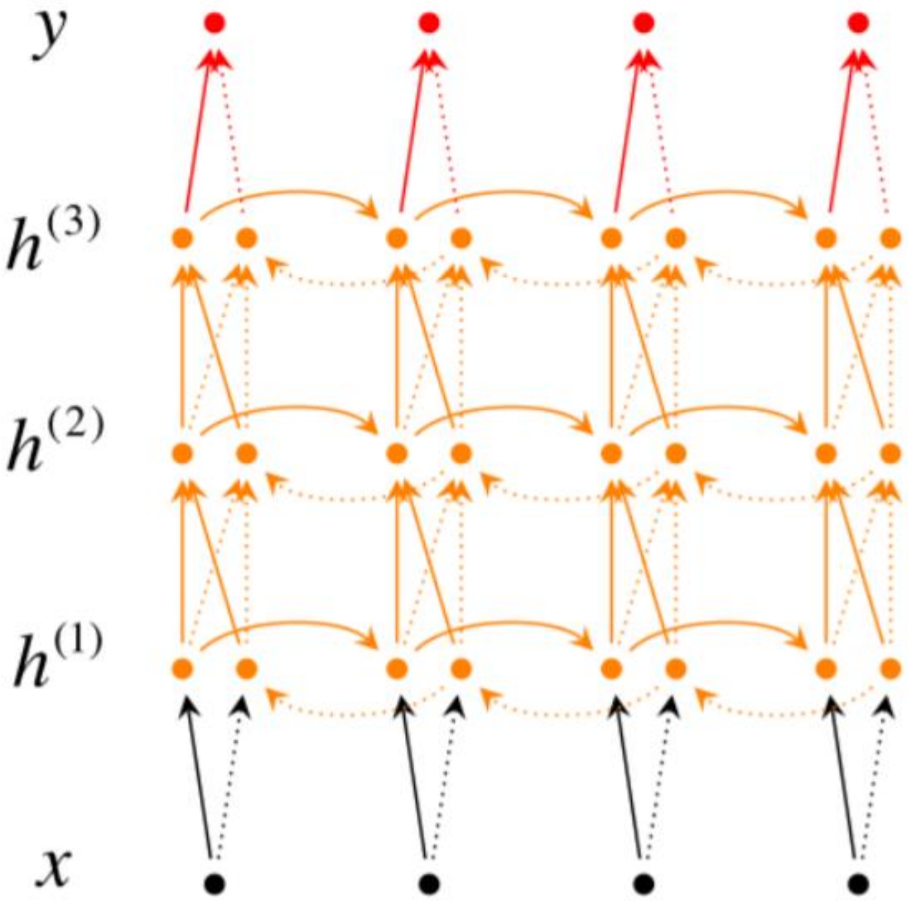


```
for all $t$, in any order do
    Backward pass for the output layer, storing $\delta$ terms at each timestep
for $t = T$ to 1 do
    BPTT backward pass for the forward hidden layer, using the stored $\delta$ terms from the output layer
for $t = 1$ to $T$ do
    BPTT backward pass for the backward hidden layer, using the stored $\delta$ terms from the output layer
```

Algorithm 3.2: BRNN Backward Pass

Figure from Graves.   Supervised Sequence Labelling with Recurrent Neural Networks

深度双向RNN Deep Bi-directional RNN

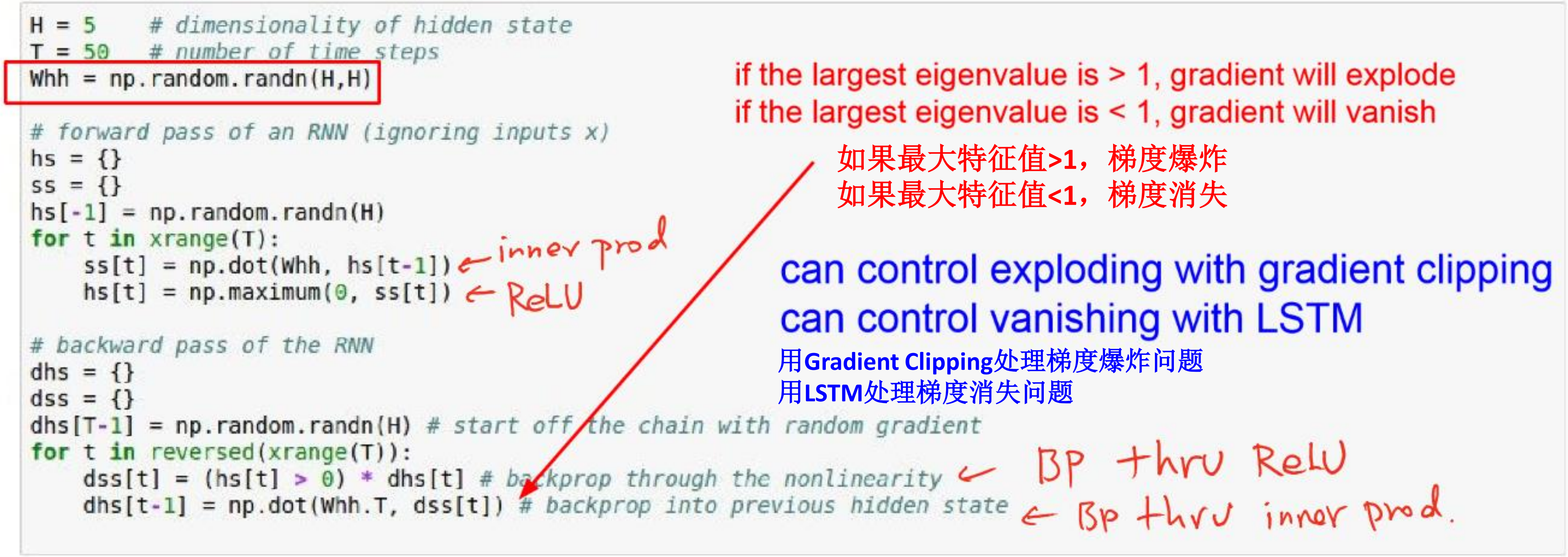


$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)};\overleftarrow{h}_t^{(L)}]+c)$$

梯度消失/爆炸问题
Gradient Vanishing/Exploding problem



```
H = 5    # dimensionality of hidden state
T = 50   # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

if the largest eigenvalue is > 1, gradient will explode
if the largest eigenvalue is < 1, gradient will vanish

如果最大特征值>1，梯度爆炸
如果最大特征值<1，梯度消失

can control exploding with gradient clipping
can control vanishing with LSTM

用Gradient Clipping处理梯度爆炸问题
用LSTM处理梯度消失问题

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

梯度消失/爆炸问题
Gradient Vanishing/Exploding problem

Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= Wf(h_{t-1}) + W^{(hx)}x_{[t]} \\ \hat{y}_t &= W^{(S)}f(h_t) \end{aligned}$$

Total error is the sum of each error at time steps t

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

Hardcore chain rule application:

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

梯度消失/爆炸问题
Gradient Vanishing/Exploding problem

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

- Remember: $h_t \quad = \quad W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

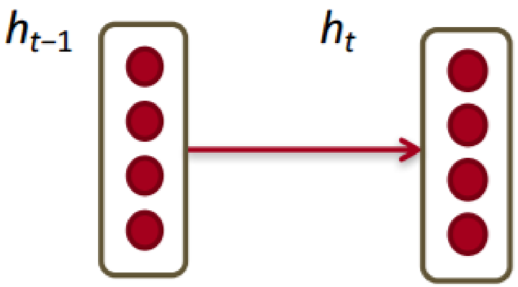
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \quad \cdots \quad \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

梯度消失/爆炸问题
Gradient Vanishing/Exploding problem

- From previous slide: $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$



- Remember: $h_t \quad = \quad W f(h_{t-1}) + W^{(hx)} x_{[t]}$

- To compute Jacobian, derive each element of matrix: $\frac{\partial h_{j,m}}{\partial h_{j-1,n}}$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \mathrm{diag}[f'(h_{j-1})]$$

- Where: $\mathrm{diag}(z) = \begin{pmatrix} z_1 & & & & \\ & z_2 & & 0 & \\ & & \ddots & & \\ & 0 & & z_{n-1} & \\ & & & & z_n \end{pmatrix}$

Check at home
that you understand
the diag matrix
formulation

梯度消失/爆炸问题
Gradient Vanishing/Exploding problem

- Analyzing the norms of the Jacobians, yields:

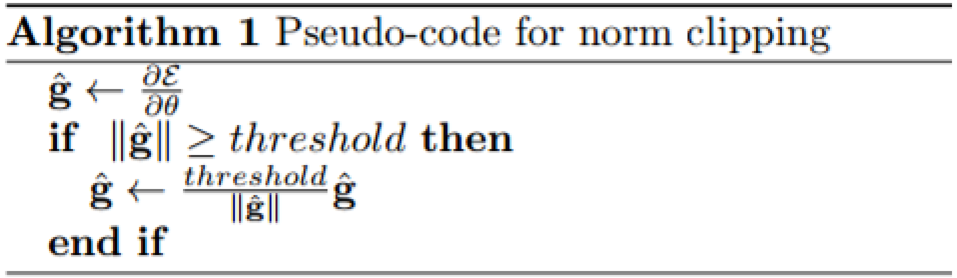
$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\text{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- Where we defined ¯‘s as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

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
```

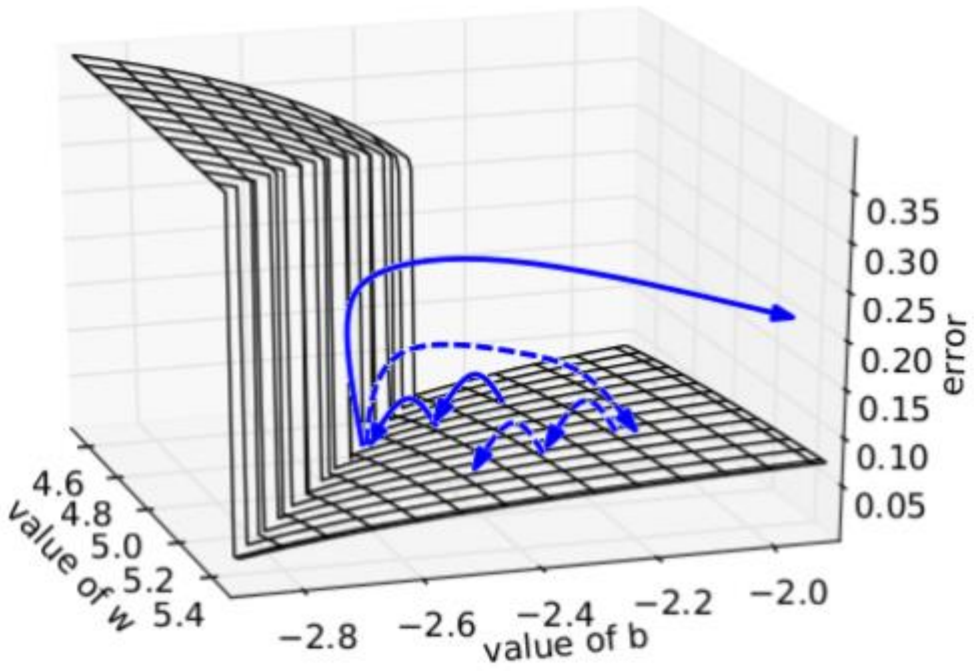


Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depicts standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a fixed size when its norm is above a threshold.

