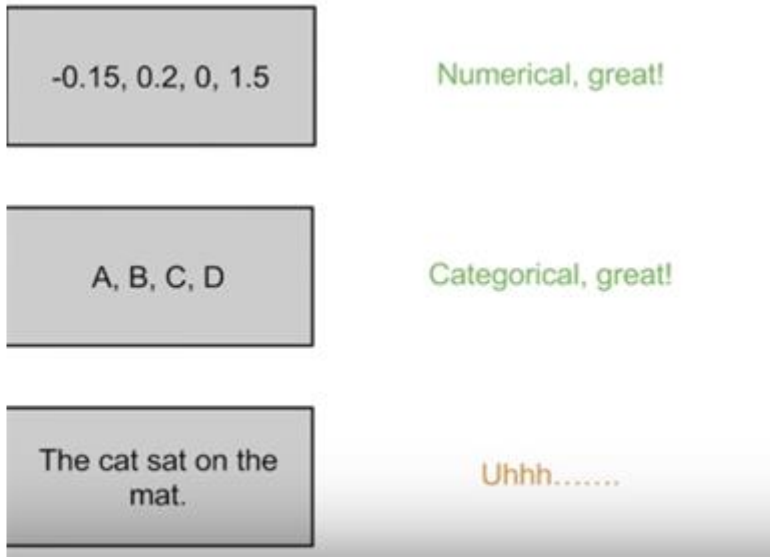
Recurrent Neural Networks (RNN)

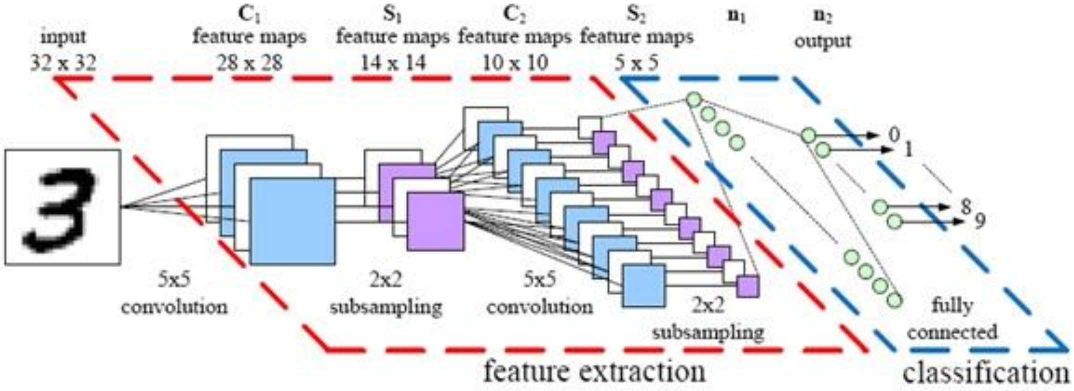
R. Venkatesh Babu, IISc

Data/Output Paradigms



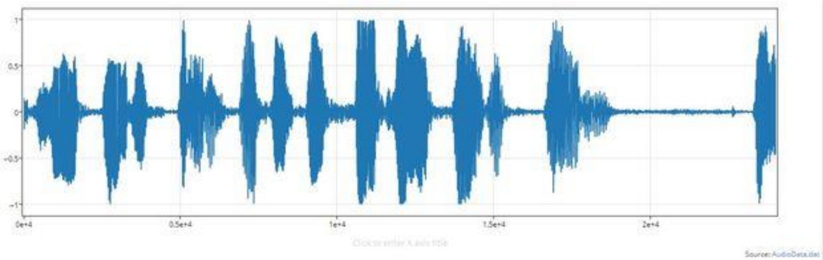
Alec Radford: https://www.youtube.com/watch?v=VINCQghQRuM

CNN Revisited



- Data is fed "in one shot", not in parts (mostly)
  - E.g. Image of "3" above is not processed row-by-row
- Captures "spatial context"
- State-less
- What if data is inherently "sequential" (composed of sequential 'parts') ?
  - Need to capture "sequential context"

Sequential data

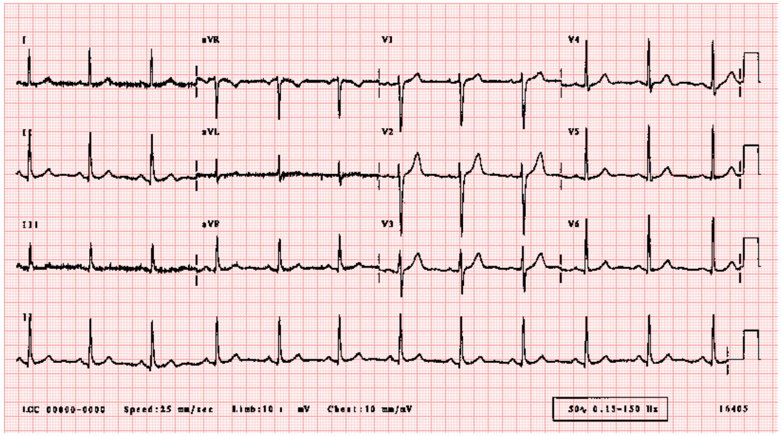


Audio

Video

O Nachiketa, after pondering well the pleasures that are or seem to he delightful, you have renounced them all. You have not taken the road abounding in wealth, where many men sink. *(Kathopanishad, II:3)*

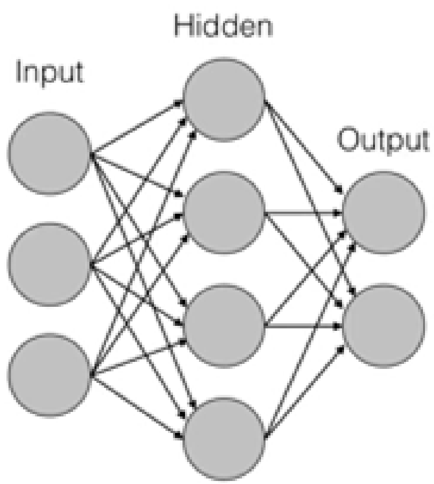
Text



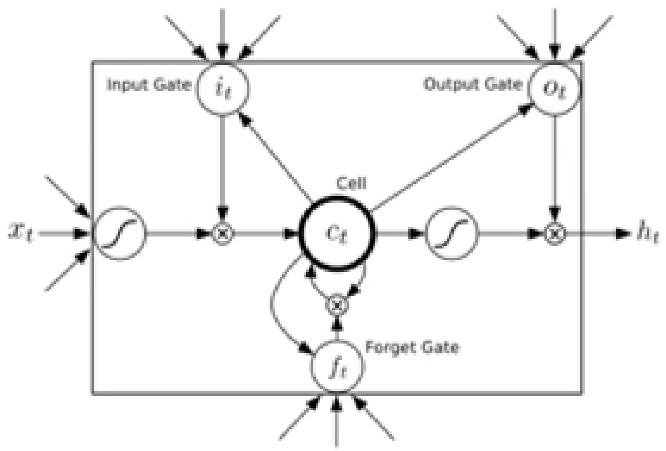
Biological

many more ……

What are RNNs for ?



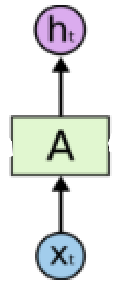
- Independence
- Fixed Length



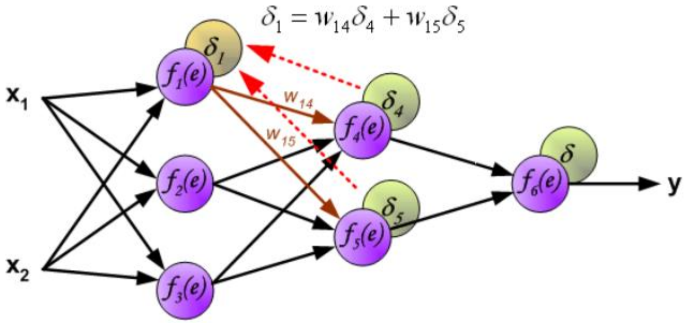
- Temporal dependencies
- Variable sequence length

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Modelling a RNN

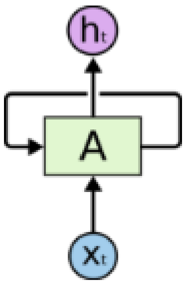


Error derivatives w.r.t weights in kth layer = ***f*(** Error derivatives from (k+1)th layer **)**



