Deep Learning 3

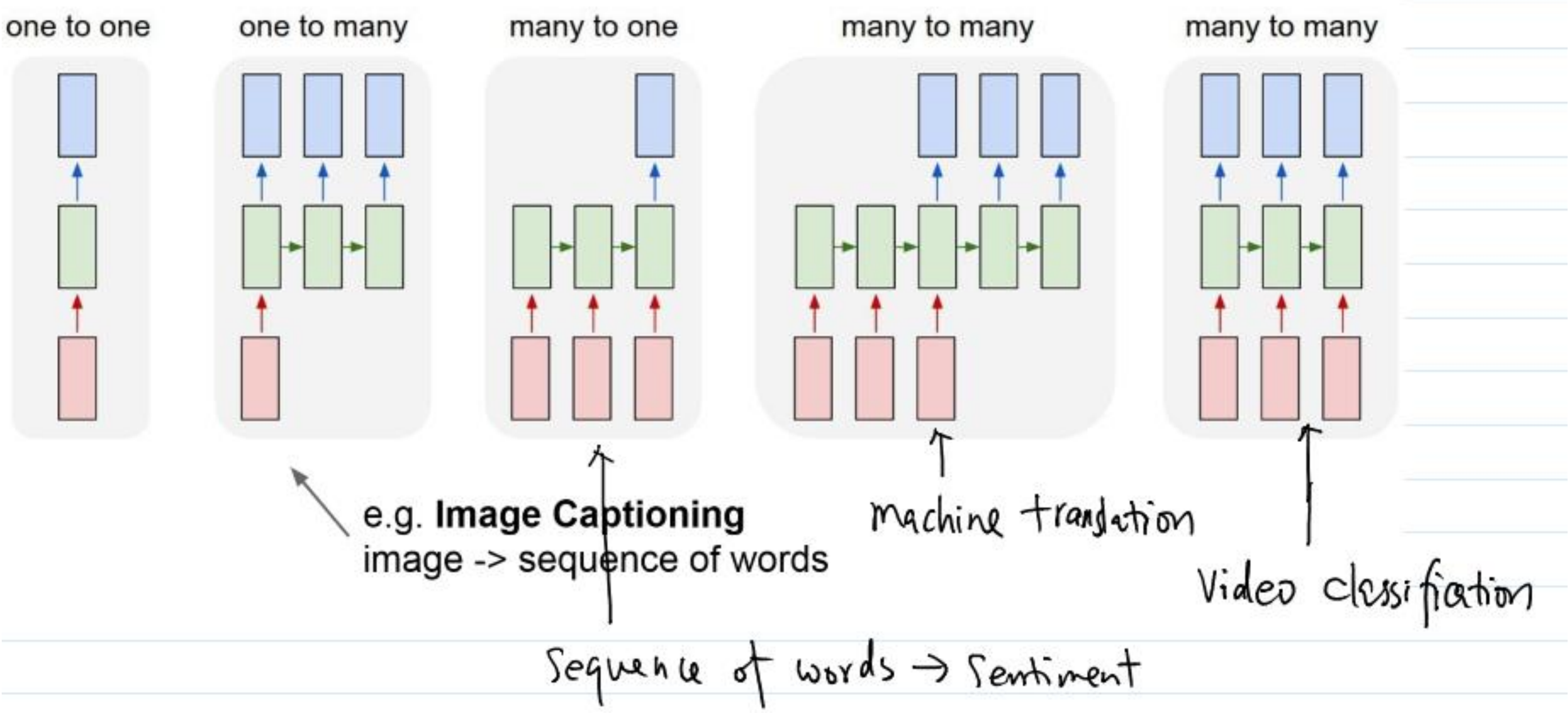
Jian Li

IIIS, Tsinghua

Recurrent Neural Networks (RNN)

CNN: parameter sharing in space
RNN: parameter sharing in time (suitable for sequences, in particular sequences with variable lengths)

Basics

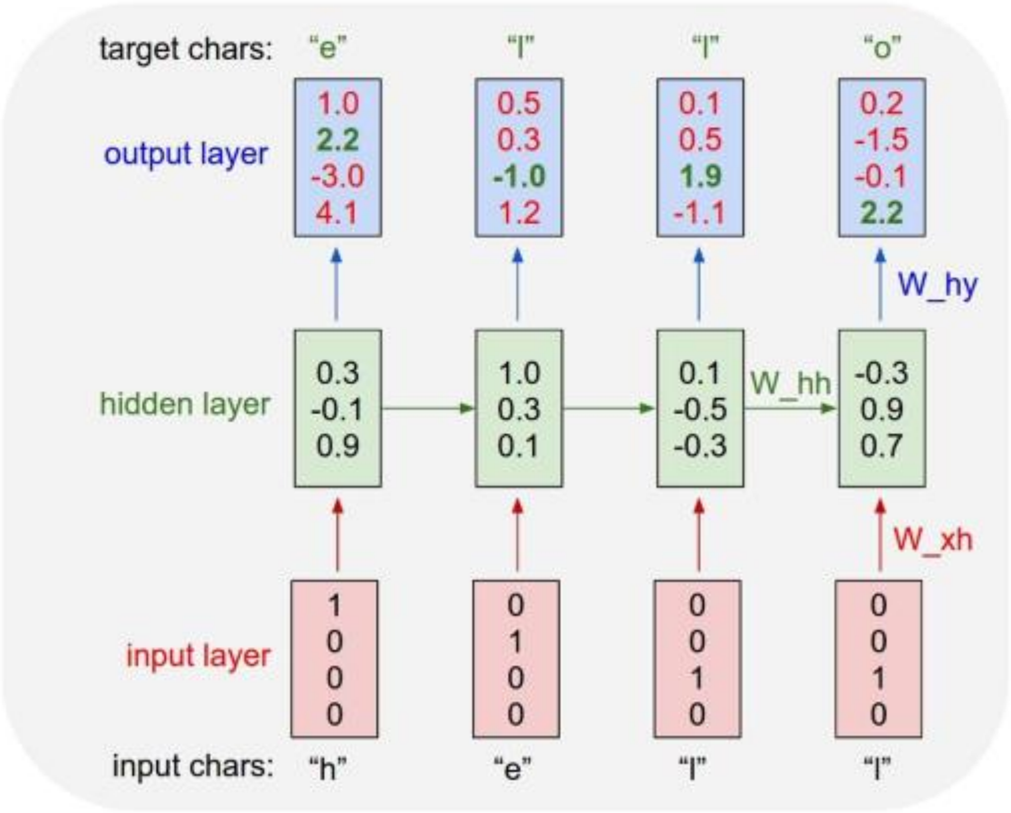

one to one
one to many
many to one
many to many
many to many
e.g. **Image Captioning**
image -> sequence of words
machine translation
Video classification
Sequence of words → Sentiment

Basics

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



- Learns time dependency gradually:

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

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

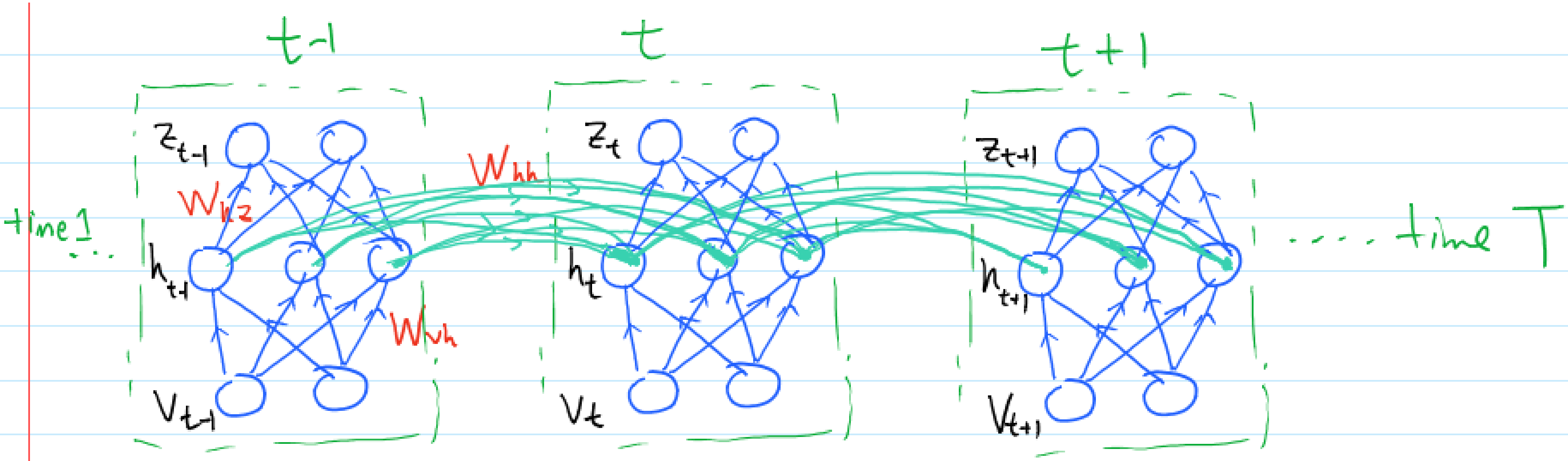
$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

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

Generated C code

```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 ");
}
```

Unroll/Unfold a RNN in time



$u_t \leftarrow W_{vh} \cdot v_t + W_{hh} \cdot h_{t-1} + b_h$ $\quad h_t \leftarrow \delta_1(u_t)$

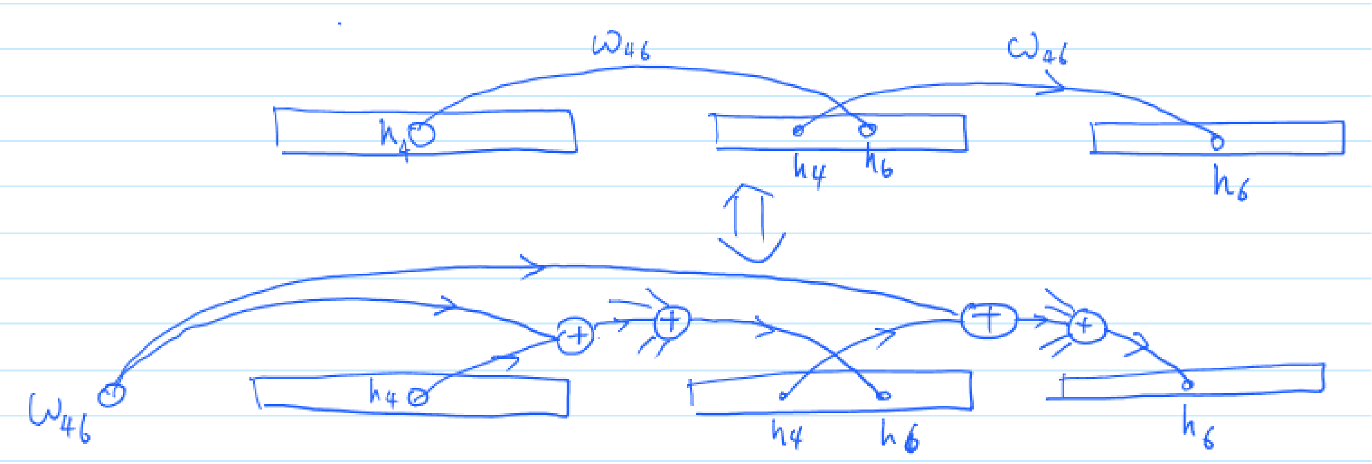
nonlinearity

$o_t \leftarrow W_{ho} \cdot h_t + b_o$ $\quad z_t \leftarrow \delta_2(o_t)$

☆ weight W_{hv}, W_{ho}, W_{hh} are shared across diff time steps

BPTT

- Backprop thru time t=T, T-1,….,2,1
- The weight variables w are shares across all time steps.
- So in backprop, they need to be imcremented when the grad flows back each time step



BPTT

- Backprop thru time

t=T,T-1,….,2,1

for $t = T \cdots 1$

$$\frac{\partial L}{\partial o_t} = \frac{\partial L(z_t, y_t)}{\partial z_t} \cdot \frac{\partial z_t}{\partial o_t}$$

elementwise; $\frac{\partial L}{\partial o_t}$: vec, $\frac{\partial L(z_t, y_t)}{\partial z_t}$: vec, $\frac{\partial z_t}{\partial o_t}$: g_2'

$(z_t = g_2(o_t))$

$$\oplus \quad \frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial b_o} + \frac{\partial L}{\partial o_t}$$

increment $\partial L / \partial b_o$

$(o_t = W_{ho} h_t + b_o)$

$$\oplus \quad \frac{\partial L}{\partial W_{ho}} \leftarrow \frac{\partial L}{\partial W_{ho}} + \frac{\partial L}{\partial o_t} \cdot h_t^T$$

matrix

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_t} + W_{ho}^T \frac{\partial L}{\partial o_t}$$

$$\frac{\partial L}{\partial \mu_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial \mu_t}$$

$\frac{\partial h_t}{\partial \mu_t}$: g_1'

$$\oplus \quad \frac{\partial L}{\partial W_{vh}} = \frac{\partial L}{\partial W_{vh}} + \frac{\partial L}{\partial \mu_t} \cdot v_t^T$$

$(\mu_t \leftarrow W_{vh} v_t + W_{hh} h_{t-1} + b_h)$

$$\oplus \quad \frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial W_{hh}} + \frac{\partial L}{\partial \mu_t} \cdot h_{t-1}^T$$

$$\oplus \quad \frac{\partial L}{\partial b_h} = \frac{\partial L}{\partial b_h} + \frac{\partial L}{\partial \mu_t}$$

$$\frac{\partial L}{\partial h_{t-1}} = W_{hh}^T \frac{\partial L}{\partial \mu_t}$$

$\oplus$ shared variable

in each iteration, the gradient need to be incremented.

Bi-directional RNN

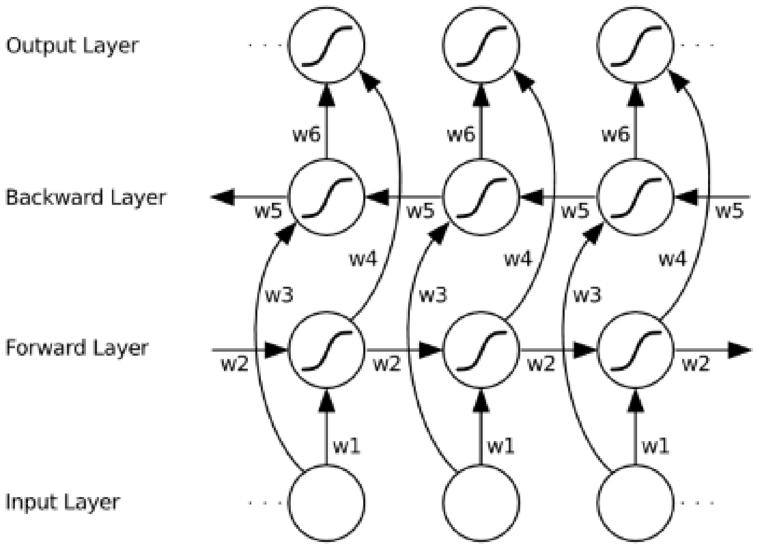


Figure 3.5: **An unfolded bidirectional network.** Six distinct sets of weights are reused at every timestep, corresponding to the input-to-hidden, hidden-to-hidden and hidden-to-output connections of the two hidden layers. Note that no information flows between the forward and backward hidden layers; 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 δ terms at each timestep
for $t = T$ to 1 **do**
  BPTT backward pass for the forward hidden layer, using the stored δ terms from the output layer
for $t = 1$ to T **do**
  BPTT backward pass for the backward hidden layer, using the stored δ terms from the output layer