- Keras Code

```
keras.layers.recurrent.SimpleRNN(units, activation='tanh', use_bias=True, kernel_initializer='glorot_uniform',
recurrent_initializer='orthogonal', bias_initializer='zeros', kernel_regularizer=None,
recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)
```

- 参数见：
  https://keras.io/layers/recurrent/

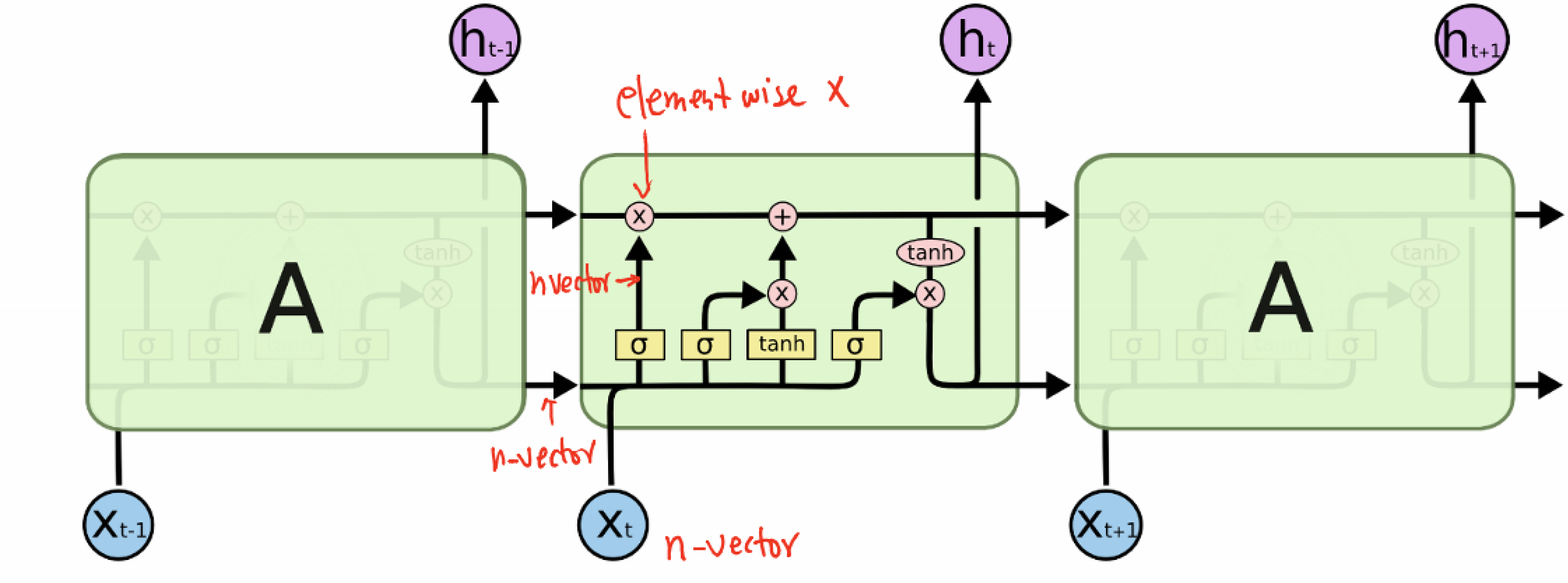
Tensorflow

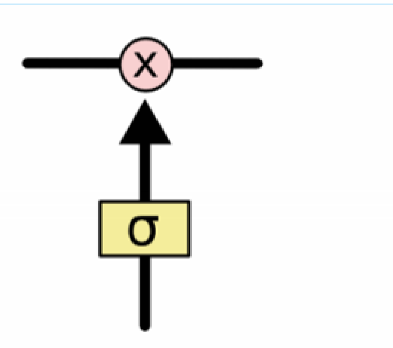
```
class tf.contrib.rnn.BasicRNNCell
```

https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/BasicRNNCell

Long Short Term Memory (LSTM)
长短记忆模型

Overview


h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
x
+
x
x
X_{t-1}
X_t
X_{t+1}
element wise x
nvector →
↑ n-vector
n-vector



gate : $\boxed{\sigma}$ outputs (or a vector) a number in [0,1]

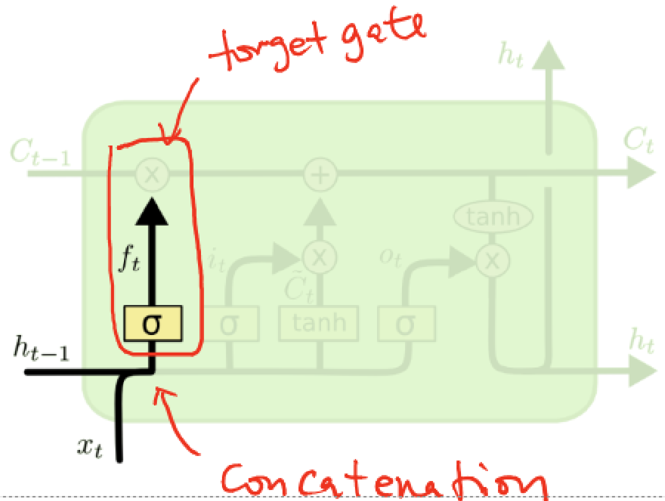
the number decides how much information passes thru

Gate 输出值在[0,1]之间，用于控制有多少信息可以流过/忘记

C_t : Cell state C_t是记忆的内容

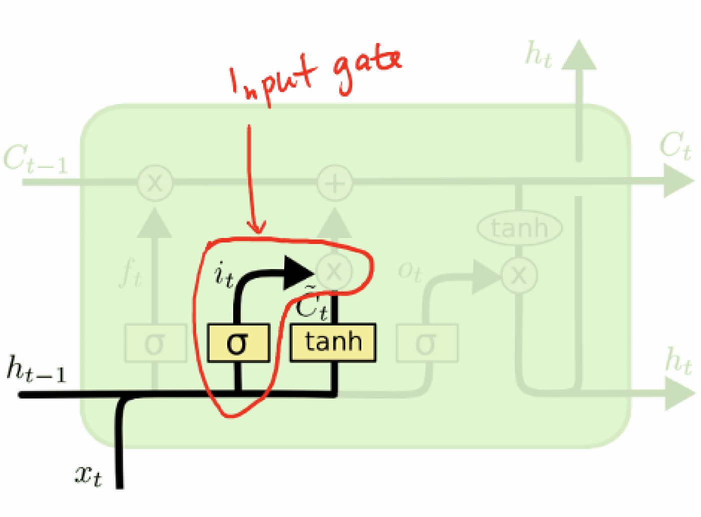
"Forget gate" : Control whether to forget the previous cell state C_{t-1}

Forget Gate 输出值在[0,1]之间，用于控制有多少以前的信息可以流过



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Input gate

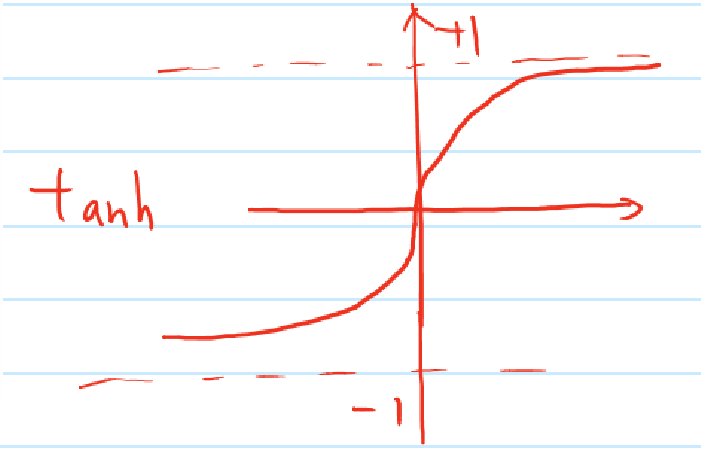


input gate

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

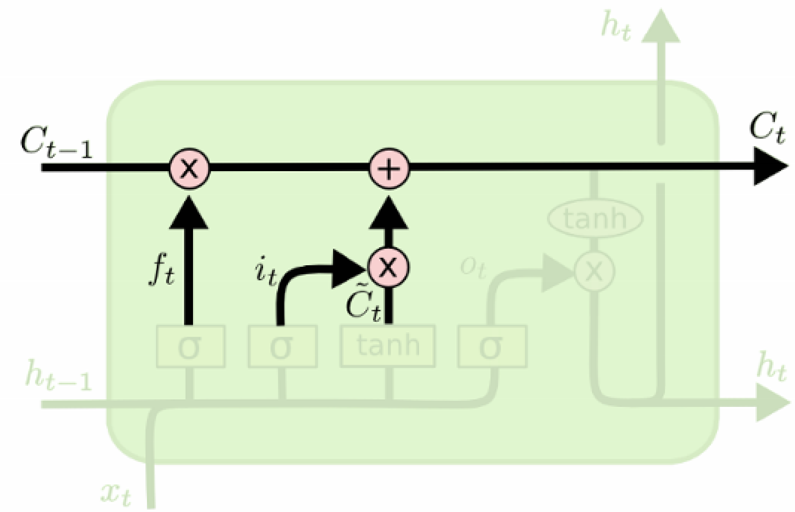
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

new cell state



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

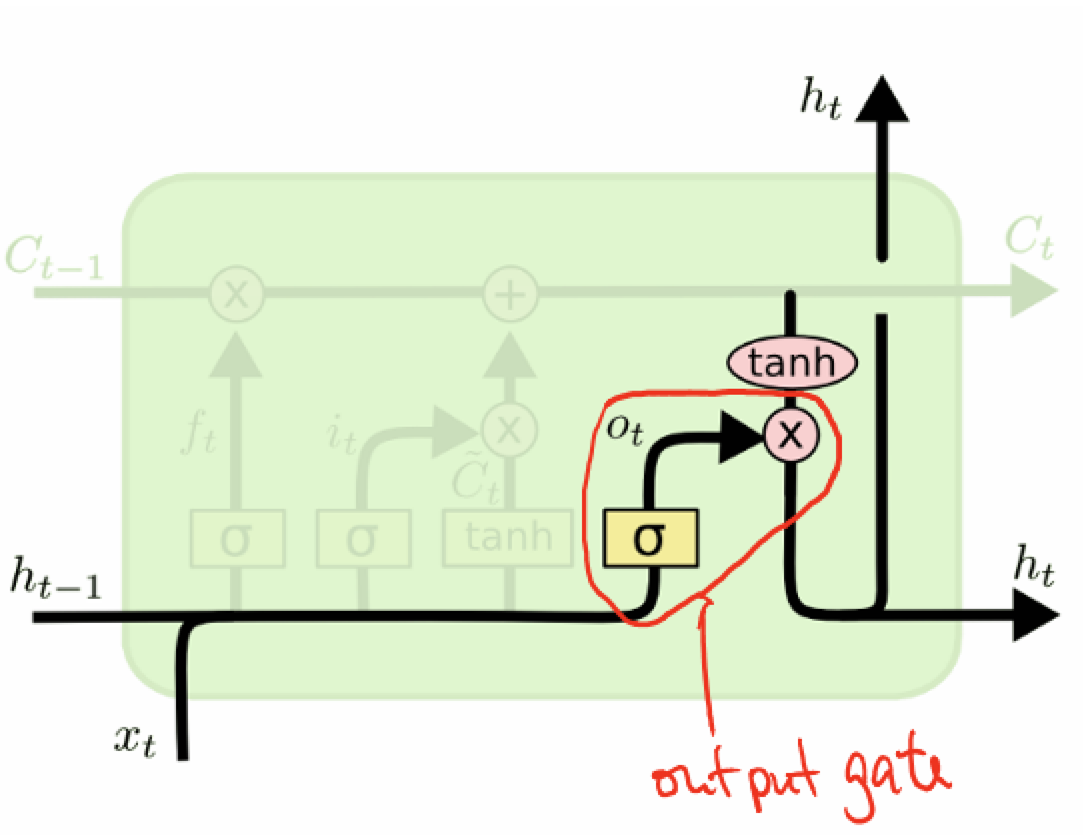
update the cell state C_t



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

new cell state

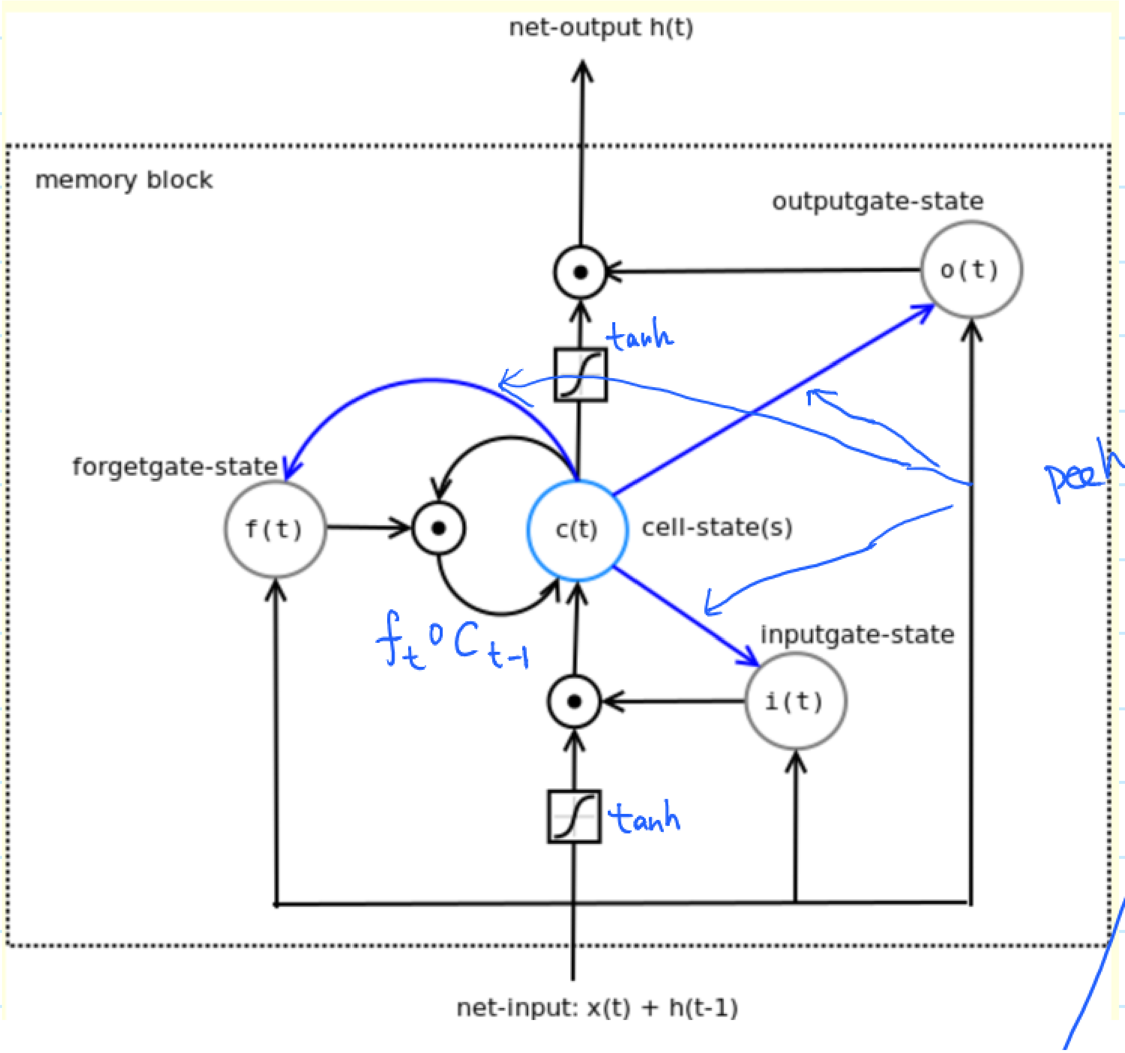
output gate



$$\overset{\text{output gate}}{\underline{o_t}} = \sigma\left(W_o \, [h_{t-1}, x_t] \, + \, b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