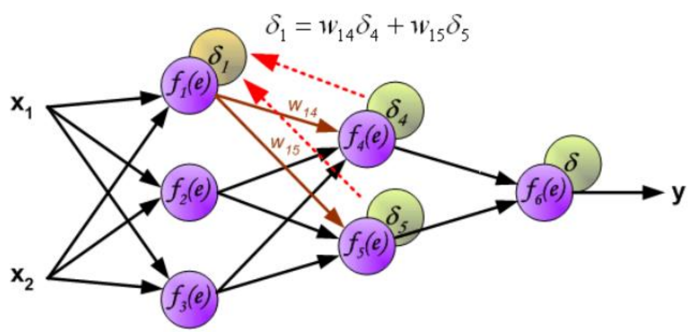
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modelling a RNN



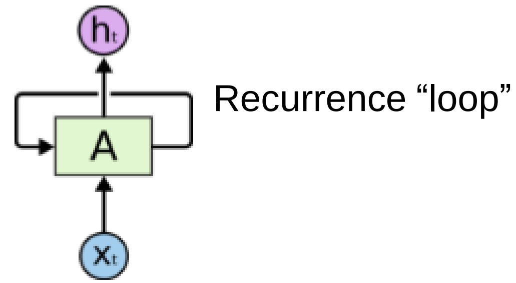
Recurrence “loop”

Error derivatives w.r.t weights in kth layer = ***f*(** Error derivatives from (k+1)th layer **)**

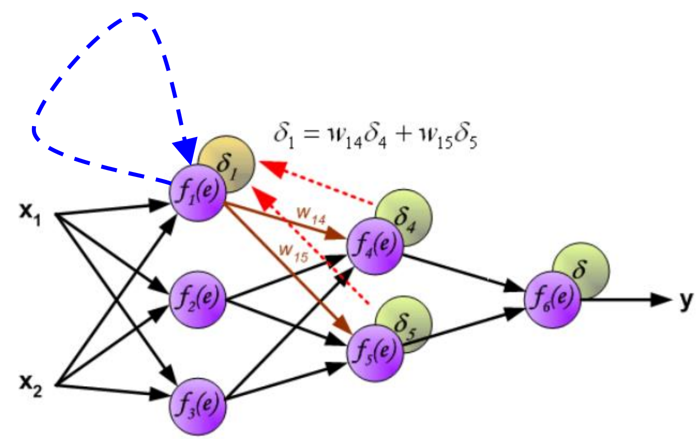


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modelling a RNN

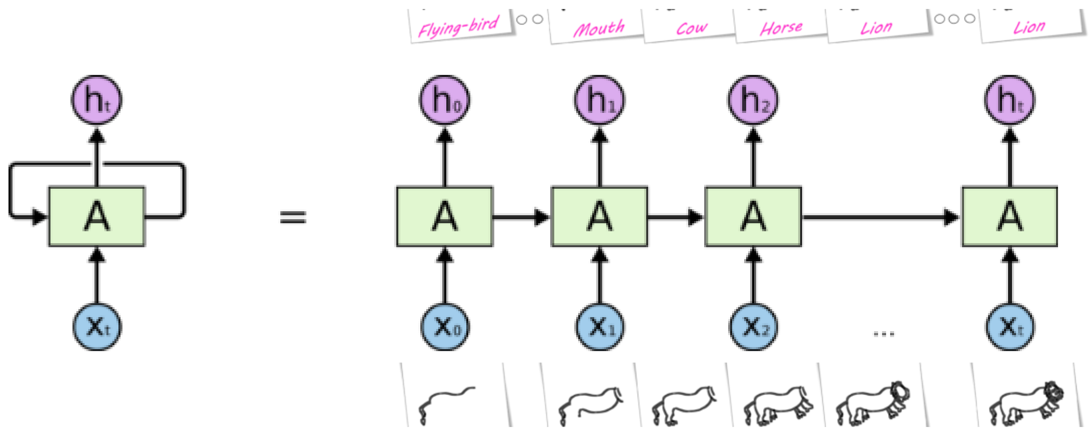


Error derivatives w.r.t weights in kth layer = ***f*(** Error derivatives from (k+1)th layer **)**
⇒ Error derivatives in terms of themselves ??!!!



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modelling a RNN



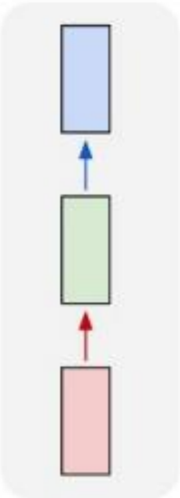
- RNN unrolled → Not “that different” from a CNN (feed-forward connections)
- But crucial differences exist !

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Modelling a RNN

Recurrent Networks offer a lot of flexibility:

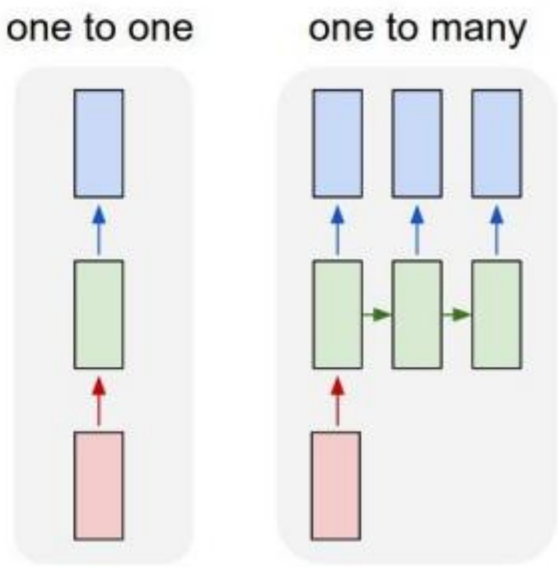
one to one

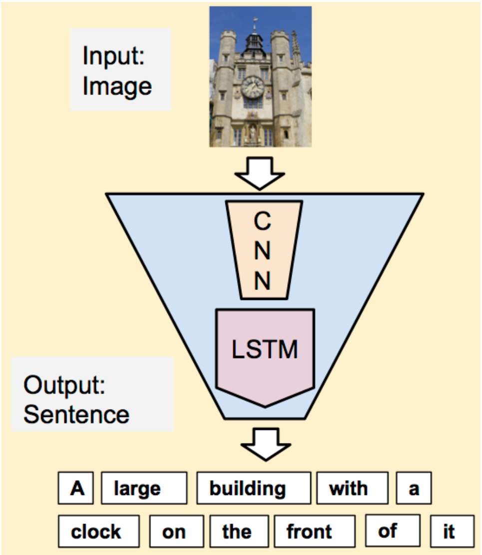


Vanilla Neural Networks

Modelling a RNN

Recurrent Networks offer a lot of flexibility:





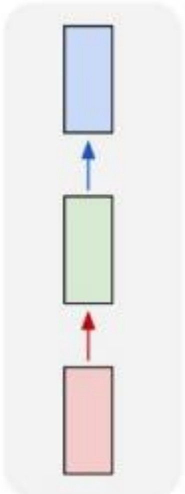
e.g. **Image Captioning**
image -> sequence of words