Algorithm 3.2: BRNN Backward Pass

Figure from Graves. Supervised Sequence Labelling with Recurrent Neural Networks

Gradient Vanishing/Exploding problem

```python
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]) ← inner prod
    hs[t] = np.maximum(0, ss[t]) ← ReLU

# 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 ← BP thru ReLU
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state ← BP thru inner prod.
```

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

can control exploding with gradient clipping
can control vanishing with LSTM

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

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:

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

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:

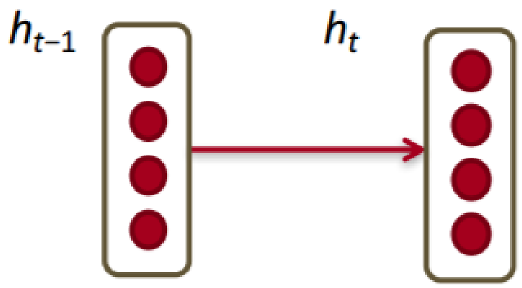
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

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

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: $h_t = Wf(h_{t-1}) + W^{(hx)}x_{[t]}$

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

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

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

Check at home that you understand the diag matrix formulation

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

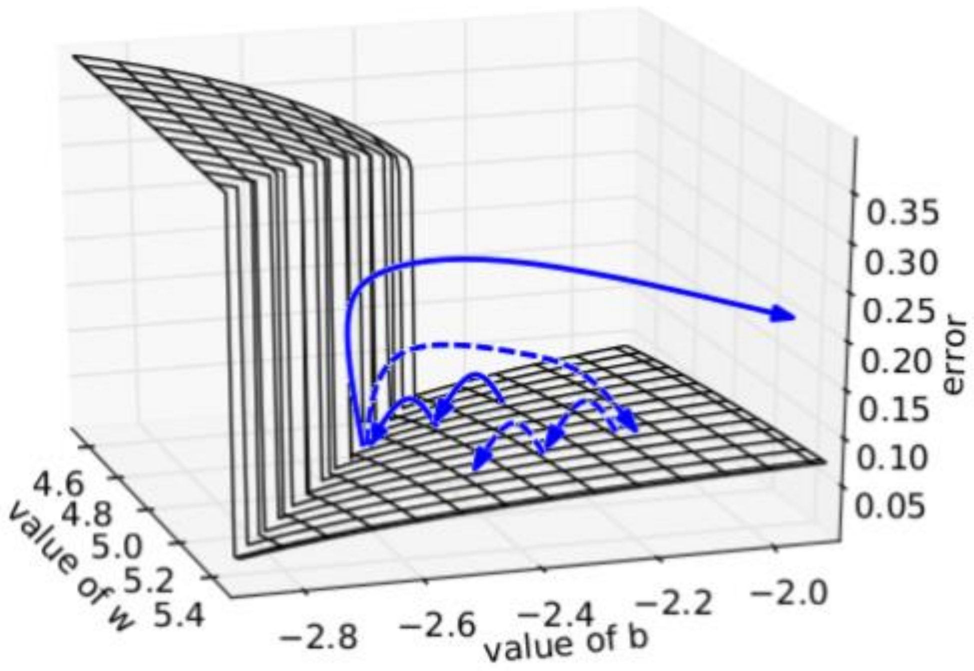
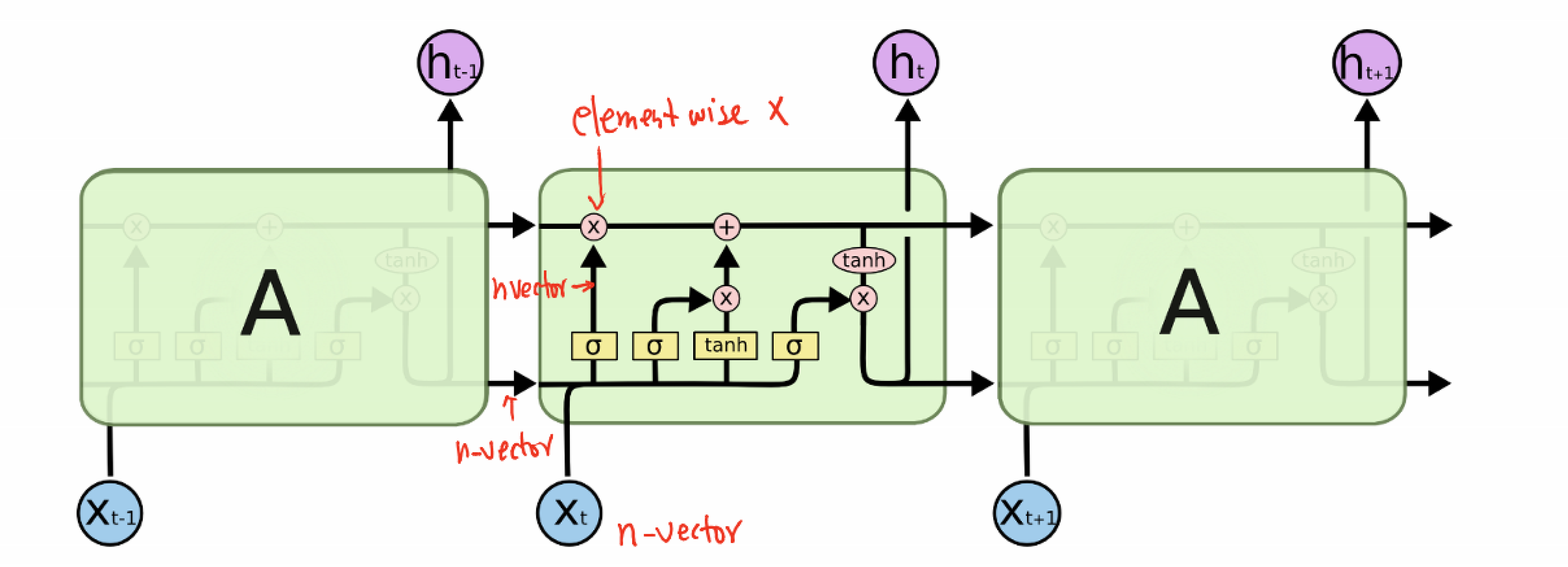


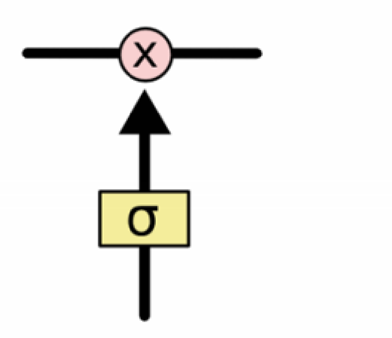
Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depicts standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a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

Long Short Term Memory (LSTM)

Overview



h t-1
h t
h t+1
element wise x
n vector
A
tanh
σ
σ
tanh
σ
A
n-vector
X t-1
X t
n-vector
X t+1