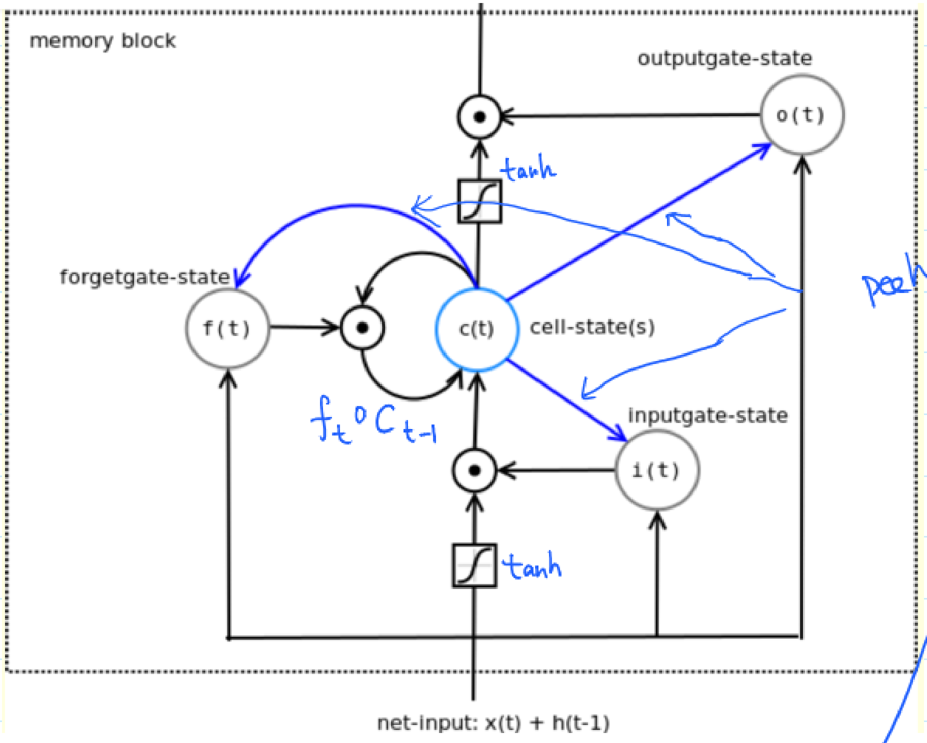
In many papers, it looks like this
LSTM在其他paper中的形式



if there is no such peehole, this is equivalent to the previous one

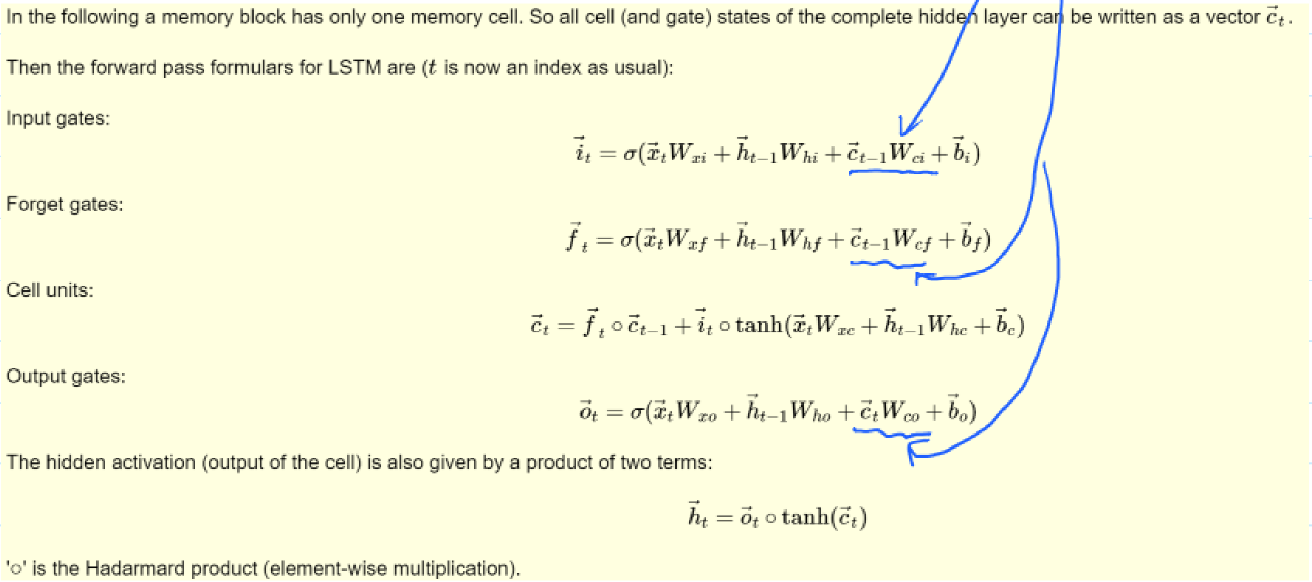
如果没有peehole，和我前面讲的等价

http://christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php



peehole connections

if there is no such peehole, this is equivalent to the previous one



In the following a memory block has only one memory cell. So all cell (and gate) states of the complete hidden layer can be written as a vector $\vec{c}_t$.

Then the forward pass formulars for LSTM are (t is now an index as usual):

Input gates:

$$\vec{i}_t = \sigma(\vec{x}_t W_{xi} + \vec{h}_{t-1} W_{hi} + \underline{\vec{c}_{t-1} W_{ci}} + \vec{b}_i)$$

Forget gates:

$$\vec{f}_t = \sigma(\vec{x}_t W_{xf} + \vec{h}_{t-1} W_{hf} + \underline{\vec{c}_{t-1}} W_{cf} + \vec{b}_f)$$

Cell units:

$$\vec{c}_t = \vec{f}_t \circ \vec{c}_{t-1} + \vec{i}_t \circ \tanh(\vec{x}_t W_{xc} + \vec{h}_{t-1} W_{hc} + \vec{b}_c)$$

Output gates:

$$\vec{o}_t = \sigma(\vec{x}_t W_{xo} + \vec{h}_{t-1} W_{ho} + \underline{\vec{c}_t W_{co}} + \vec{b}_o)$$

The hidden activation (output of the cell) is also given by a product of two terms:

$$\vec{h}_t = \vec{o}_t \circ \tanh(\vec{c}_t)$$

'∘' is the Hadarmard product (element-wise multiplication).

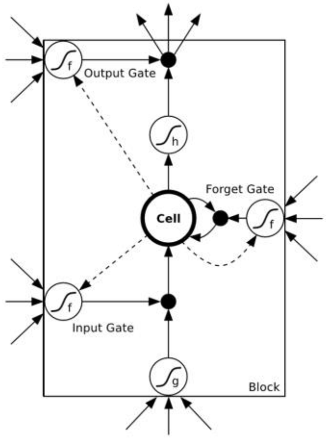
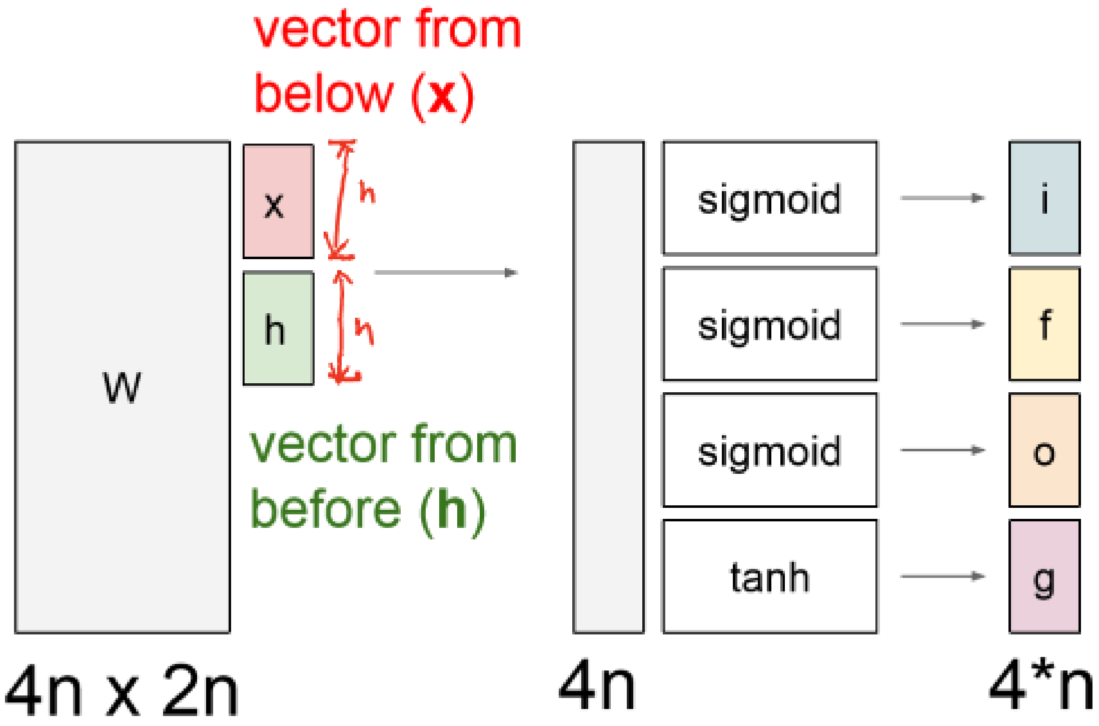
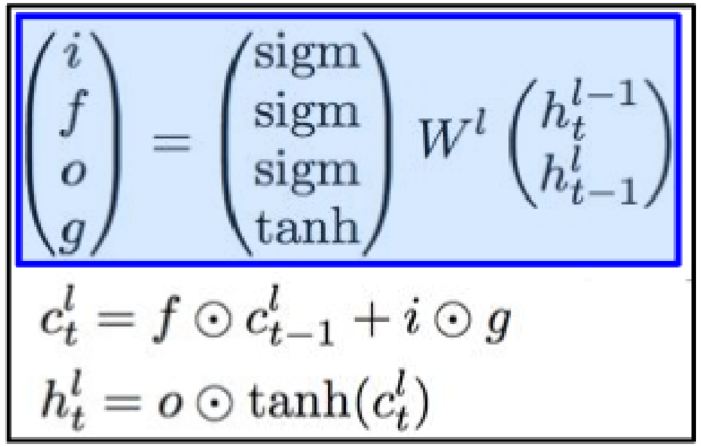


Figure 4.2: **LSTM memory block with one cell.** The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell's previous state. No activation function is applied within the cell. The gate activation function 'f' is usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open). The cell input and output activation functions ('g' and 'h') are usually tanh or logistic sigmoid, though in some cases 'h' is the identity function. The weighted 'peephole' connections from the cell to the gates are shown with dashed lines. All other connections within the block are unweighted (or equivalently, have a fixed weight of 1.0). The only outputs from the block to the rest of the network emanate from the output gate multiplication.