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

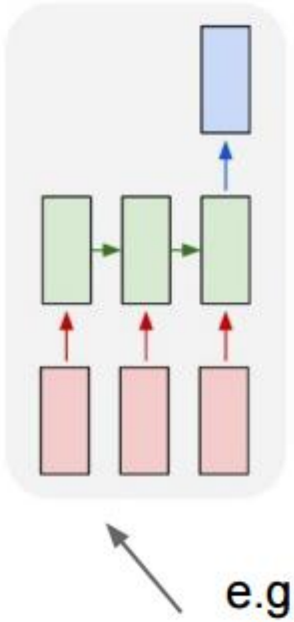
Modelling a RNN

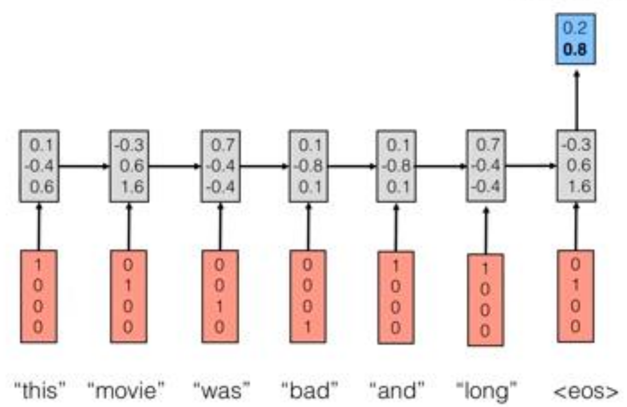
Recurrent Networks offer a lot of flexibility:

one to one



many to one



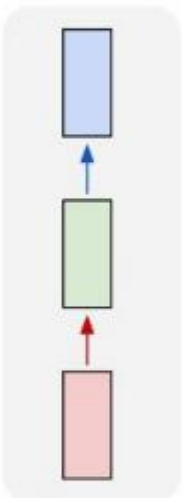


e.g. **Sentiment Classification**
sequence of words -> sentiment

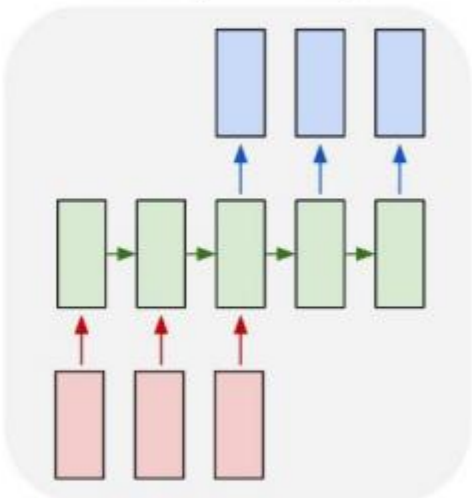
Modelling a RNN

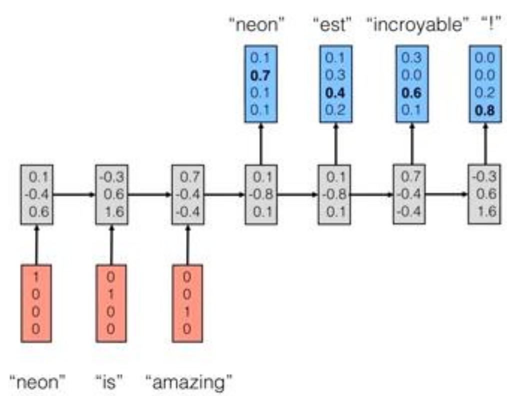
Recurrent Networks offer a lot of flexibility:

one to one



many to many



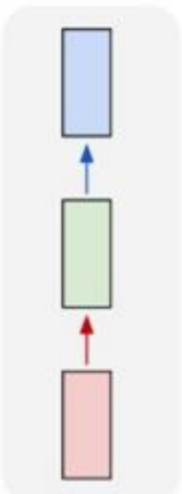


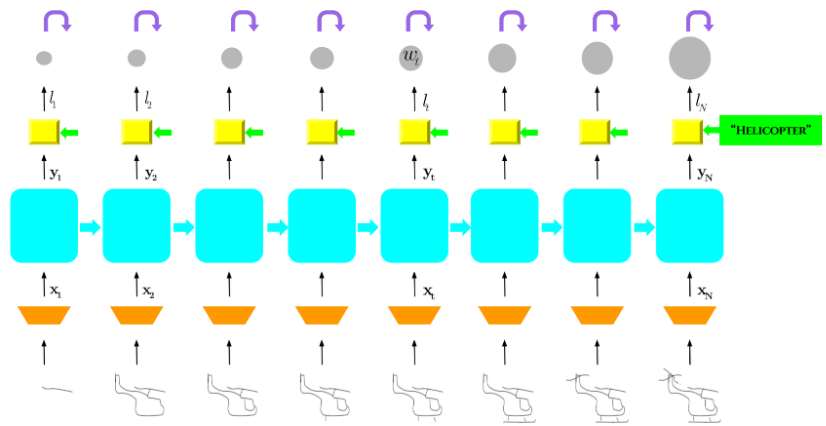
e.g. **Machine Translation**
seq of words -> seq of words

Modelling a RNN

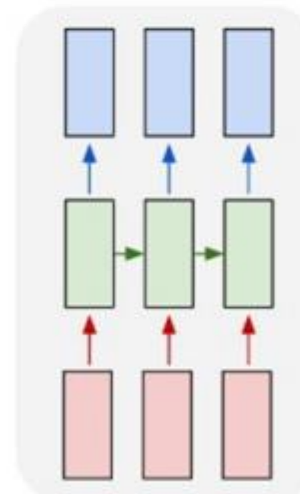
Recurrent Networks offer a lot of flexibility:

one to one



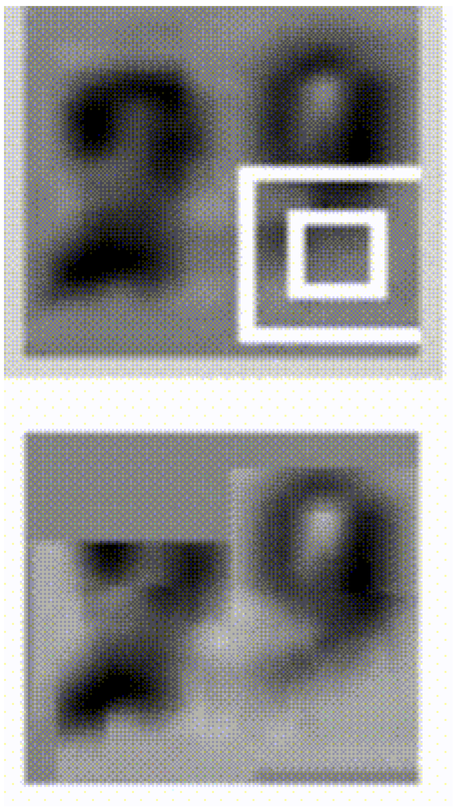


many to many



E.g. On-line freehand sketch recognition
Seq of strokes -> Seq of "guesses"

Paradigm: Sequential "processing"



Reading house numbers

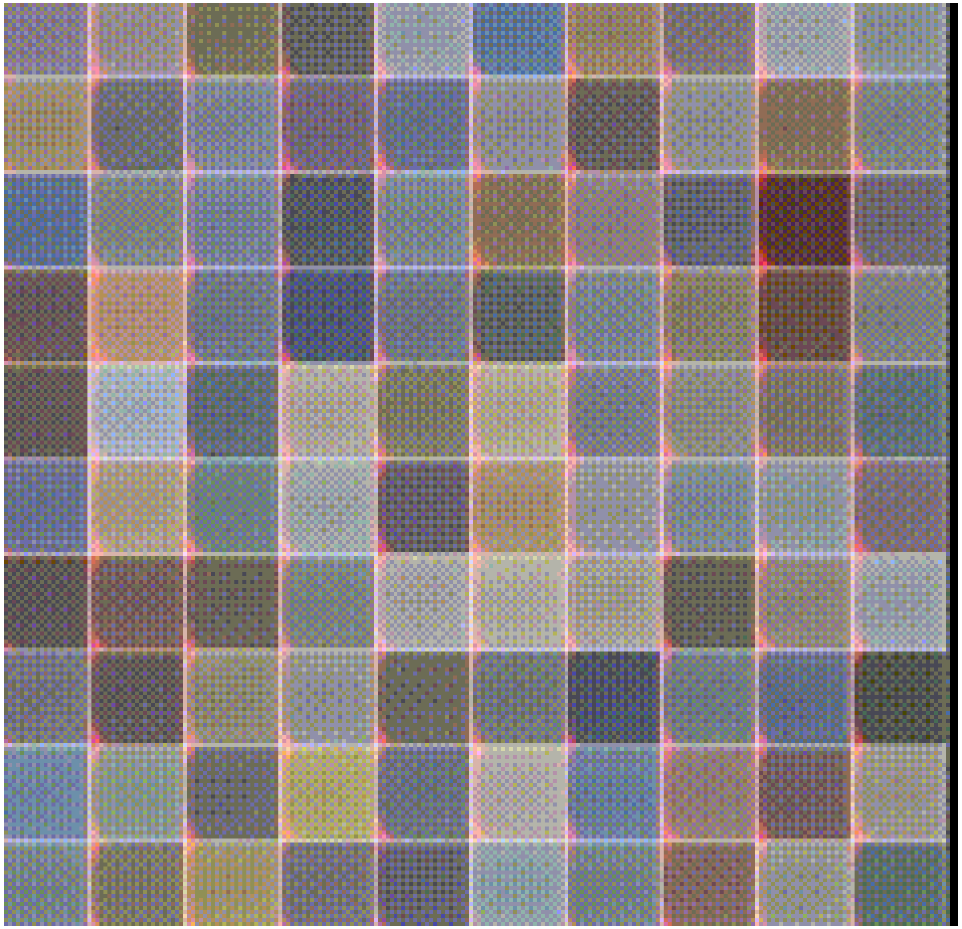
Input/Output may be "fixed", but processing can be sequential !