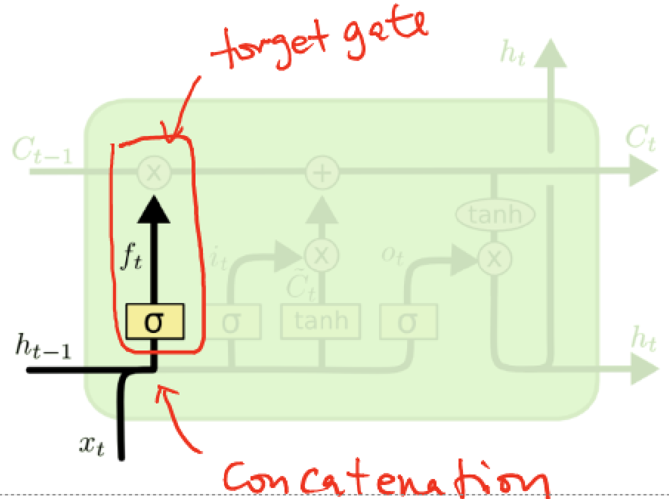


gate : [σ] outputs a number in [0,1] (or a vector)

the number decides how much information passes thru

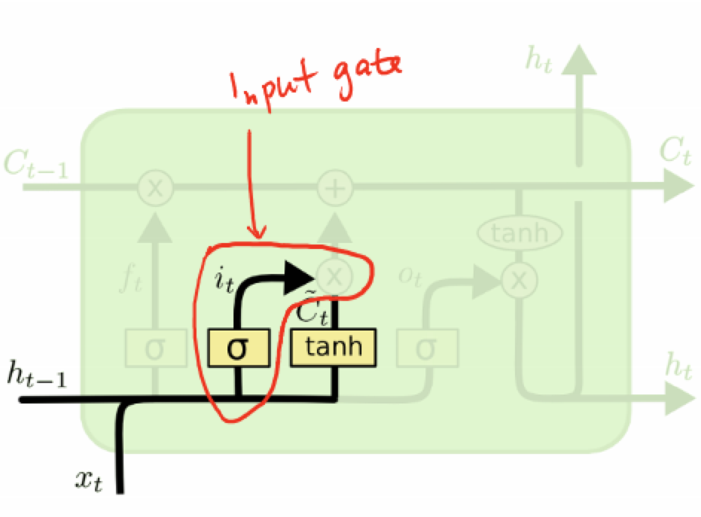
C_t : cell state

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



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

Input gate



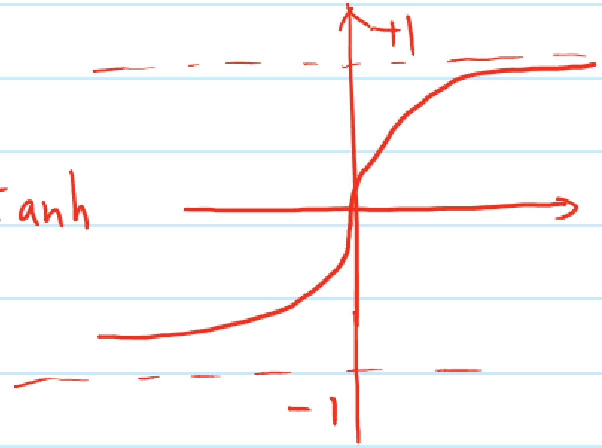
input gate ↓

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

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

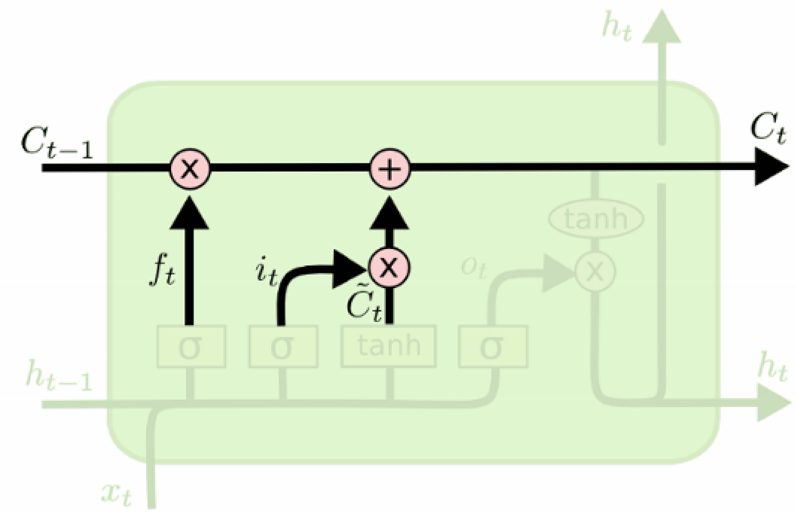
↑ new cell state

tanh



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

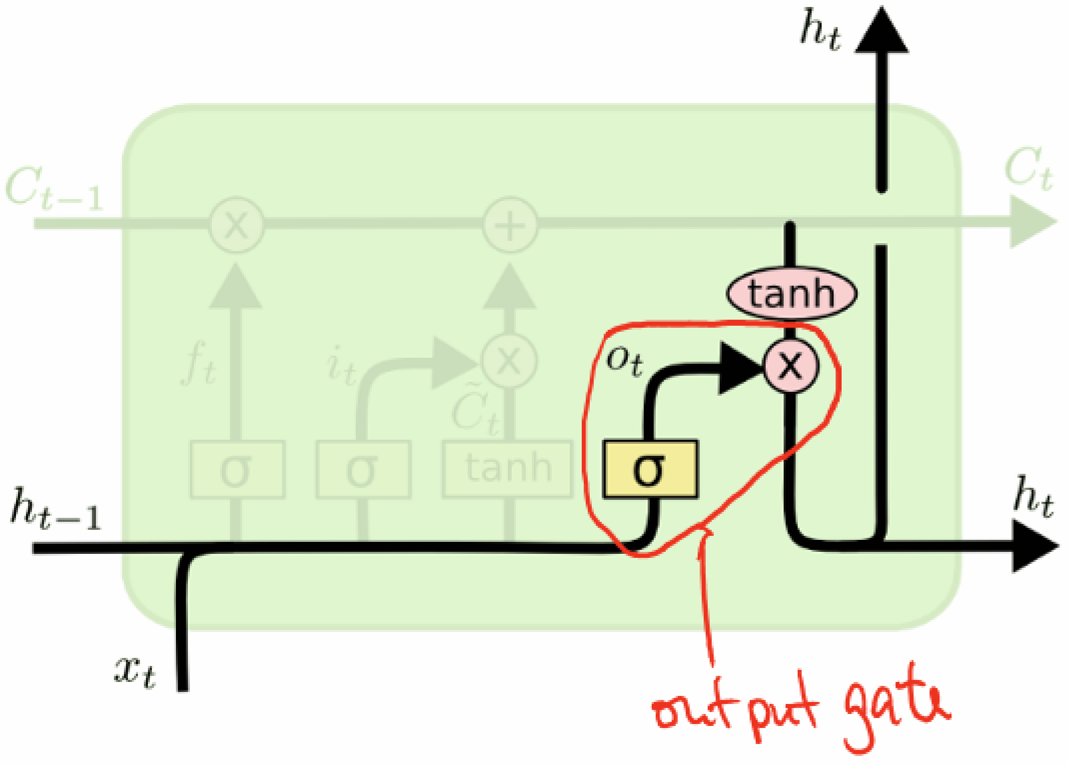
update the cell state C_t



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

↑ new cell state

output gate

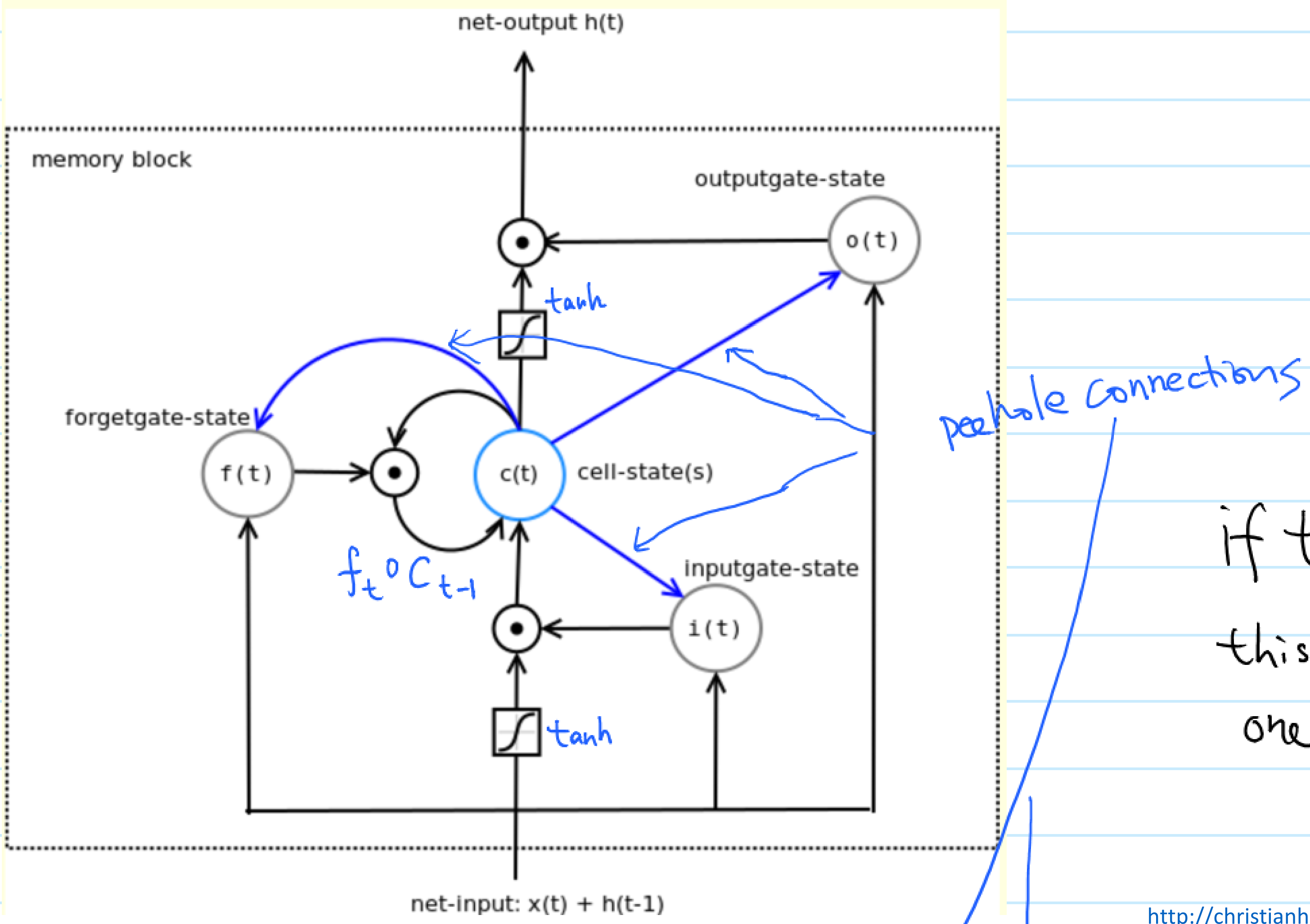


output gate