精简矩阵形式 More compact





$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

An LSTM Network
LSTM网络

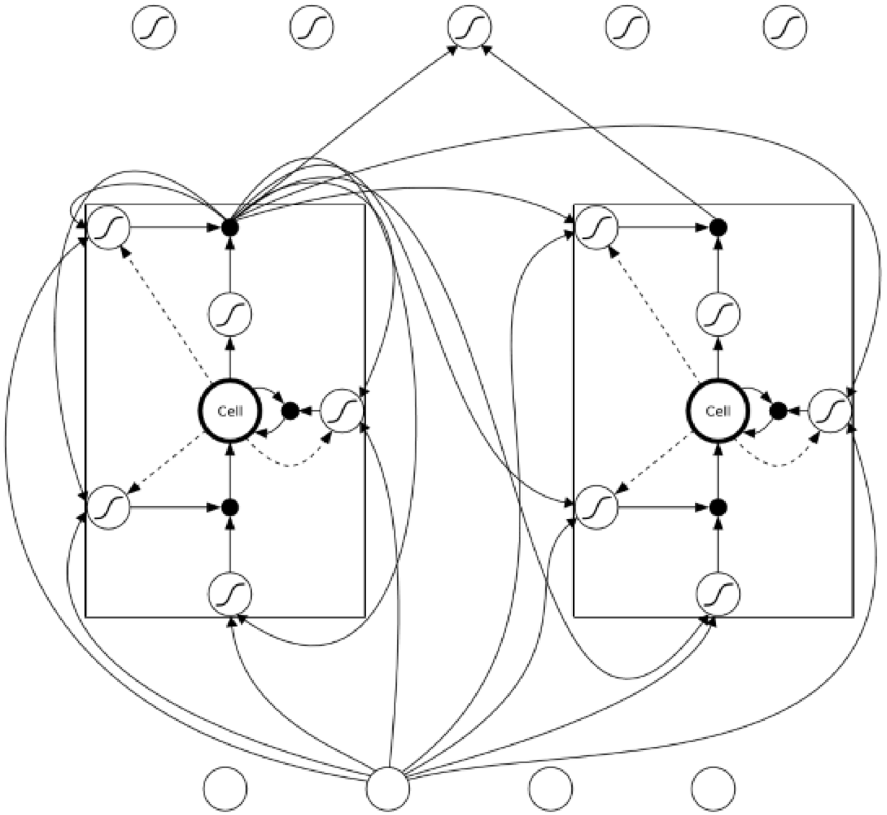


Figure 4.3: **An LSTM network.** The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output.

← not the network Unrolled in time

该图并未随时间展开
因此，输出作为feedback喂给输入

How LSTM deal with gradient vanishing problem LSTM如何解决梯度消失问题

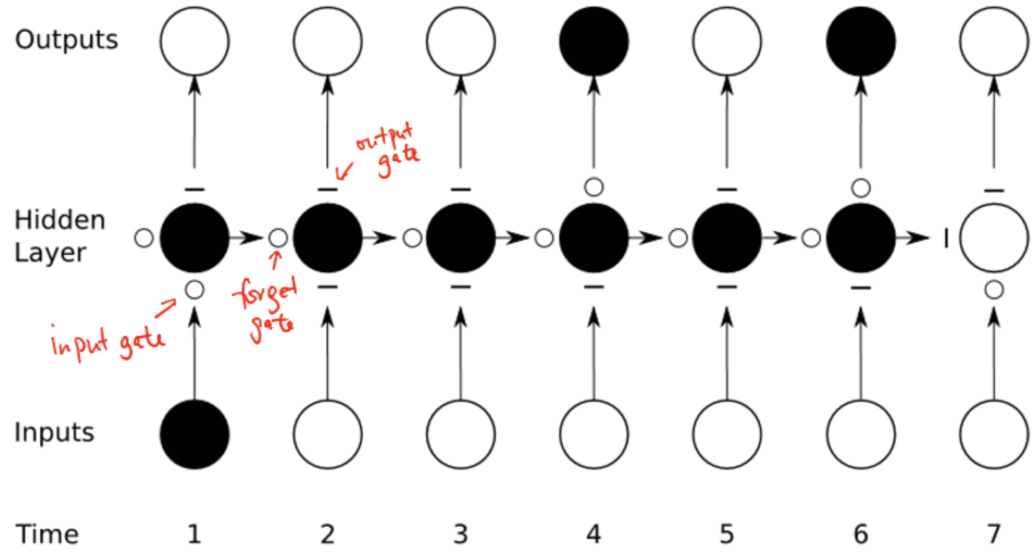
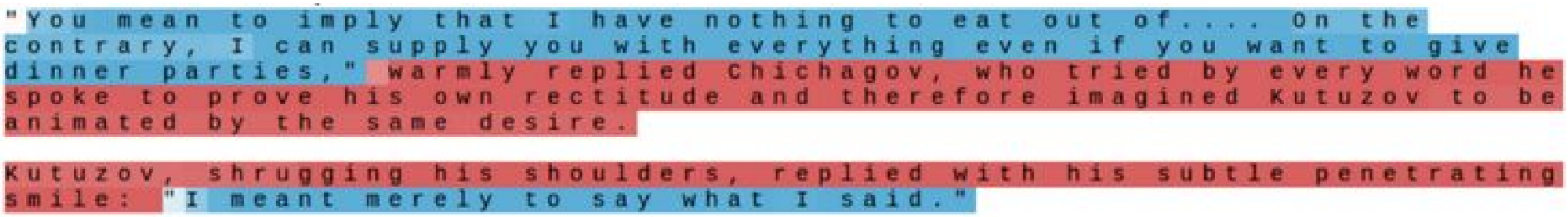


Figure 4.4: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Figure from Graves.   Supervised Sequence Labelling with Recurrent Neural Networks

Visualizing LSTM
LSTM可视化

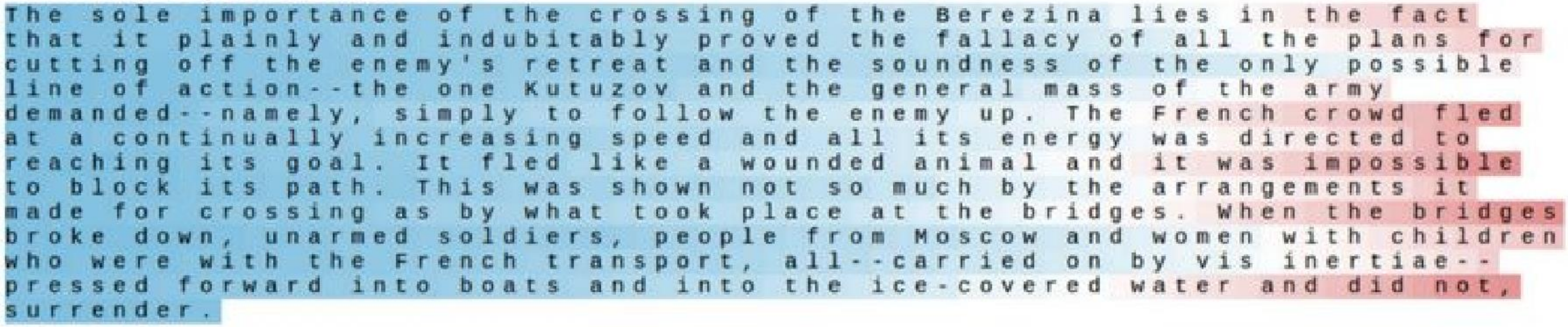


"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

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

quote detection cell

Cell sensitive to position in line:

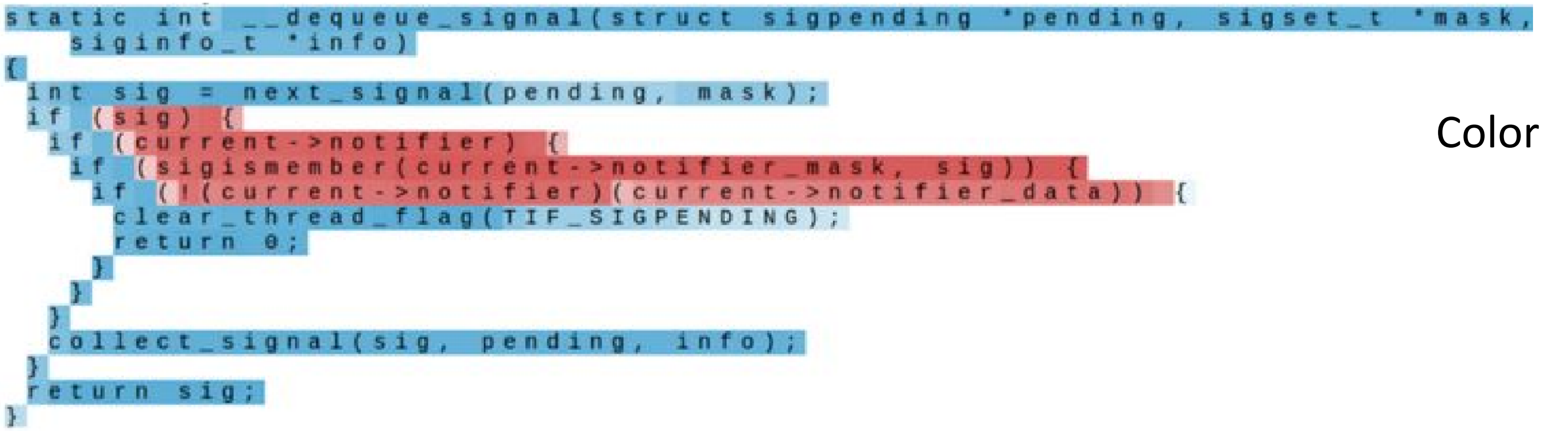


The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply 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.

line length tracking cell

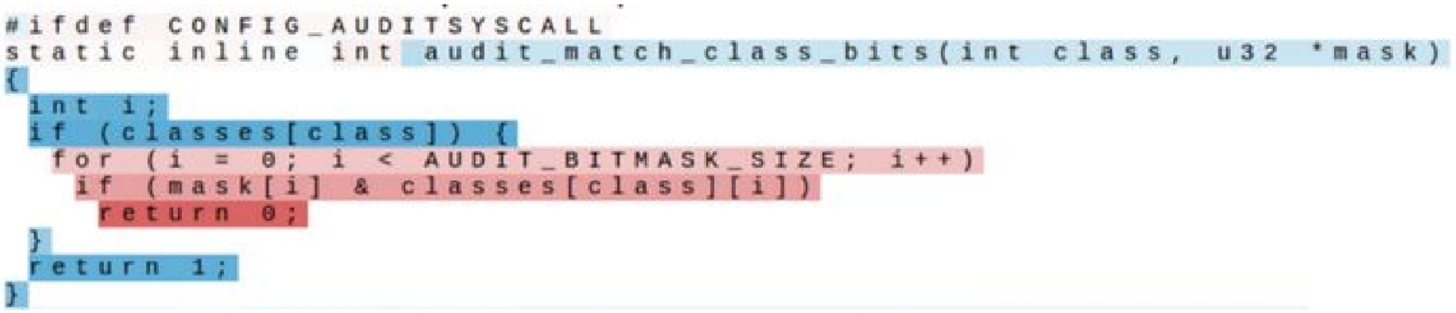
Visualizing LSTM

Color: cell state



```
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



```
#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

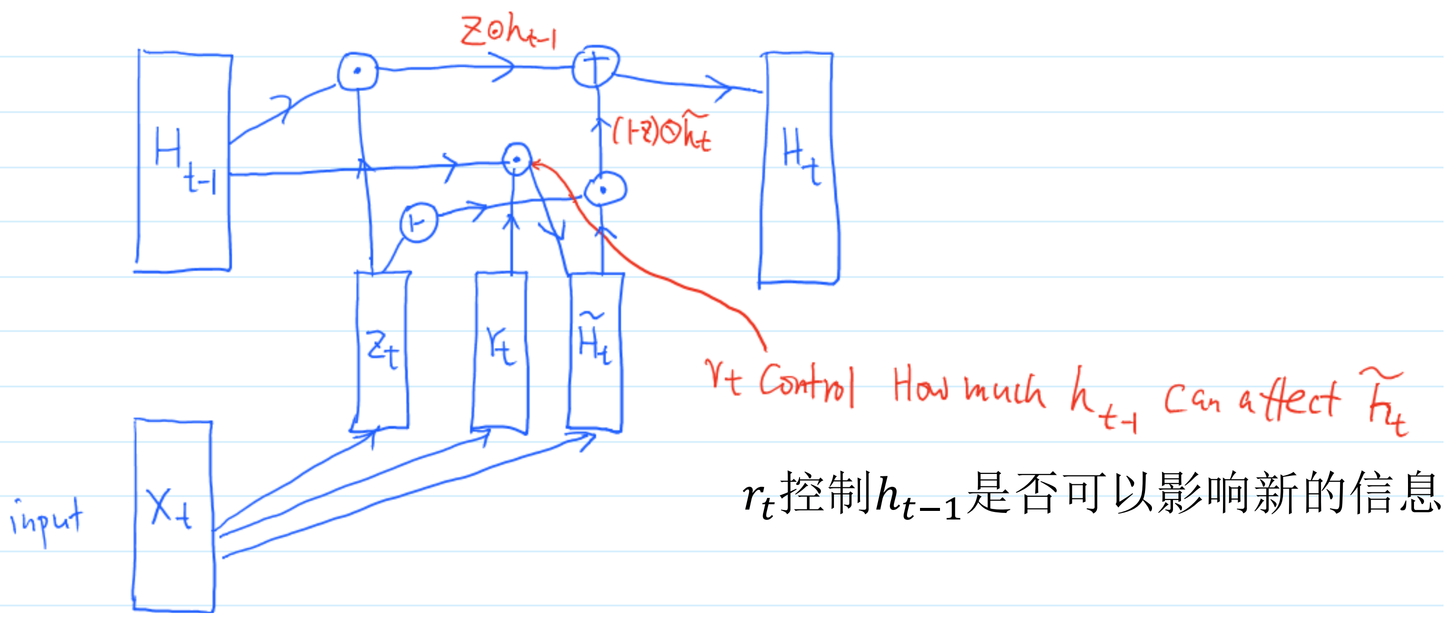
- Keras Code

```
keras.layers.recurrent.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', unit_forget_bias=True,
kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)
```