Multiple Object Recognition with Visual Attention, Ba et al.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Paradigm: Sequential "processing"

DRAW: A Recurrent Neural Network For Image Generation, Gregor et al.



Adding color to house numbers

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

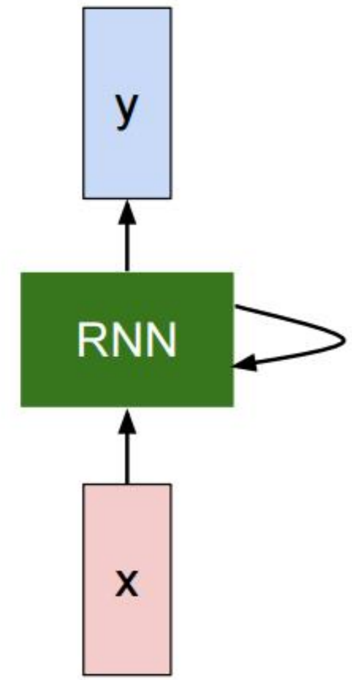
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

some function with parameters W



Task-specific abstraction for portion of sequence seen up to this point ($x_1, x_2 \ldots x_{t-1}$)

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

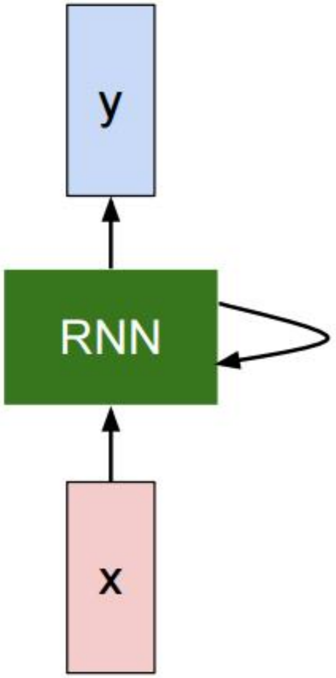
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

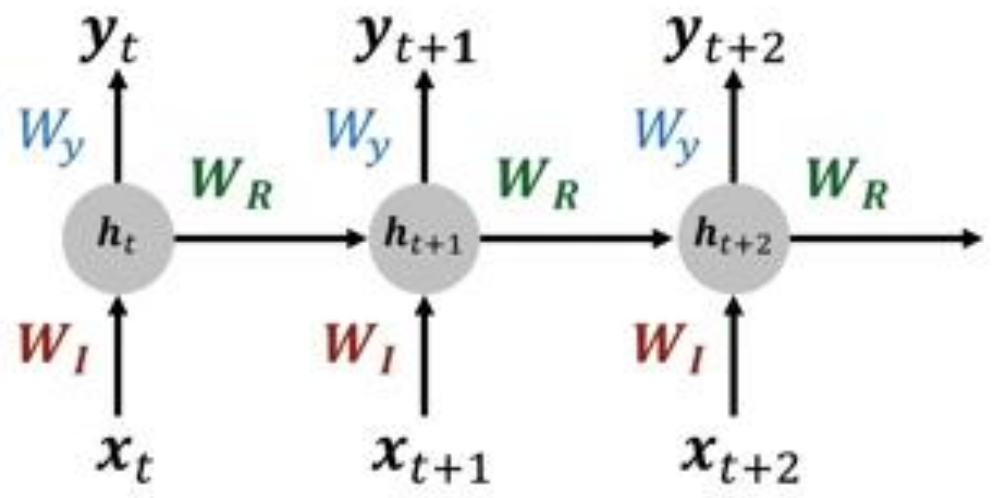
input vector at some time step



Notice: the same function and the same set of parameters are used at every time step.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

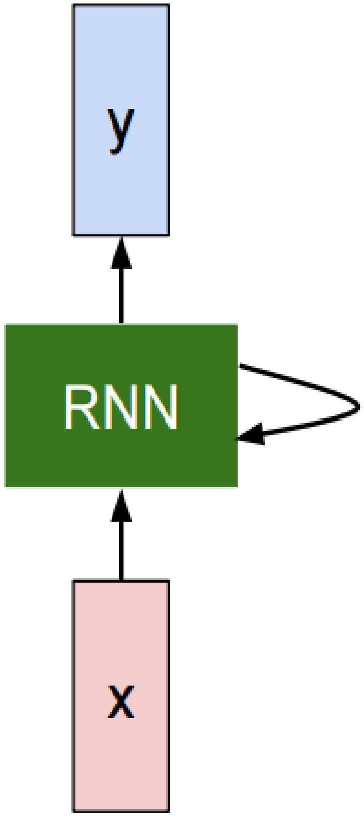


3 sets of parameters - W_I,W_y,W_R (shared for each time-step)