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

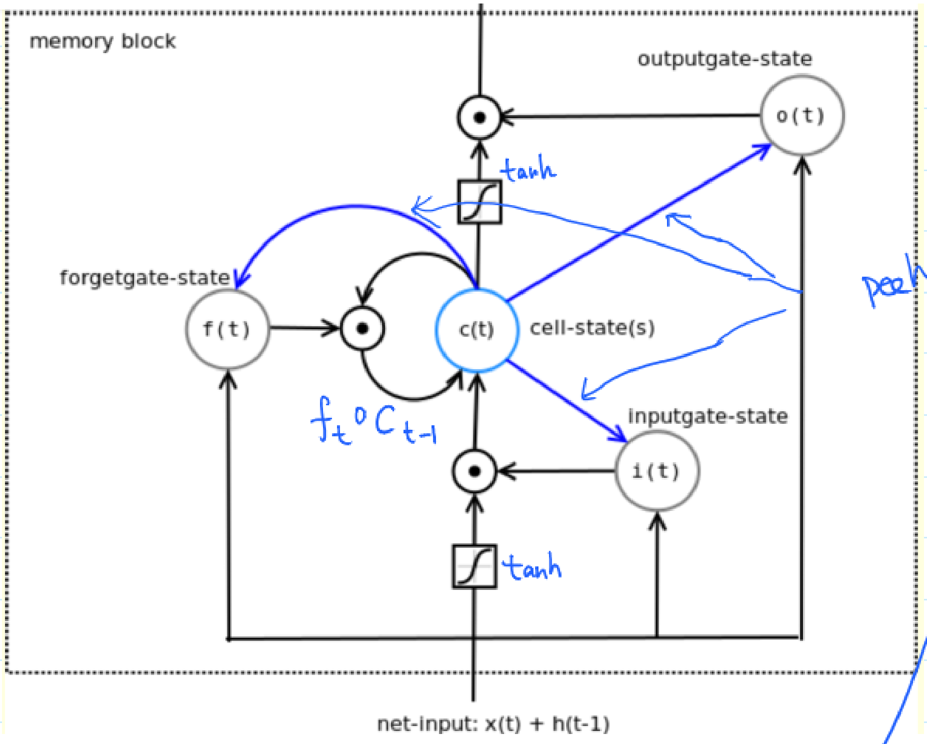
$$h_t = o_t * \tanh\left(C_t\right)$$

In many papers, it looks like this



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

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:

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

Forget gates:

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

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:

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

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).

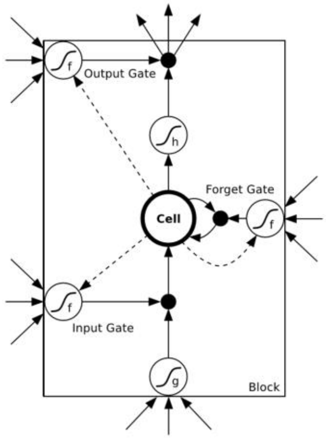
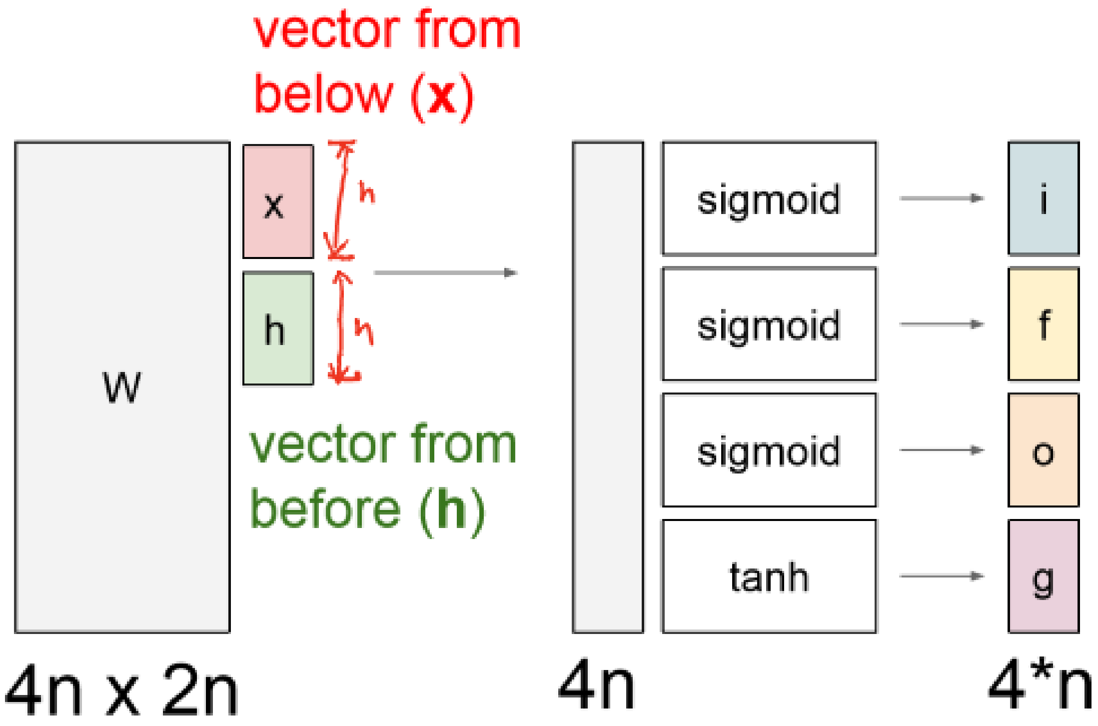


Figure 4.2: **LSTM memory block with one cell.** The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the 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.