- 参数见：
  https://keras.io/layers/recurrent/

GRU (Gated Recurrent Unit)

GRU (Gated Recurrent Unit)



r_t控制h_{t-1}是否可以影响新的信息$\tilde{h}_t$

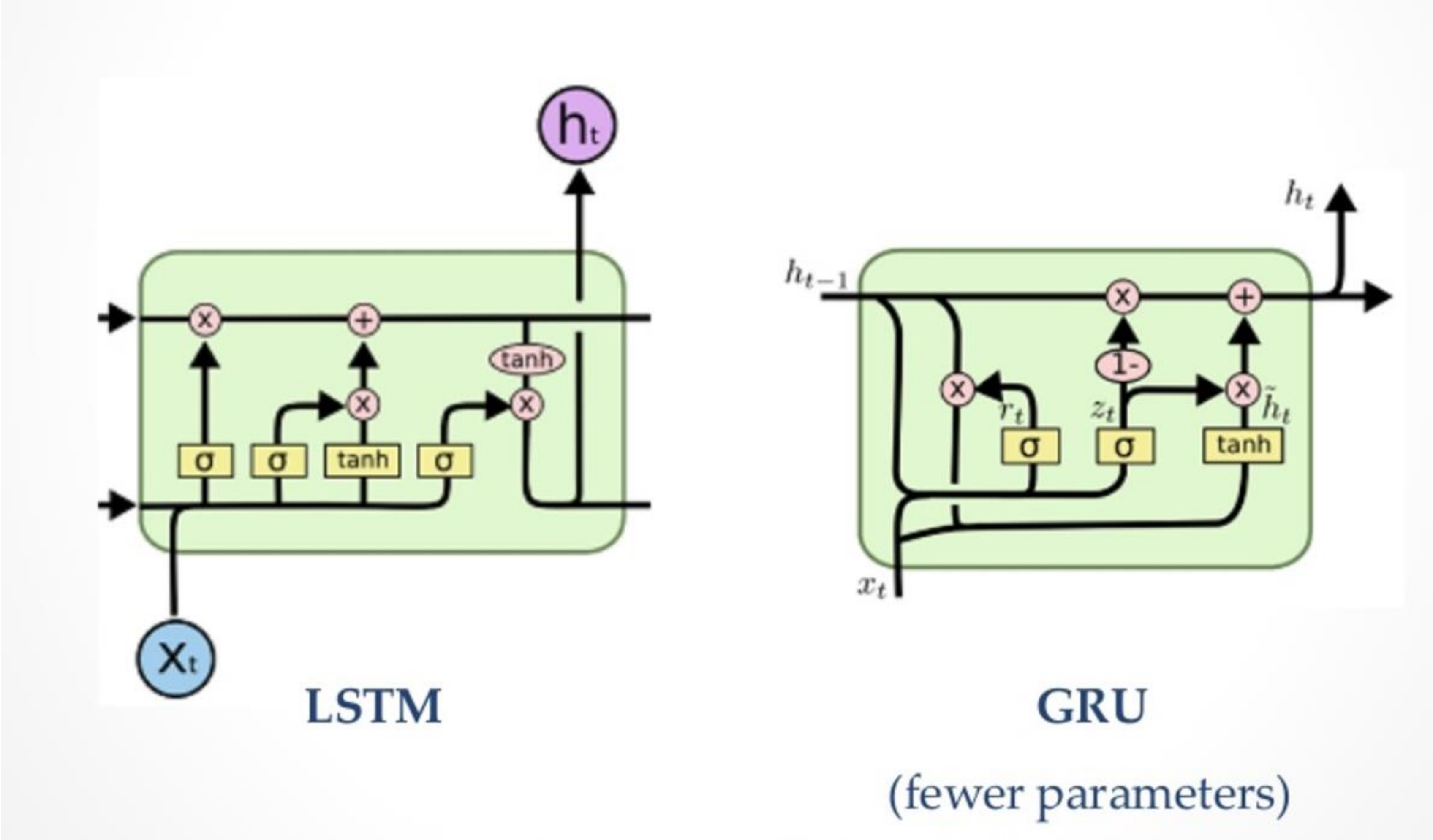
- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$
  If reset gate unit is ~0, then this ignores previous memory and only stores the new word information

如果reset gate是0，则忽略过去的记忆，只记录新的信息

- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

最终记忆结合了当前的记忆和以前的记忆

LSTM and GRU



h_t
tanh
x
+
σ
σ
tanh
σ
X_t
h_t
h_{t-1}
x
+
1-
r_t
z_t
h̃_t
σ
σ
tanh
x_t
LSTM
GRU
(fewer parameters)

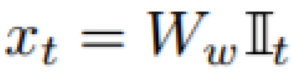
- Keras Code

```
keras.layers.recurrent.GRU(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', kernel_regularizer=None,
recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)
```

- 参数见：
  https://keras.io/layers/recurrent/

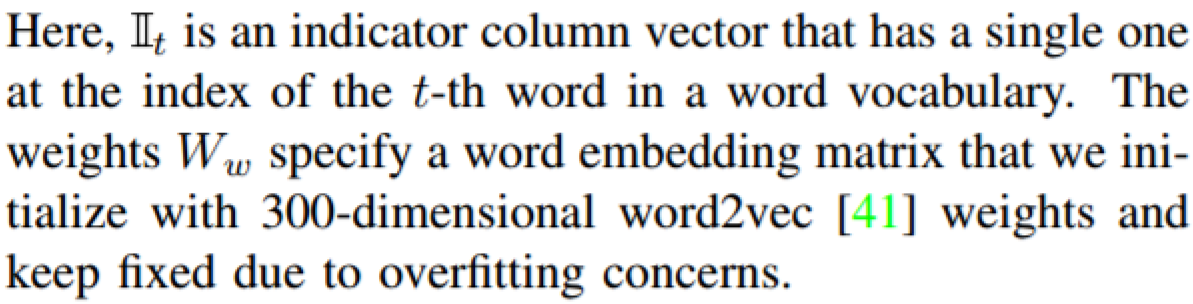
Embedding 嵌入

- Embedding: Embed a high-dim vector to a low dim space 将高维向量映射成低维向量
- We first embed each word to a short vector as follows:



$$x_t = W_w \mathbb{I}_t$$

One-hot vector (0,0,1,0,….,0,0)



Here, $\mathbb{I}_t$ is an indicator column vector that has a single one at the index of the t-th word in a word vocabulary. The weights W_w specify a word embedding matrix that we initialize with 300-dimensional word2vec [41] weights and keep fixed due to overfitting concerns.

- 一般一个词可以表示成一个one-hot向量, 例如 (0,0,1,0,….,0,0)
- 但该向量维数太高，不宜直接喂给神经网络，所以一般要通过Embedding映射成低维向量
- 通常W初始化成Word2vec得到的结果

Word2Vec is a very popular idea in natural language processing. Check it out for yourself.

Man - Woman + King = ? Answer: Queue.

Embedding 嵌入

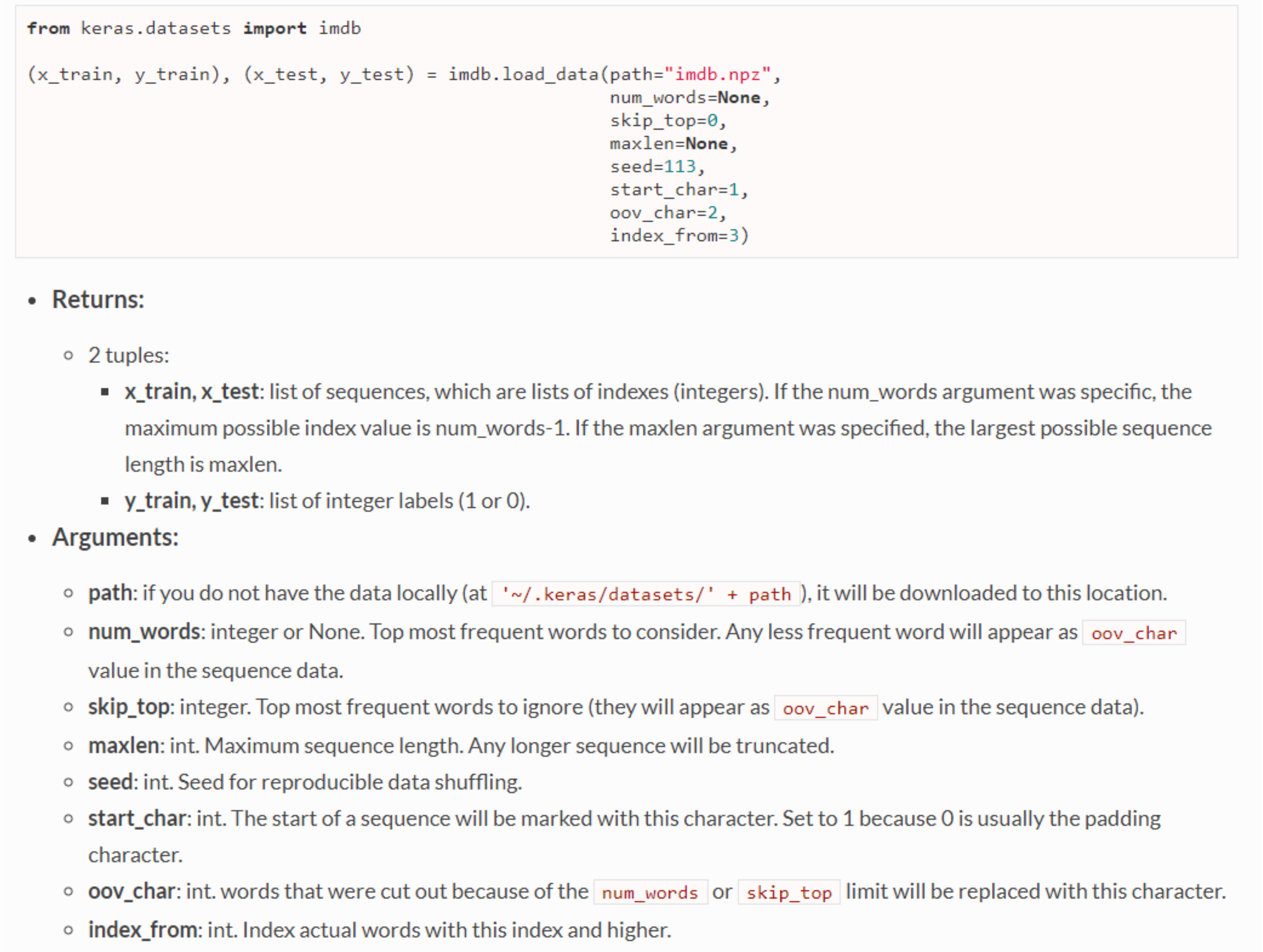
- https://keras.io/layers/embeddings/

```
keras.layers.embeddings.Embedding(input_dim, output_dim, embeddings_initializer='uniform',
embeddings_regularizer=None, activity_regularizer=None, embeddings_constraint=None, mask_zero=False,
input_length=None)
```

Example Keras code

- https://keras.io/datasets/
- **IMDB Movie reviews sentiment classification**

- Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".
- As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