https://www.youtube.com/watch?v=Ukgii7Yd_cU

(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



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

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

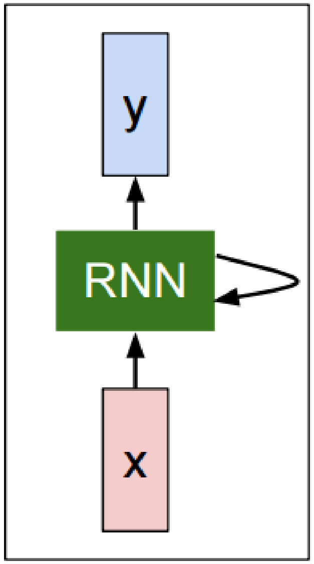
$$y_t = W_{hy}h_t$$

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

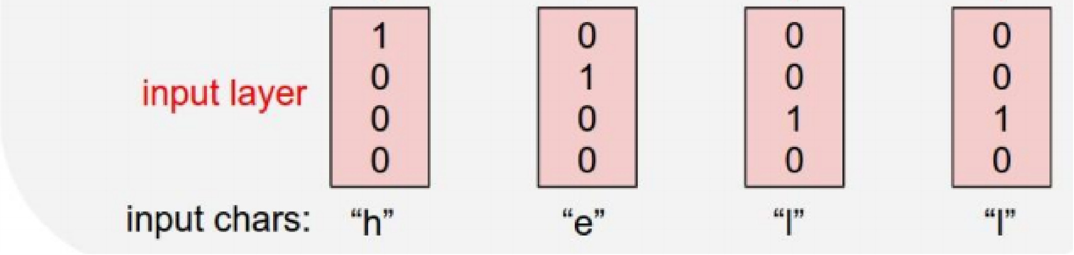


Recurrent Neural Network

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

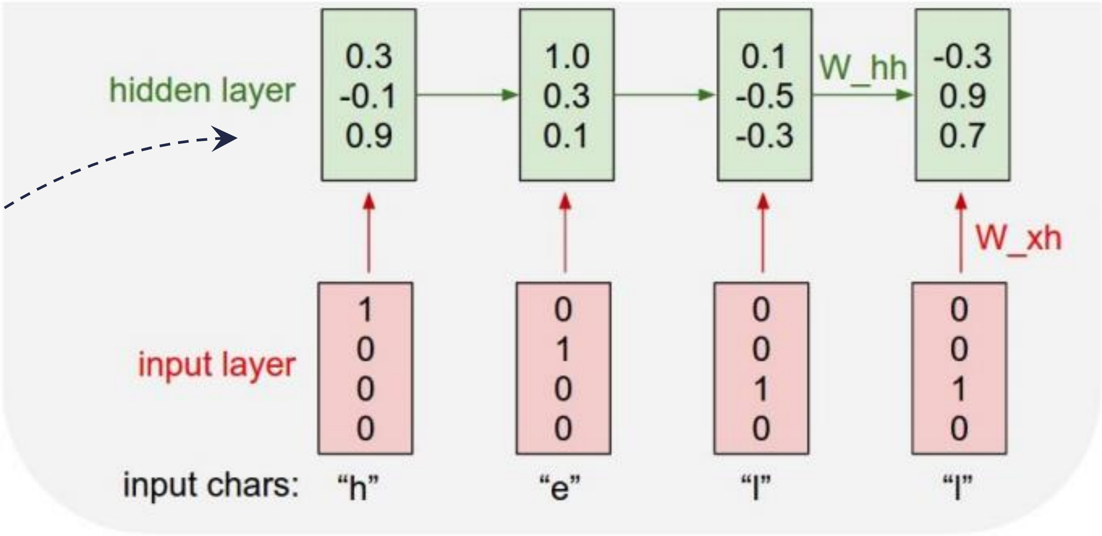
Recurrent Neural Network

Character-level language model example

$$\boxed{h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)}$$

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**



h_t → abstraction of current input upto step t

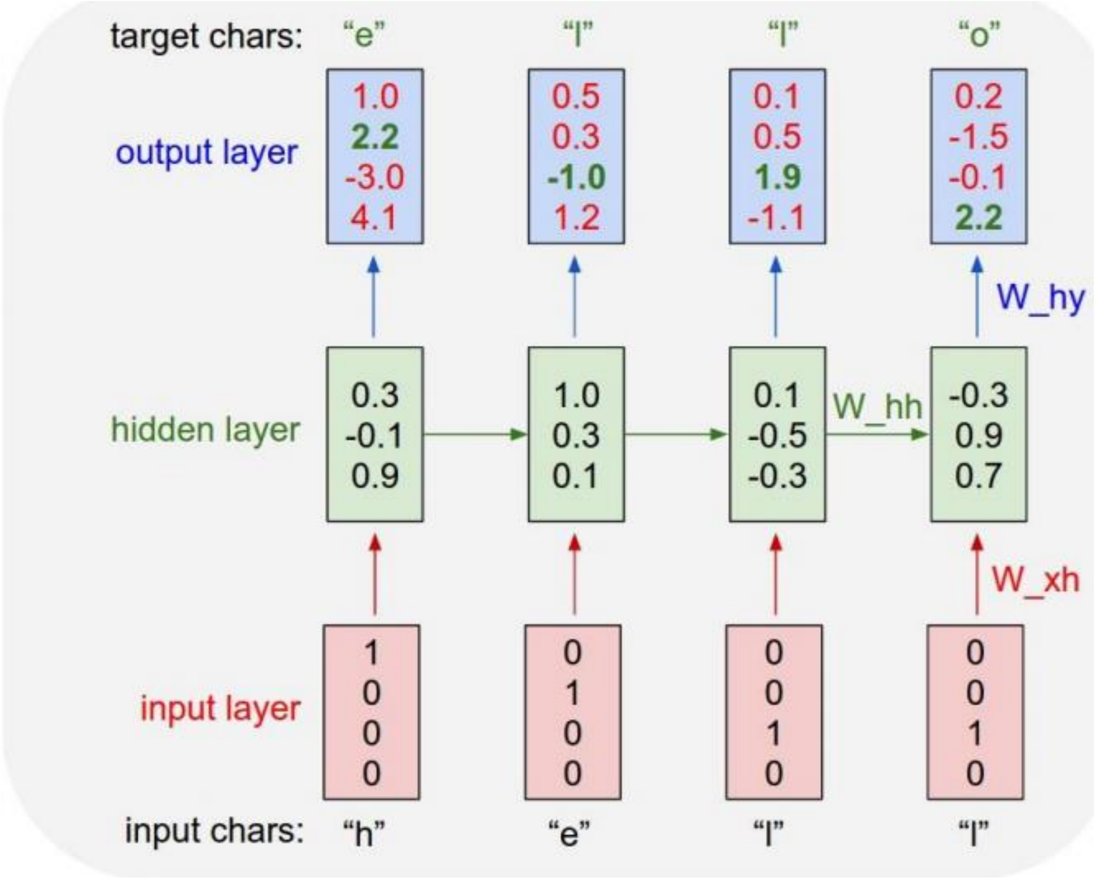
Recurrent Neural Network

$$y_t = W_{hy} h_t$$

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**



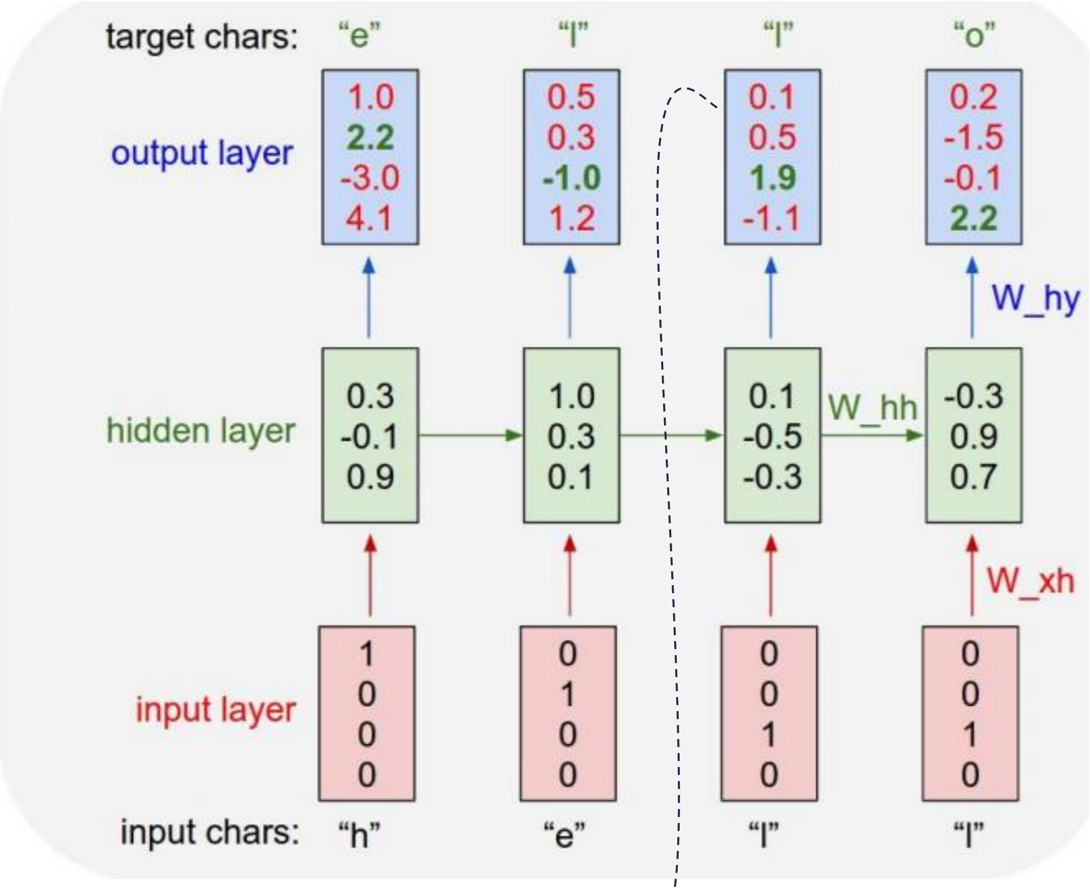
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**



$$p(\text{next letter} = \text{"l"} \mid \text{previous letters} = \{\text{"h"}, \text{"e"}, \text{"l"}\})$$

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
    Admit impediments. Love is not love
Which alters when it alteration finds,
    Or bends with the remover to remove:
O no! it is an ever-fixed mark
    That looks on tempests and is never shaken;
It is the star to every wandering bark,
    Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
    Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
    But bears it out even to the edge of doom.
If this be error and upon me proved,
    I never writ, nor no man ever loved.