More compact

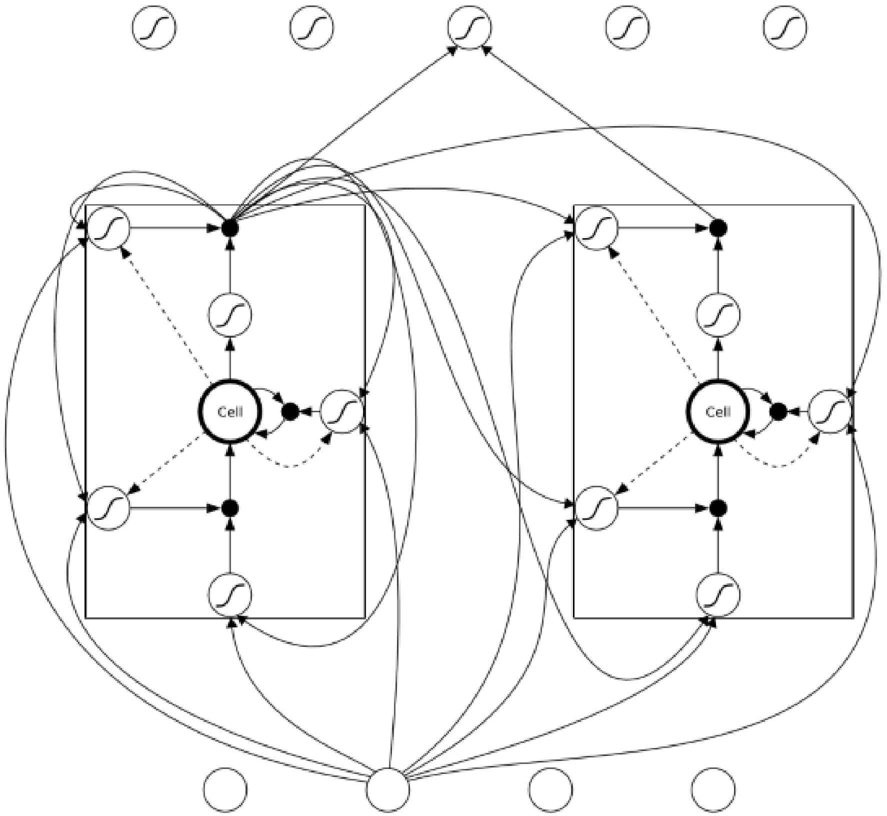


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

An LSTM Network



← not the network unrolled in time

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.

How LSTM deal with gradient vanishing problem

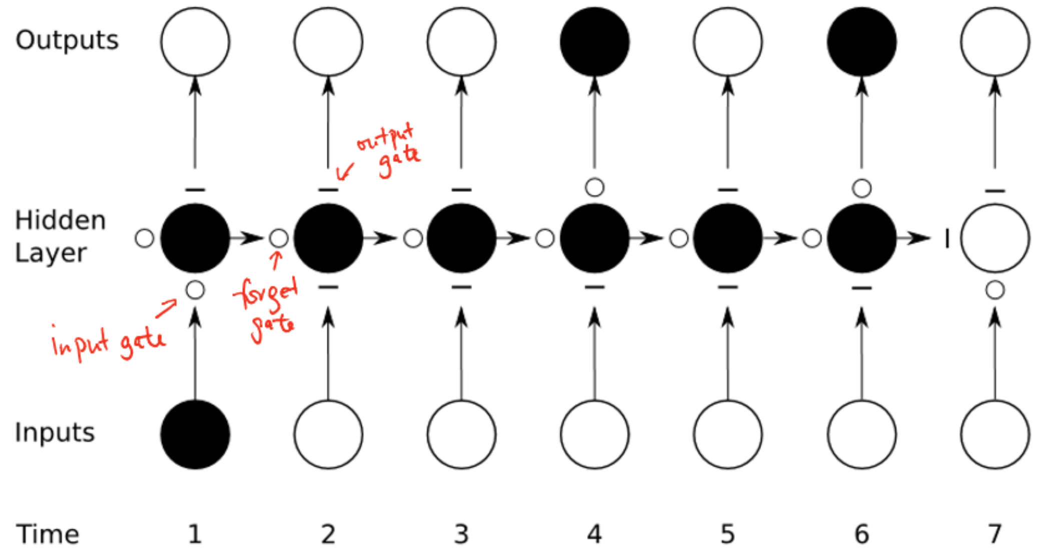
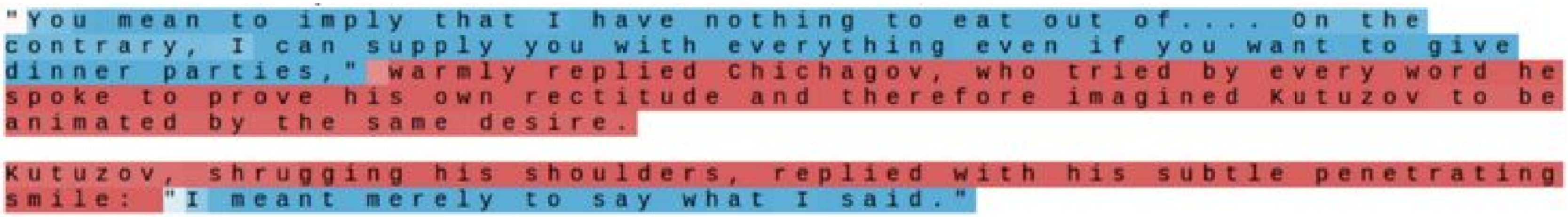


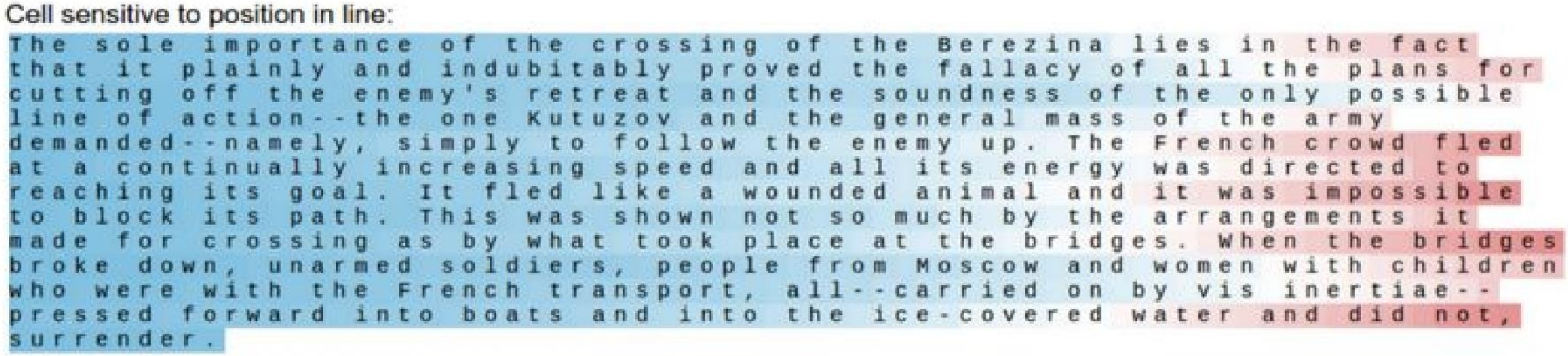
Figure 4.4: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output 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



quote detection cell



line length tracking cell

Visualizing LSTM

```
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;
}
```