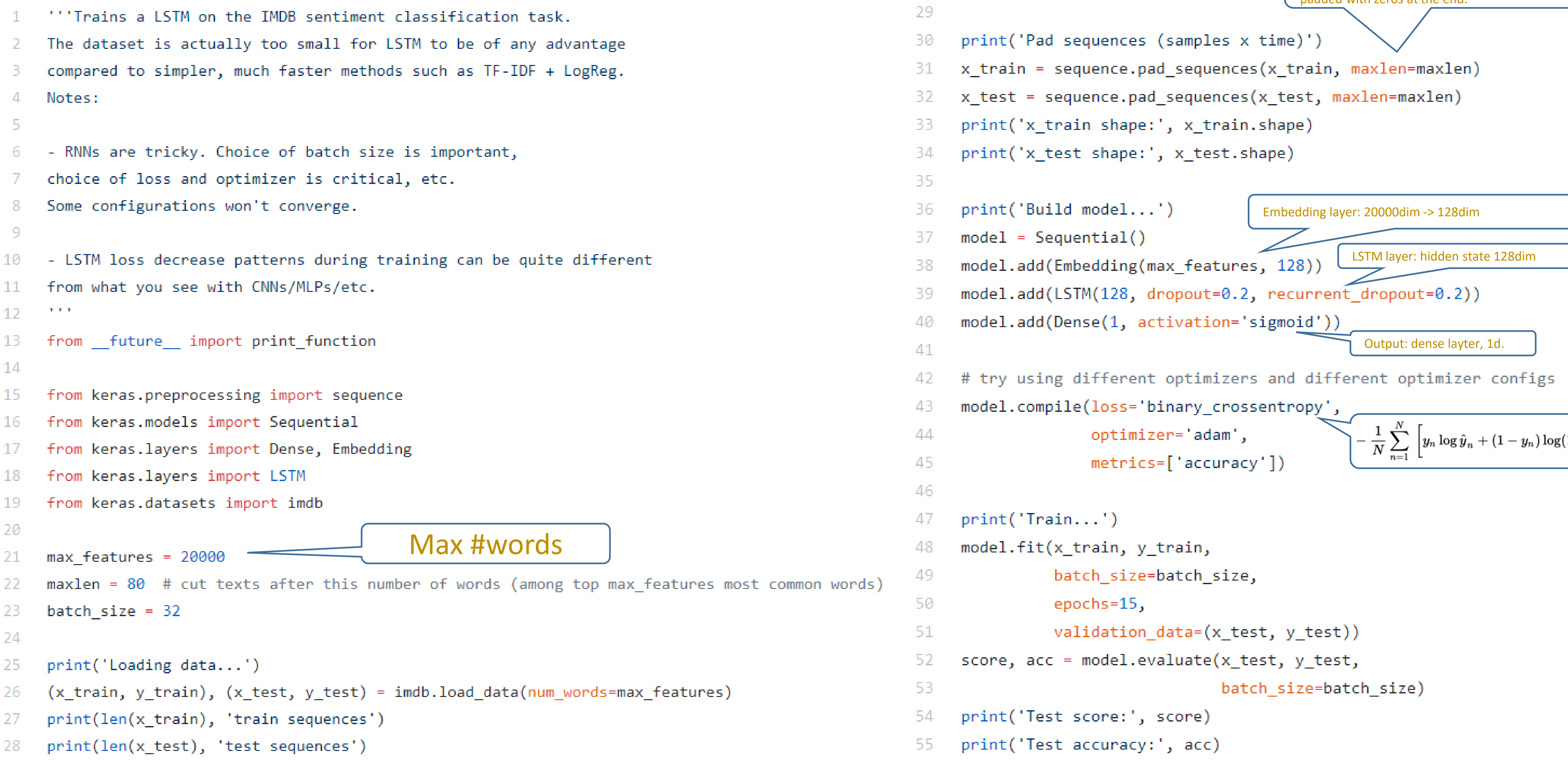


```
from keras.datasets import imdb

(x_train, y_train), (x_test, y_test) = imdb.load_data(path="imdb.npz",
                                                      num_words=None,
                                                      skip_top=0,
                                                      maxlen=None,
                                                      seed=113,
                                                      start_char=1,
                                                      oov_char=2,
                                                      index_from=3)
```

- **Returns:**
  - 2 tuples:
    - **x_train, x_test**: list of sequences, which are lists of indexes (integers). If the num_words argument was specific, the maximum possible index value is num_words-1. If the maxlen argument was specified, the largest possible sequence length is maxlen.
    - **y_train, y_test**: list of integer labels (1 or 0).
- **Arguments:**
  - **path**: if you do not have the data locally (at `'~/.keras/datasets/' + path`), it will be downloaded to this location.
  - **num_words**: integer or None. Top most frequent words to consider. Any less frequent word will appear as `oov_char` value in the sequence data.
  - **skip_top**: integer. Top most frequent words to ignore (they will appear as `oov_char` value in the sequence data).
  - **maxlen**: int. Maximum sequence length. Any longer sequence will be truncated.
  - **seed**: int. Seed for reproducible data shuffling.
  - **start_char**: int. The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character.
  - **oov_char**: int. words that were cut out because of the `num_words` or `skip_top` limit will be replaced with this character.
  - **index_from**: int. Index actual words with this index and higher.

Keras Code

https://github.com/fchollet/keras/blob/master/examples/imdb_lstm.py



```python
'''Trains a LSTM on the IMDB sentiment classification task.
The dataset is actually too small for LSTM to be of any advantage
compared to simpler, much faster methods such as TF-IDF + LogReg.
Notes:

- RNNs are tricky. Choice of batch size is important,
choice of loss and optimizer is critical, etc.
Some configurations won't converge.

- LSTM loss decrease patterns during training can be quite different
from what you see with CNNs/MLPs/etc.
'''
from __future__ import print_function

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb

max_features = 20000
maxlen = 80  # cut texts after this number of words (among top max_features most common words)
batch_size = 32

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print('Build model...')
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print('Train...')
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=15,
          validation_data=(x_test, y_test))
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size)
print('Test score:', score)
print('Test accuracy:', acc)
```

Max #words

maxlen: None or int. Maximum sequence length, longer sequences are truncated and shorter sequences are padded with zeros at the end.

Embedding layer: 20000dim -> 128dim

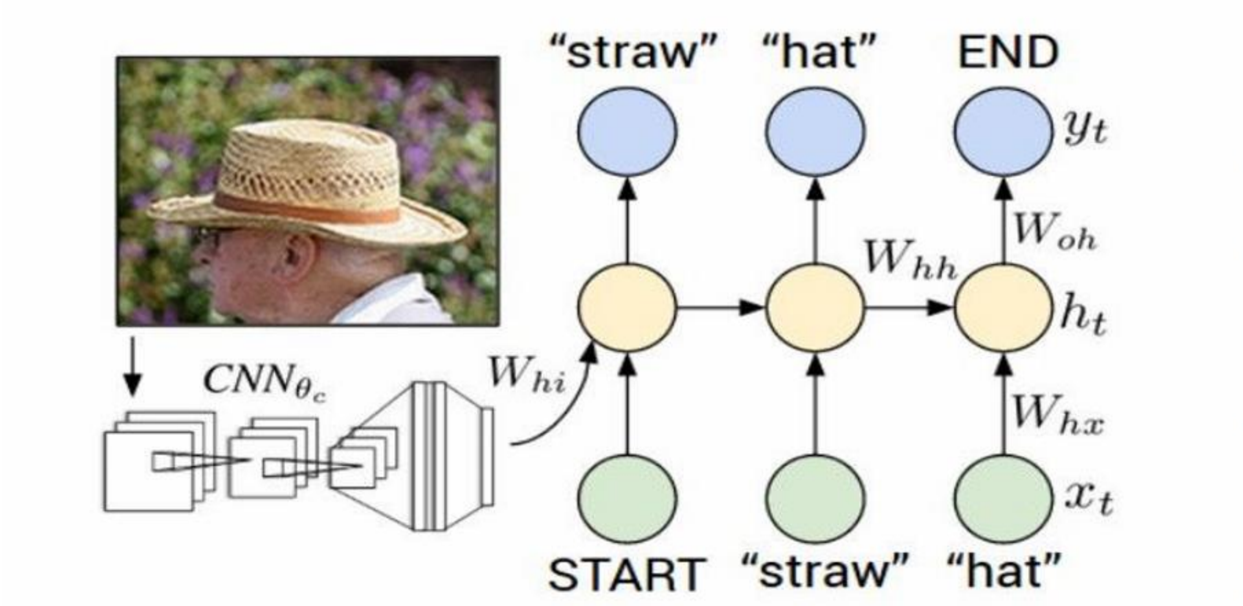
LSTM layer: hidden state 128dim

Output: dense layter, 1d.

$-\frac{1}{N}\sum_{n=1}^{N}\left[y_n\log\hat{y}_n+(1-y_n)\log(1-\hat{y}_n)\right]$

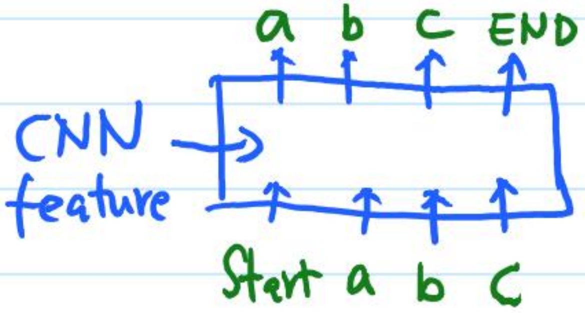
Application – Image Captioning
应用 – 生成图片说明

Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy, Li



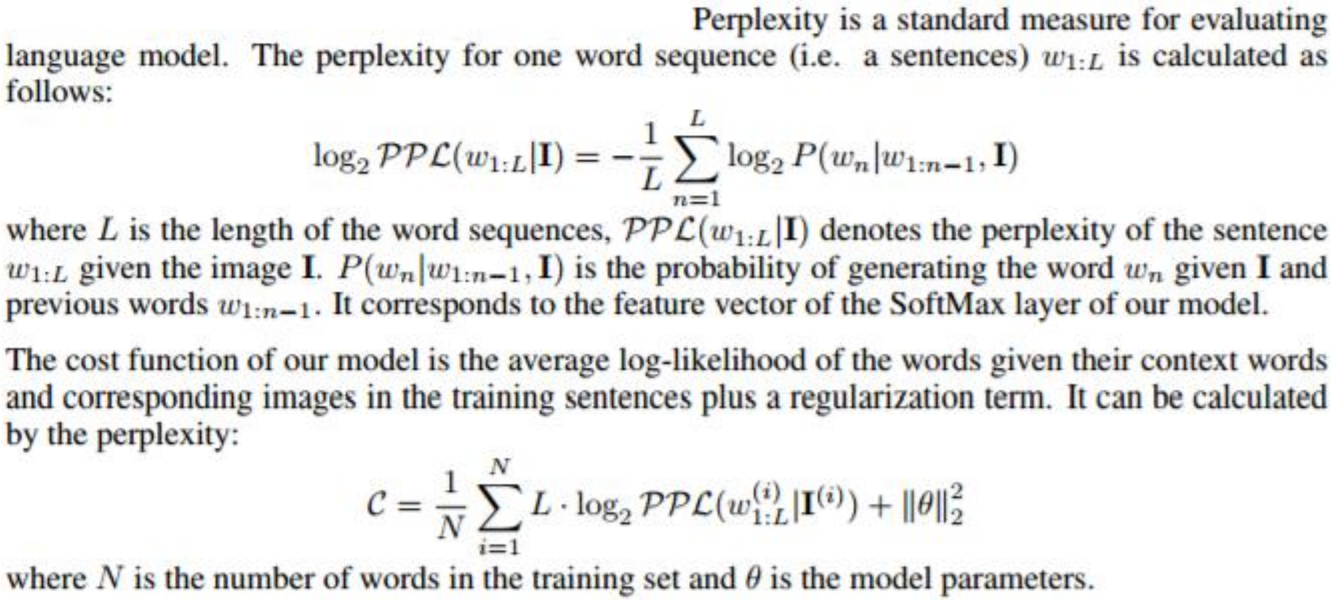
training such RNN is no different from the usual one

training data



Some details - Loss function for training

- One could use the cross entropy loss (treating the output, by softmax, as a classification problem, i.e., classifying the words) ---
  - Maximize the log probability assigned to the target labels (e.g., in Karpathy and Li)
  - Also called the perplexity measure (e.g., in Mao et al.)



Perplexity is a standard measure for evaluating language model. The perplexity for one word sequence (i.e. a sentences) $w_{1:L}$ is calculated as follows:

$$\log_2 \mathcal{PPL}(w_{1:L}|\mathbf{I}) = -\frac{1}{L}\sum_{n=1}^{L} \log_2 P(w_n|w_{1:n-1}, \mathbf{I})$$

where L is the length of the word sequences, $\mathcal{PPL}(w_{1:L}|\mathbf{I})$ denotes the perplexity of the sentence $w_{1:L}$ given the image $\mathbf{I}$. $P(w_n|w_{1:n-1}, \mathbf{I})$ is the probability of generating the word w_n given $\mathbf{I}$ and previous words $w_{1:n-1}$. It corresponds to the feature vector of the SoftMax layer of our model.

The cost function of our model is the average log-likelihood of the words given their context words and corresponding images in the training sentences plus a regularization term. It can be calculated by the perplexity:

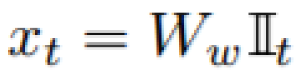
$$\mathcal{C} = \frac{1}{N}\sum_{i=1}^{N} L \cdot \log_2 \mathcal{PPL}(w_{1:L}^{(i)}|\mathbf{I}^{(i)}) + \|\theta\|_2^2$$

where N is the number of words in the training set and θ is the model parameters.