Recurrent Neural Network

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.
```

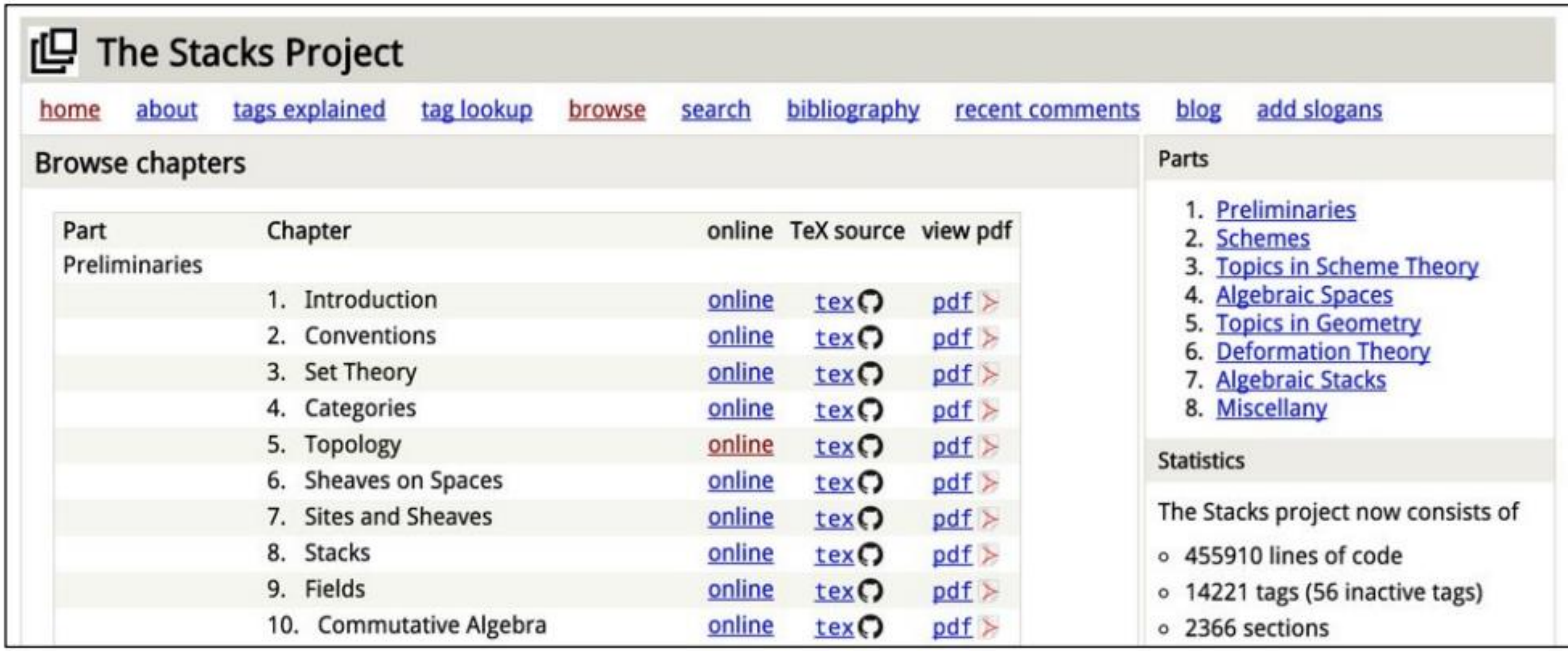
train more

```
"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.
```

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

open source textbook on algebraic geometry



The Stacks Project

home about tags explained tag lookup browse search bibliography recent comments blog add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Proof. Omitted. ☐

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. ☐

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. ☐

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

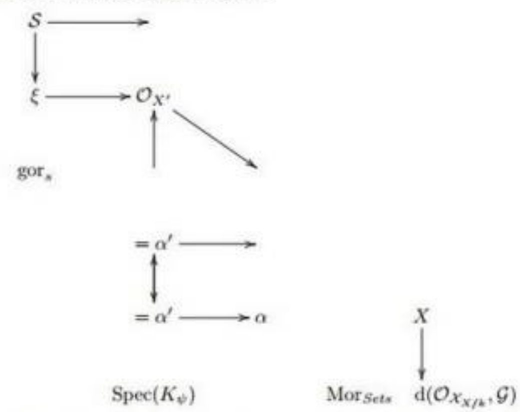
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. ☐

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

☐

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. ☐

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

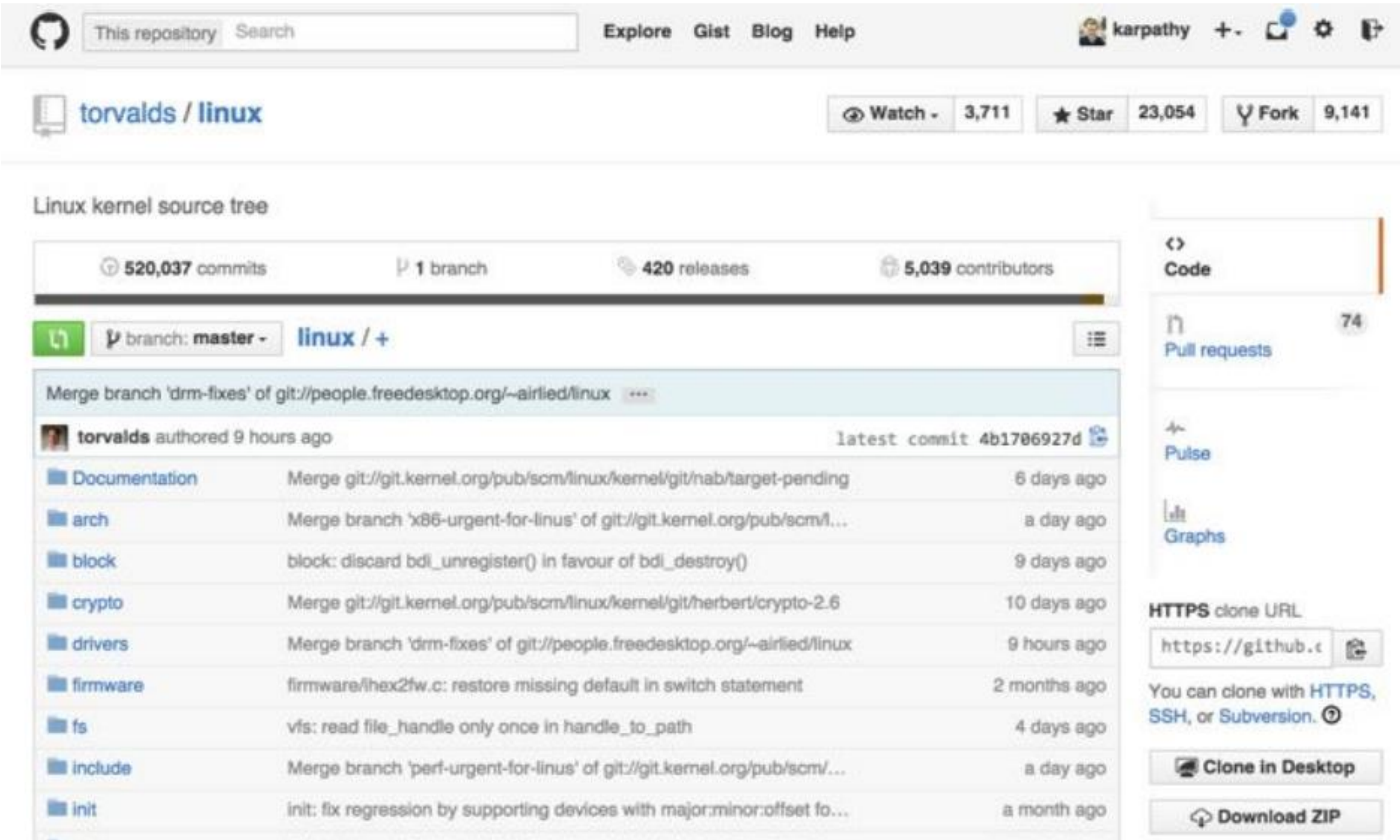
$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. ☐

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