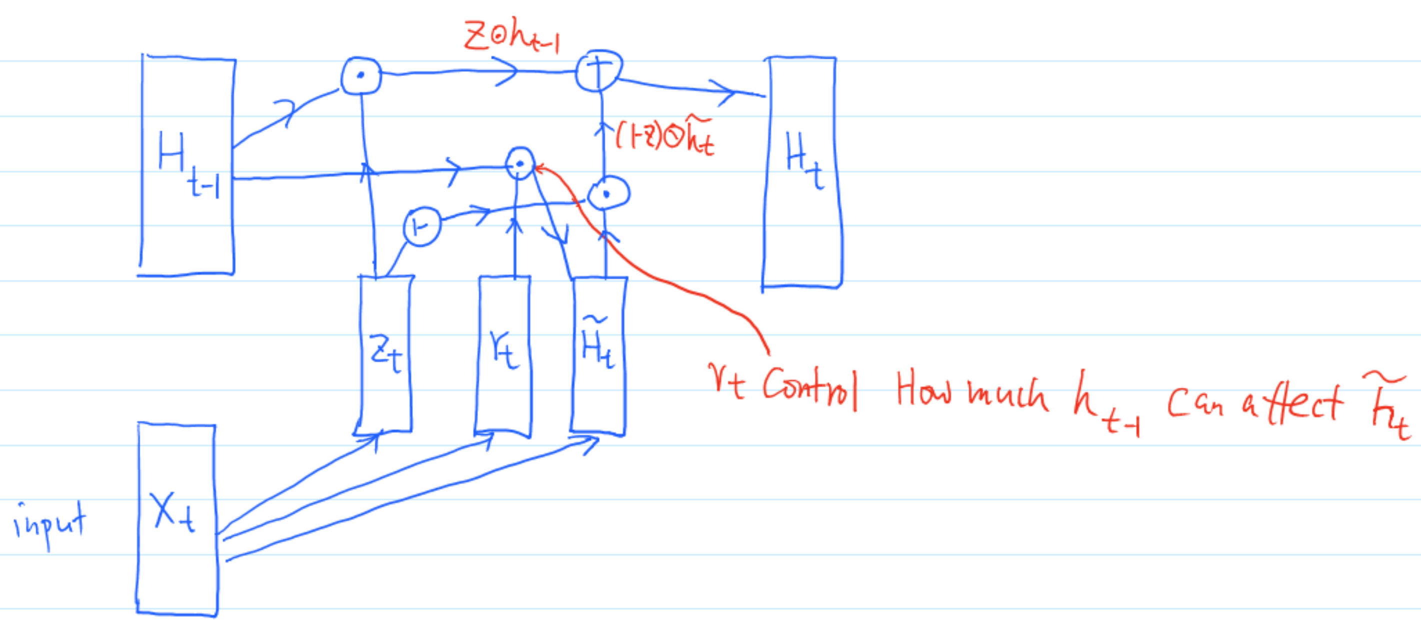
Color: cell state

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

GRU (Gated Recurrent Unit)



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

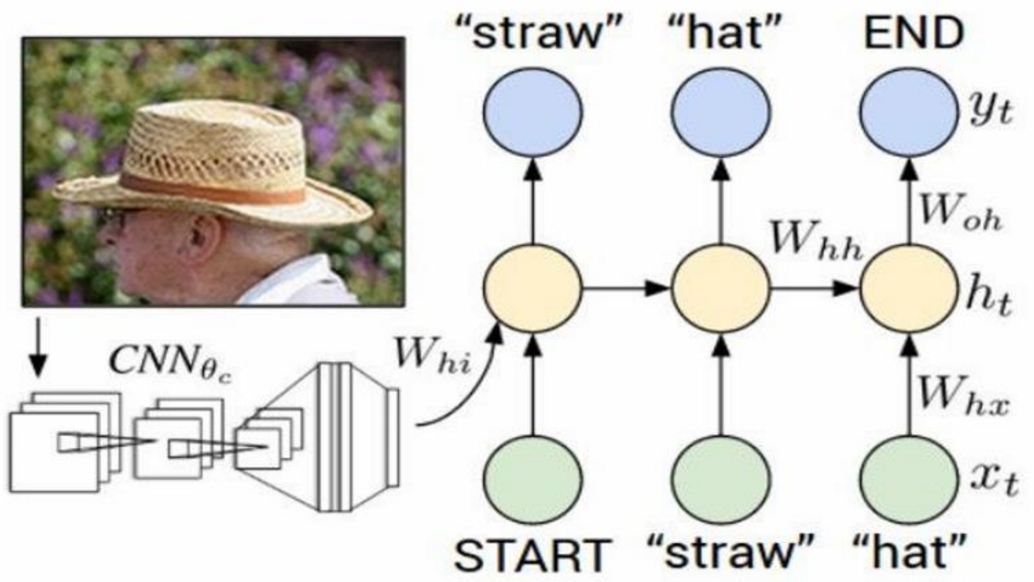
$$z_t = \mathrm{sigm}(W_{xz} x_t + W_{hz} h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh} x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

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

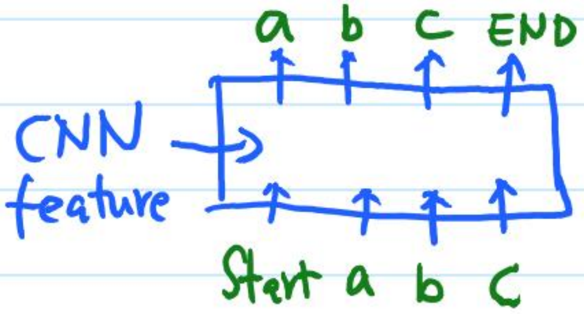
Application – Image Captioning

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:

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

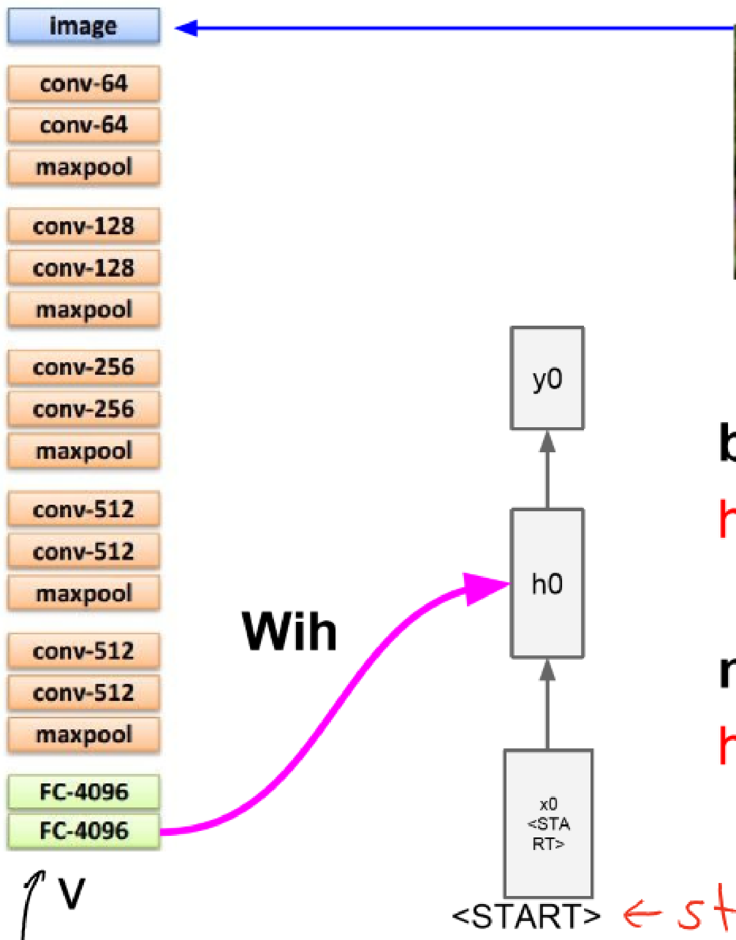
Here, $\mathbb{I}_t$ is an indicator column vector that has a single one at the index of the t-th word in a word vocabulary. The weights W_w specify a word embedding matrix that we initialize with 300-dimensional word2vec [41] weights and keep fixed due to overfitting concerns.