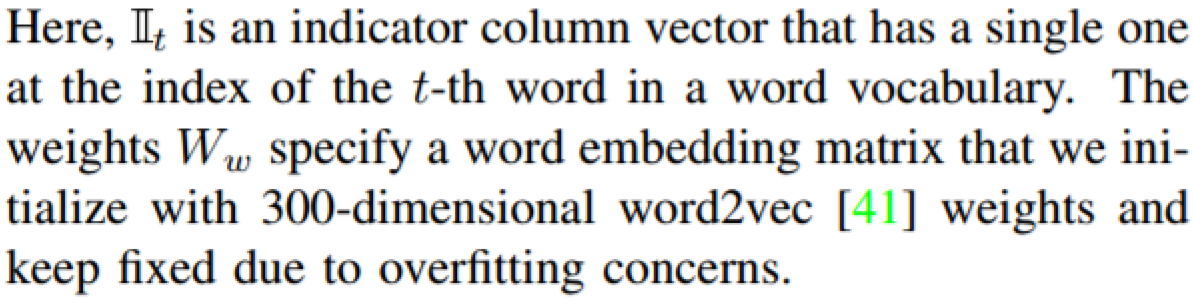
Mao et al. Explain Images with Multimodal Recurrent Neural Networks

Some Details - Embedding

- Embedding: We first embed each word to a short vector as follows:



$$x_t = W_w \mathbb{I}_t$$



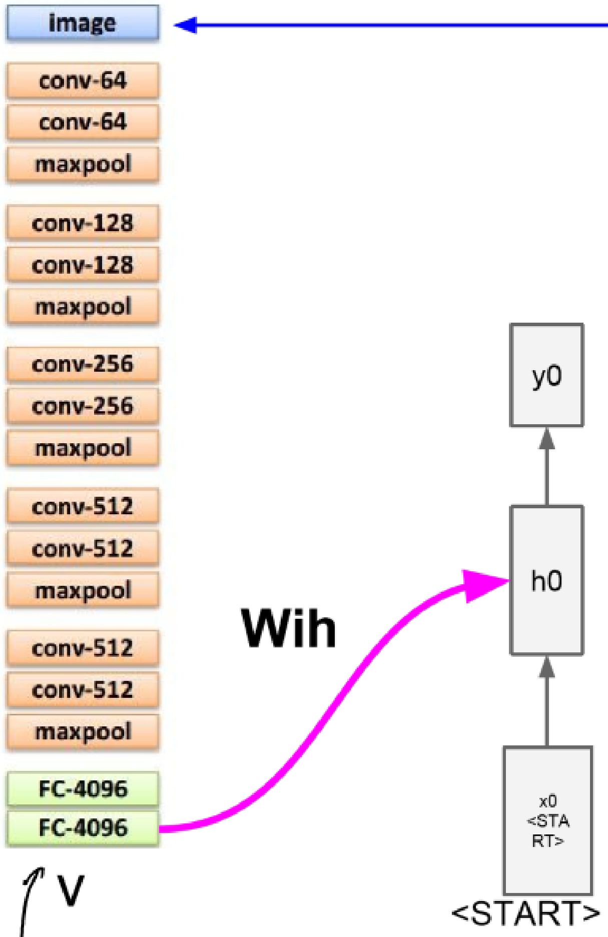
Here, $\mathbb{I}_t$ is an indicator column vector that has a single one at the index of the t-th word in a word vocabulary. The weights W_w specify a word embedding matrix that we initialize with 300-dimensional word2vec [41] weights and keep fixed due to overfitting concerns.

Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy, Li

Word2Vec is a very popular idea in natural language processing. Check it out for yourself.

Man - Woman + King = ? Answer: Queue.

TESTING PHASE:



test image

← start token

before:

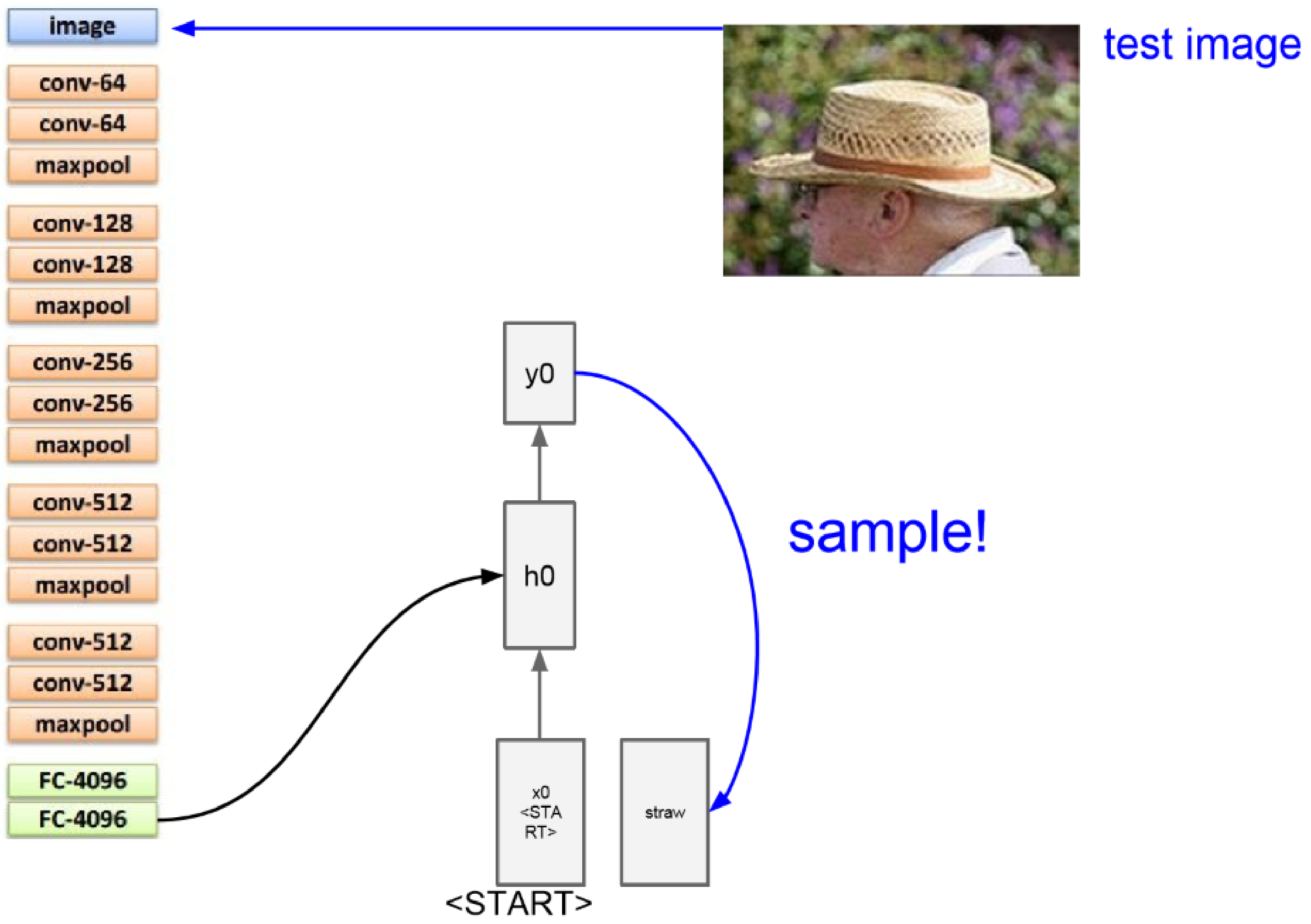
h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih** * v)

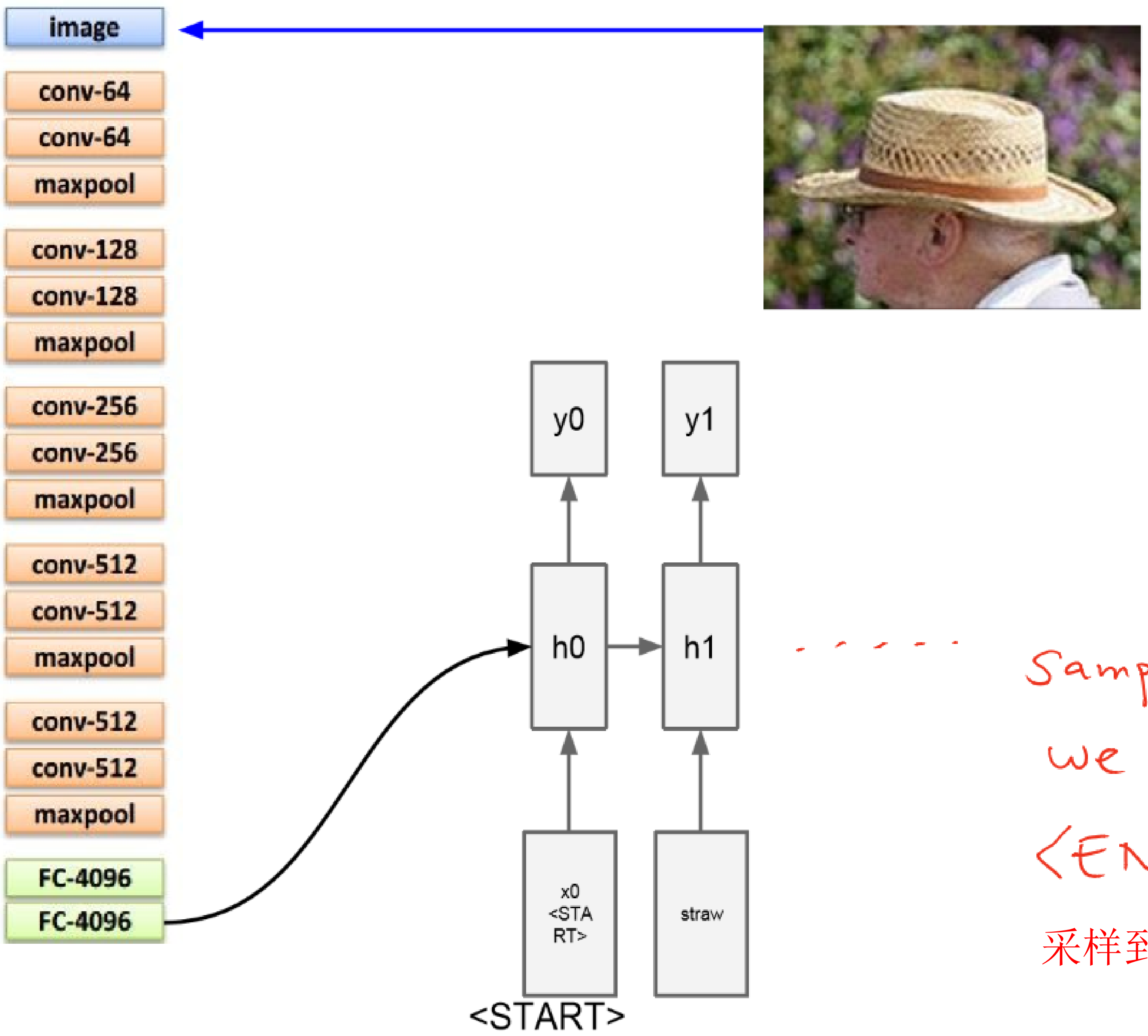
remove the last two layers of CNN

去掉CNN的最后两层（我们不需要用CNN做分类，只要取它高层的feature）



In the testing phase, we sample the output of the first time stemp. and feed it to RNN in the 2nd step