```
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 ");
}
```

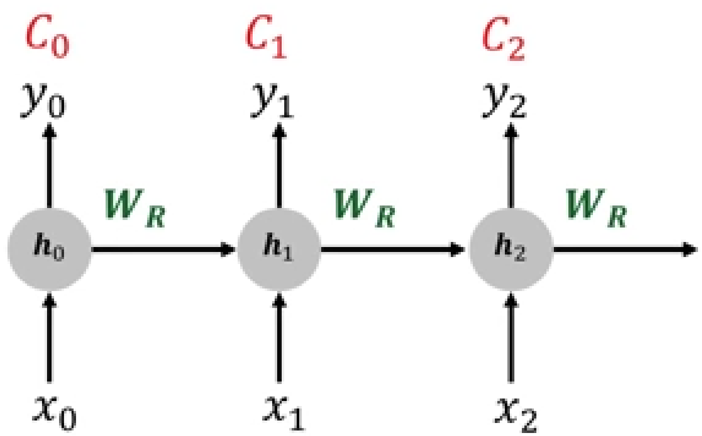
Generated C code

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Training RNNs - Backpropagation through time

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



$$\frac{\partial C}{\partial W_R} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial W_R}$$

$$\frac{\partial C_t}{\partial W_R} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_R}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_R^T \mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))$$

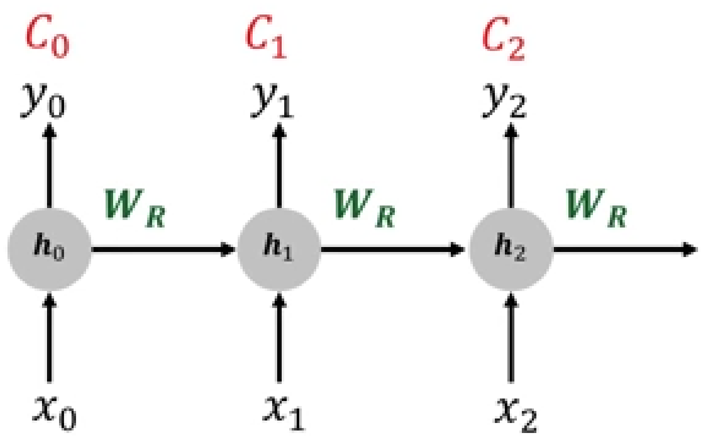
- (Potentially) multiple, intermediate costs C_0, C_1, C_2
- Shared weights $\mathbf{W_R}$

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs - BPTT, Vanishing gradients

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



$$\frac{\partial C}{\partial W_R} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial W_R}$$

$$\frac{\partial C_t}{\partial W_R} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_R}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_R^T \mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leqslant \|W_R^T\| \, \|\mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))\| \leqslant \gamma_{W_R}\gamma_{g_h}$$

Vanishing gradients !

(Pascanu, et al., On the difficulty of training Recurrent Neural Networks.)

Training RNNs

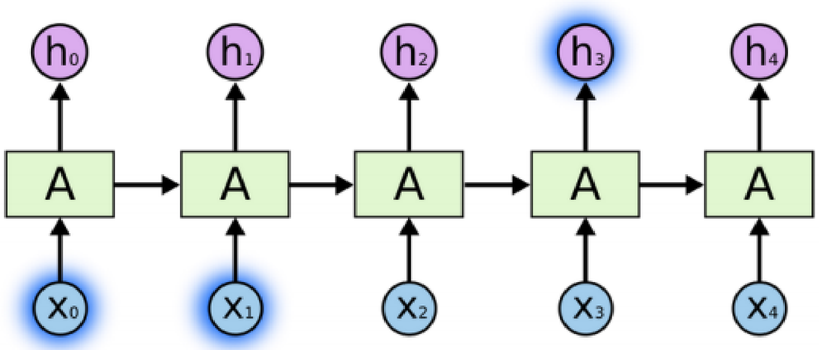
Choice of a(.) is crucial too !

- **$|W_R| << 1 \rightarrow$** Vanishing gradients
- **$|W_R| >> 1 \rightarrow$** Vanishing gradients
- **$|W_R|$ 'close to 1'** is ideal

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Fundamental issues with RNNs

- RNNs connect previous information to the present task
  - Previous video frames may help understand present frame
- Sometimes we need only look at recent previous information to predict
  - To predict the last word of "The clouds are in the *sky*" we don't need any further context. It is obvious that the word is "sky"

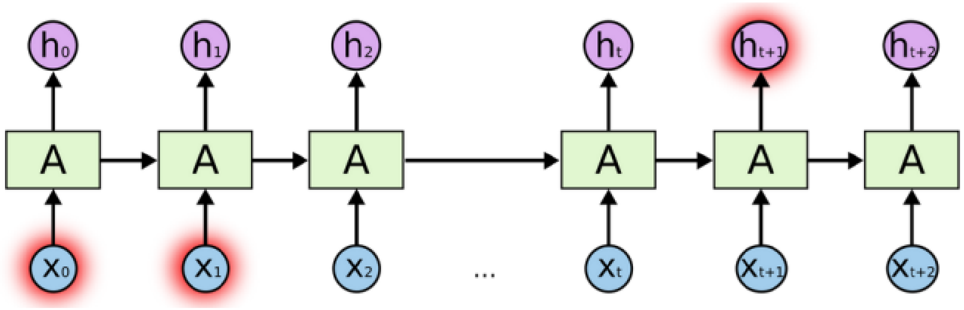


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Fundamental issues with RNNs

Problem of Long-term dependency

- There are cases where we need more context
- To predict the last word in the sentence "I grew up in France….I speak fluent *French*"
- Using only recent information suggests that the last word is the name of a language. But more distant past indicates that it is French
- It is possible that the gap between the relevant information and where it is needed is very large



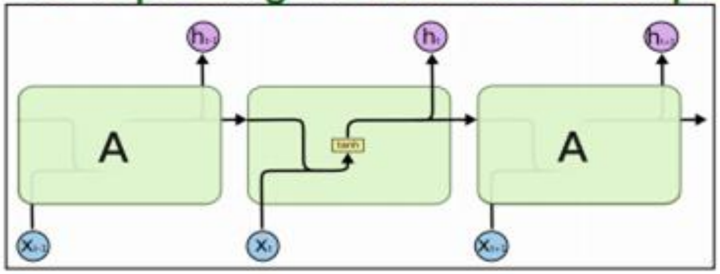
http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

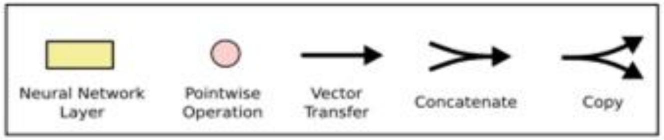

I'LL EMAIL THIS TO MYSELF SO I CAN REMEMBER IT
HEY LOOK, A NEW EMAIL!


Not only is my short-term memory horrible, but so is my short-term memory.
Fb.com/MinionQuote
DespicableMeMinions.org

Long Short Term Memory (LSTM)

- Explicitly designed to avoid the long-term dependency problem
- RNNs have the form of a repeating chain structure
  - The repeating module has a simple structure such as tanh

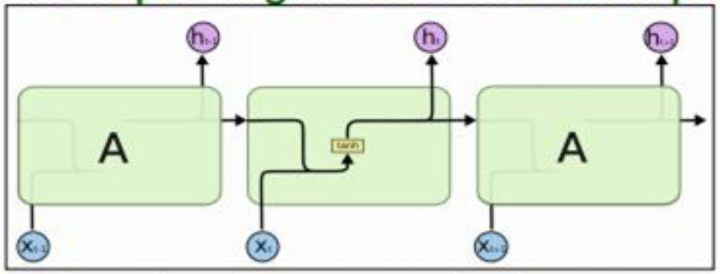


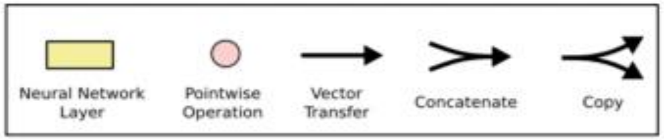


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

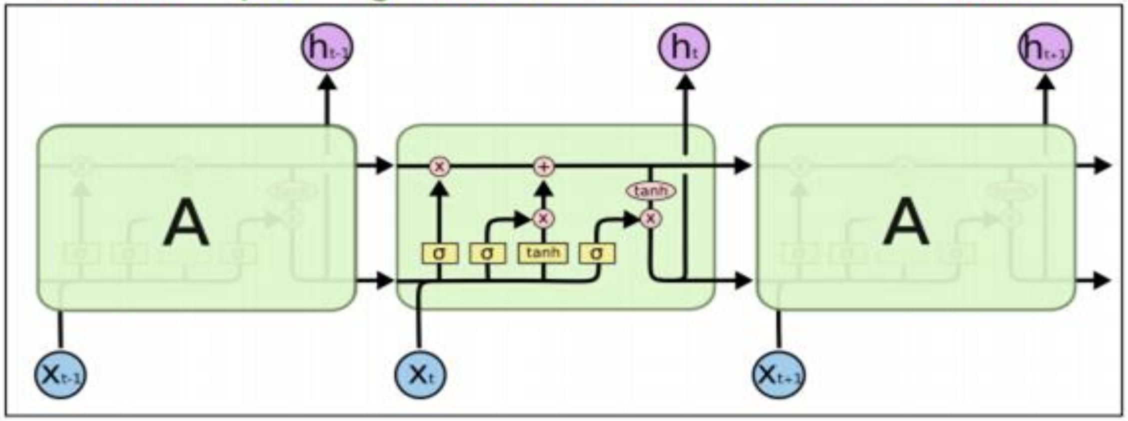
Long Short Term Memory (LSTM)

- Explicitly designed to avoid the long-term dependency problem
- RNNs have the form of a repeating chain structure
  - The repeating module has a simple structure such as tanh





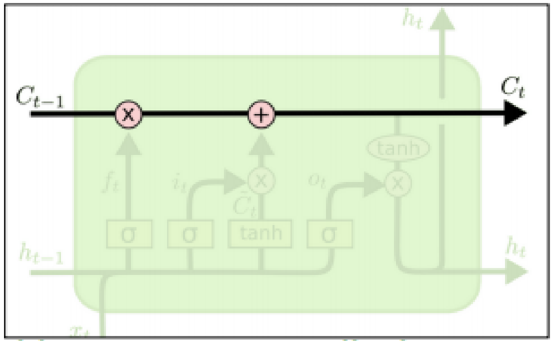
- LSTMs also have a chain structure
  - but the repeating module has a different structure



LSTM

Core idea behind LSTM

- The key to LSTM is the cell state, C_t , the horizontal line running through the top of the diagram
- Like a conveyor belt
  - Runs through entire chain with minor interactions