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:



← start token

test image

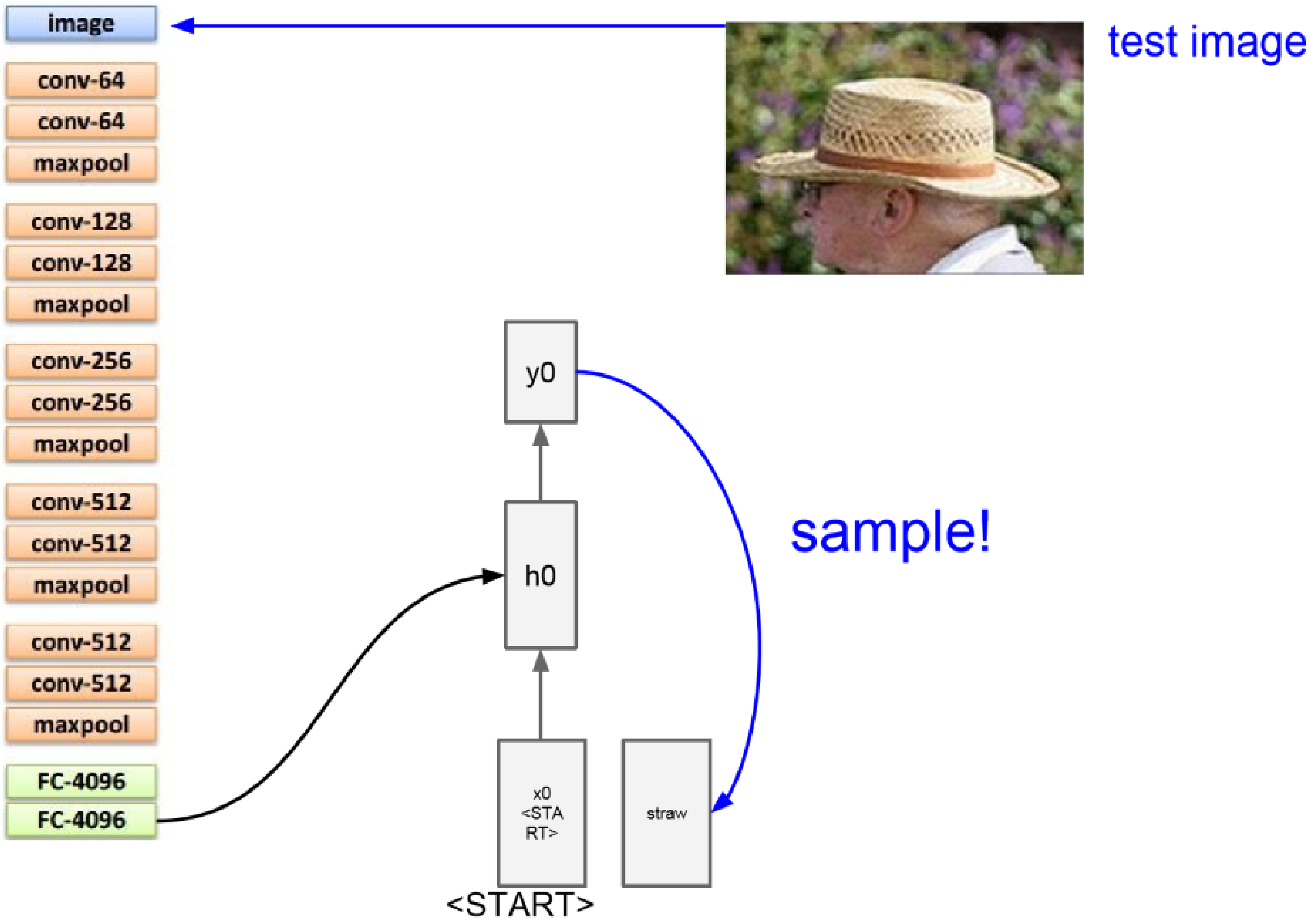
before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

remove the last two layers of CNN



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

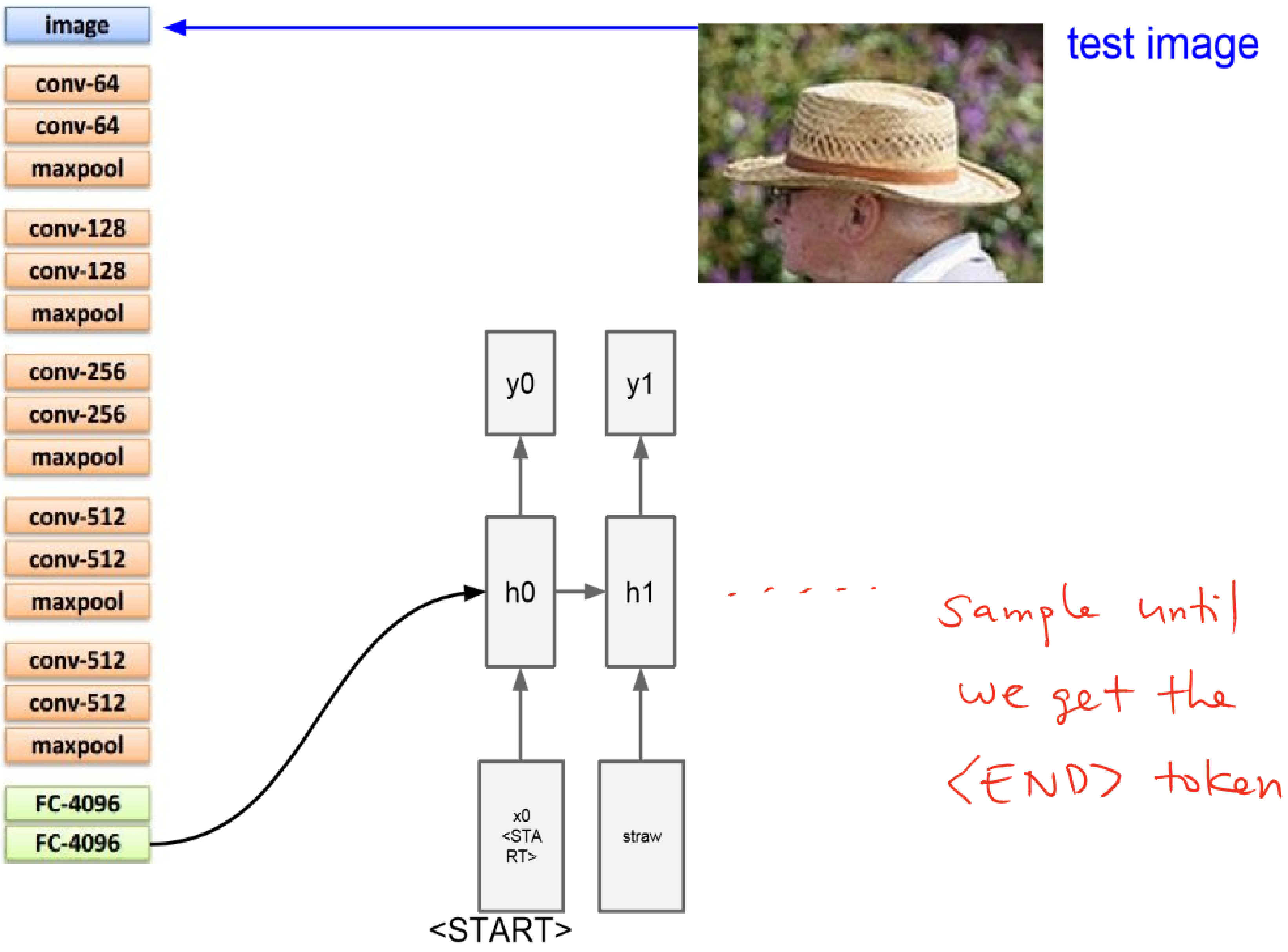


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
x0 <STA RT>
straw
<START>
Sample until we get the <END> token

- 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