在testing阶段，我们对output采样，得到下一个词，然后将下一个词作为RNN的下一个输入

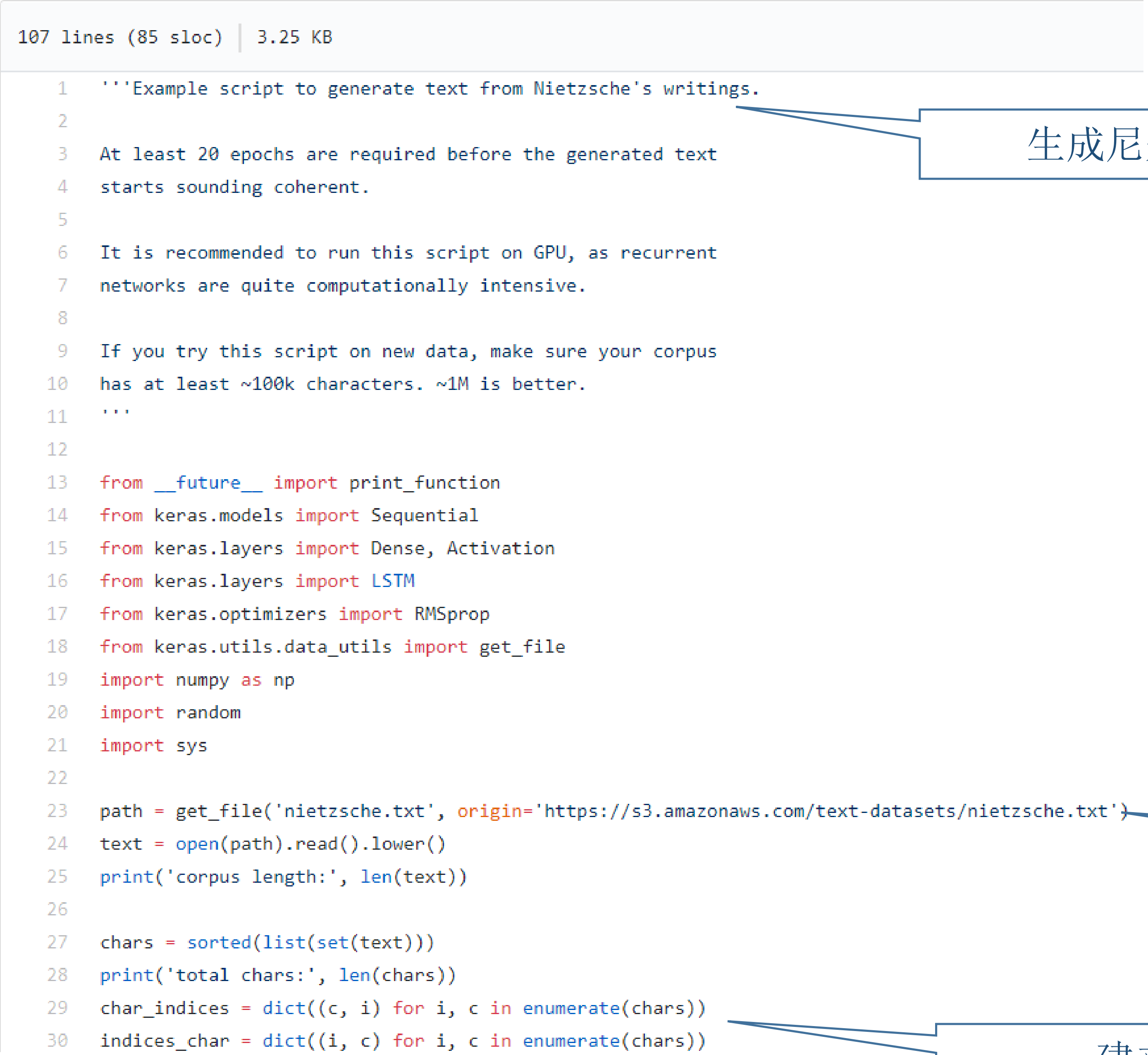


test image

Sample until we get the <END> token

采样到结束符<end>为止

Text Generation - Example
文本生成 – 代码示例



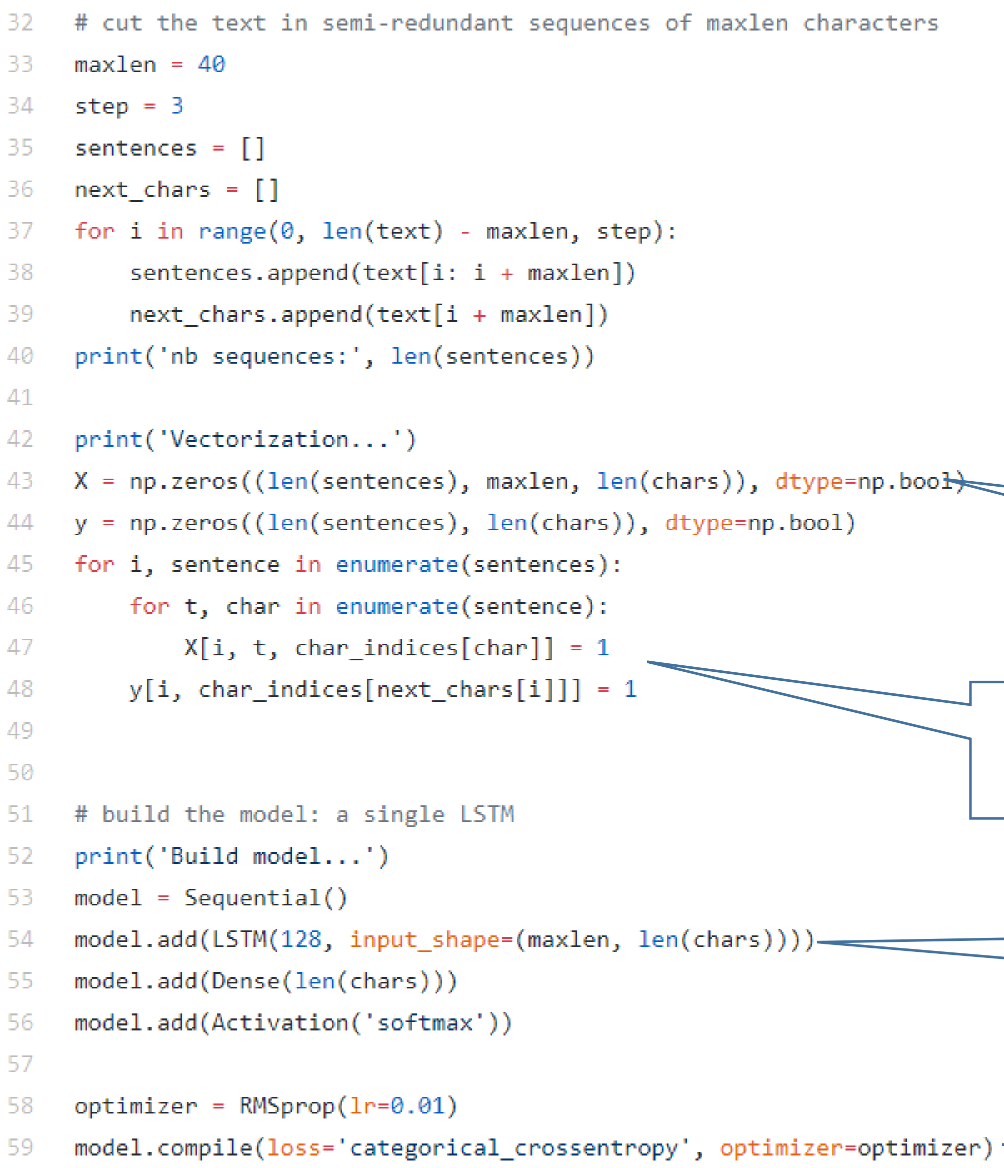
107 lines (85 sloc) | 3.25 KB

```python
'''Example script to generate text from Nietzsche's writings.

At least 20 epochs are required before the generated text
starts sounding coherent.

It is recommended to run this script on GPU, as recurrent
networks are quite computationally intensive.

If you try this script on new data, make sure your corpus
has at least ~100k characters. ~1M is better.
'''

from __future__ import print_function
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
import random
import sys

path = get_file('nietzsche.txt', origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt')
text = open(path).read().lower()
print('corpus length:', len(text))

chars = sorted(list(set(text)))
print('total chars:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))
```

生成尼采的作品

读取尼采的文本作为训练数据

建立词的字典



```python
# cut the text in semi-redundant sequences of maxlen characters
maxlen = 40
step = 3
sentences = []
next_chars = []
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('nb sequences:', len(sentences))

print('Vectorization...')
X = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        X[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1


# build the model: a single LSTM
print('Build model...')
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

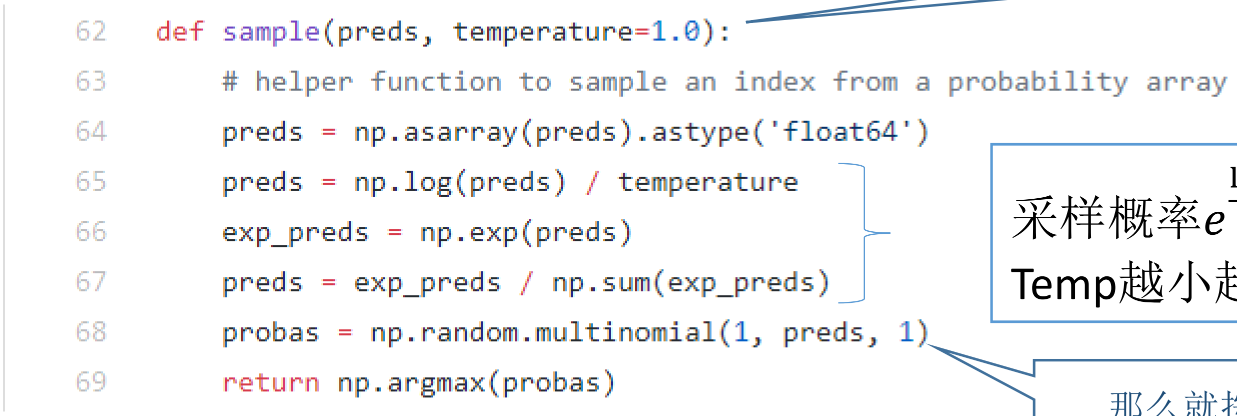
建立训练数据 X, y
X: 3-d tensor (#句子，句子长度，#词)
Y: 2-d tensor (#句子， #词)

建立训练数据 X, y
都使用1-hot vector

建立网络，一层LSTM，一层Dense，最后输出是softmax，维数为#词 (即len(chars))

指定loss function和优化方法

子过程，采样下一个词

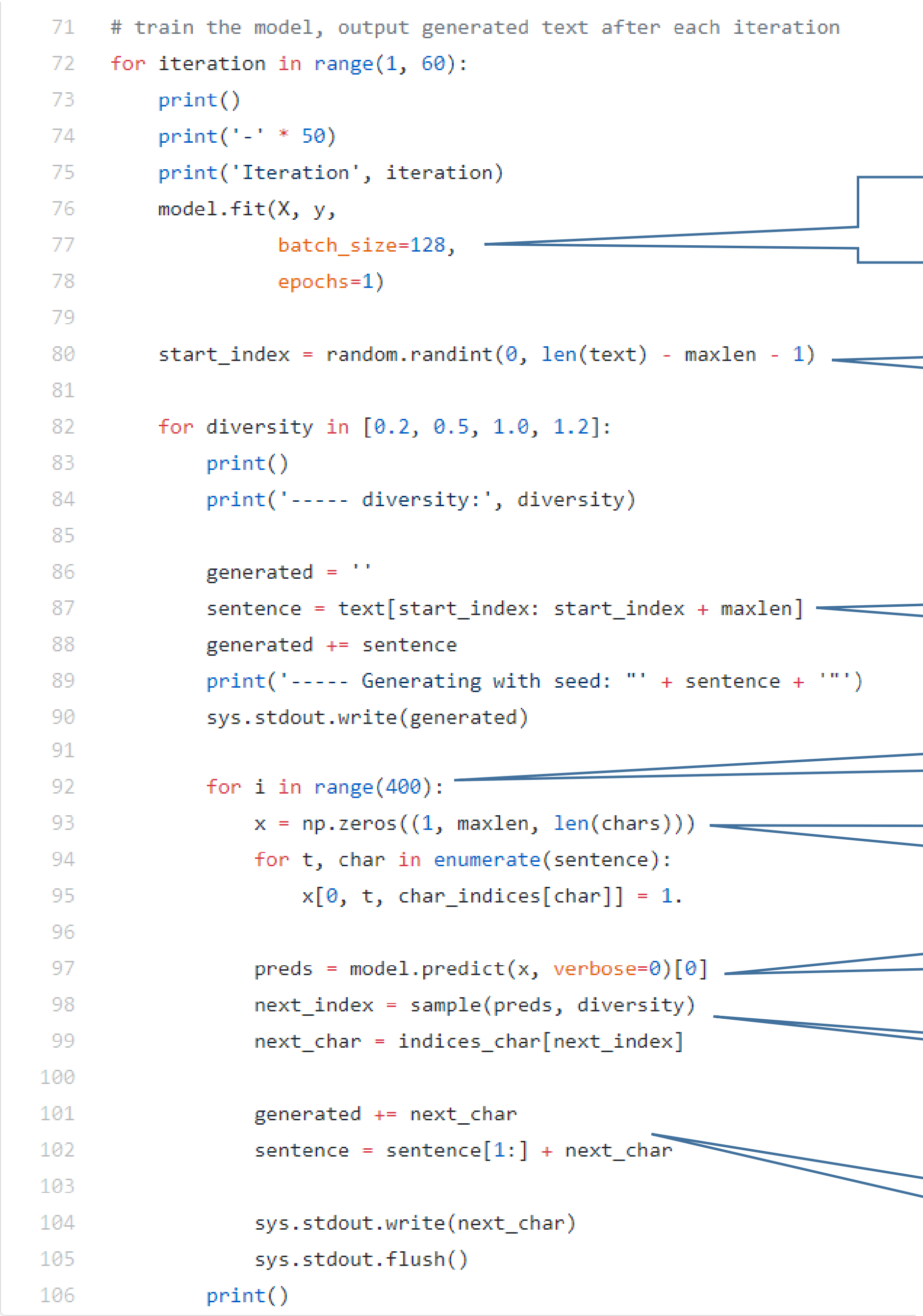


```python
def sample(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

Preds是一个prob vector，
Temperature控制采样的平均程度
如果Temperature=1，那么就按preds指定的概率采样

采样概率$e^{\frac{\log a_i}{T}} / \sum_j e^{\frac{\log a_j}{T}}$
Temp越小越平均

那么就按preds指定的概率采样1次



```python
# train the model, output generated text after each iteration
for iteration in range(1, 60):
    print()
    print('-' * 50)
    print('Iteration', iteration)
    model.fit(X, y,
              batch_size=128,
              epochs=1)

    start_index = random.randint(0, len(text) - maxlen - 1)

    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print()
        print('----- diversity:', diversity)

        generated = ''
        sentence = text[start_index: start_index + maxlen]
        generated += sentence
        print('----- Generating with seed: "' + sentence + '"')
        sys.stdout.write(generated)

        for i in range(400):
            x = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x[0, t, char_indices[char]] = 1.

            preds = model.predict(x, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]

            generated += next_char
            sentence = sentence[1:] + next_char

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print()
```

训练模型

生成文本的起始词，在text中的位置

Sentence是在text中的一句话，起始词是start_index

接着目前的sentence，再生成400个词

X是目前sentence的1-hot vector 表示（可以堪称maxlen个dim=len(chars)的1-hot vectors）

Preds是模型的预测，即网络的输出 （dim=len(chars)）

按preds向量制定的概率采样下一个词

将下一个词拼到目前生成的句子上

- Awesome RNN: a lot of useful references
    - https://github.com/kjw0612/awesome-rnn

- Some slides borrowed from cs231n, cs224d at Stanford

http://cs231n.stanford.edu/syllabus.html

http://cs224d.stanford.edu/index.html