  - LSTM does have the ability to remove/add information to cell state regulated by structures called gates

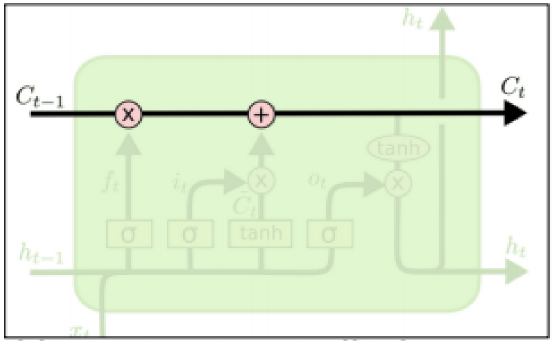


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

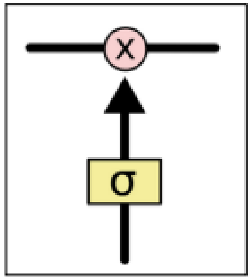
LSTM

Core idea behind LSTM

- The key to LSTM is the cell state, C_t, the horizontal line running through the top of the diagram



- Like a conveyor belt
  - Runs through entire chain with minor interactions
  - LSTM does have the ability to remove/add information to cell state regulated by structures called gates
- Gates are an optional way to let information through

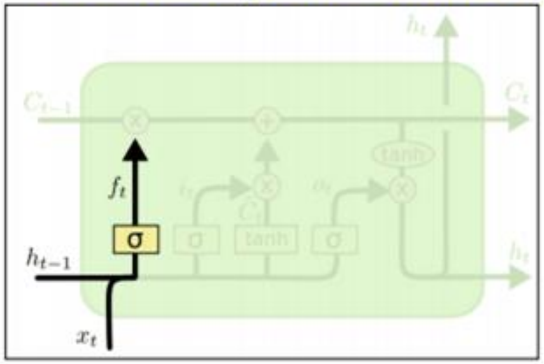


- Consist of a sigmoid and a multiplication operation
- Sigmoid outputs a value between 0 and 1
  - 0 means let nothing through
  - 1 means let eveything through
- LSTM has three of these gates, to protect and control cell state

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Example of language model: predict next word based on previous ones
  - Cell state may include the gender of the present subject
- First step: information to throw away from cell state



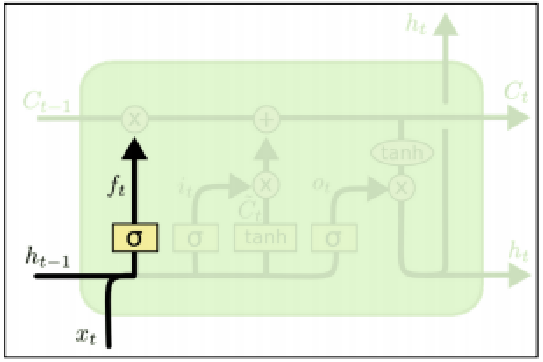
Called *forget gate layer*

It looks at $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ and outputs a number between 0 and 1 for each member of $\boldsymbol{C}_{t-1}$ for whether to forget

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

LSTM - Components

- Example of language model: predict next word based on previous ones
  - Cell state may include the gender of the present subject
- First step: information to throw away from cell state



Called *forget gate layer*

It looks at $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ and outputs a number between 0 and 1 for each member of $\boldsymbol{C}_{t-1}$ for whether to forget

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

In language model

consider trying to predict the next word based on all previous ones

The cell state may include the gender of the present subject

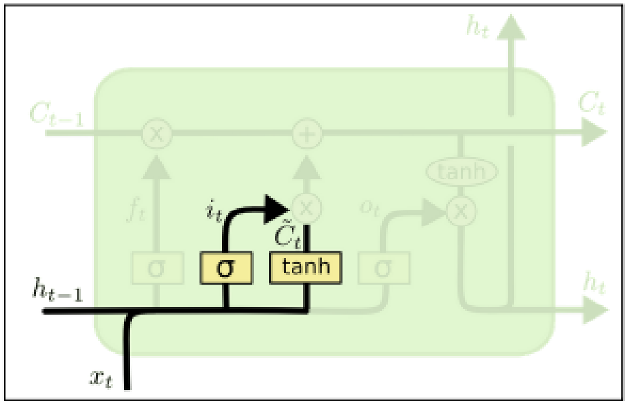
so that the proper pronouns can be used

When we see a new subject we want to forget old subject

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Next step is to decide as to what new information we're going to store in the cell state



This has two parts:
first a sigmoid layer called *Input gate layer*: decides which values we will update
Next a $\tanh$ layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.
In the third step we will combine these two to create an update to the state

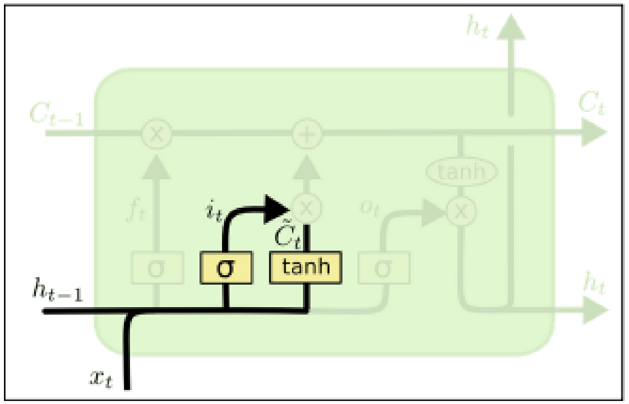
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

LSTM - Components

- Next step is to decide as to what new information we're going to store in the cell state



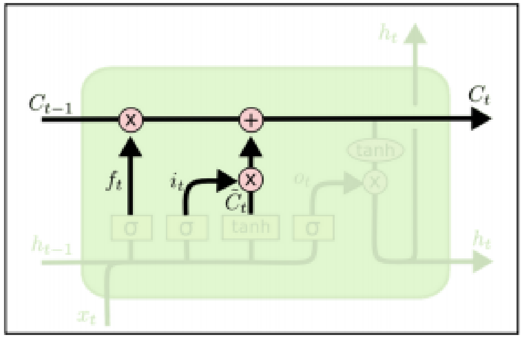
This has two parts:
first a sigmoid layer called *Input gate layer*: decides which values we will update
Next a $\tanh$ layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state.
In the third step we will combine these two to create an update to the state

In the Language model, we'd want to add the gender of the new subject to the cell state, to replace the old One we are forgetting

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM - Components

- It's now time to update old cell state C_{t-1} into new cell state C_t
  - The previous step decided what we need to do
  - We just need to do it



We multiply the old state by f_t, forgetting the things we decided to forget earlier.
Then we add $i_t * \tilde{C}_t$
This is the new candidate values, scaled by how much we decided to update each state value

In the Language model,
this is where we'd actually drop the information about the old subject's gender and add the new information,
as we decided in previous steps

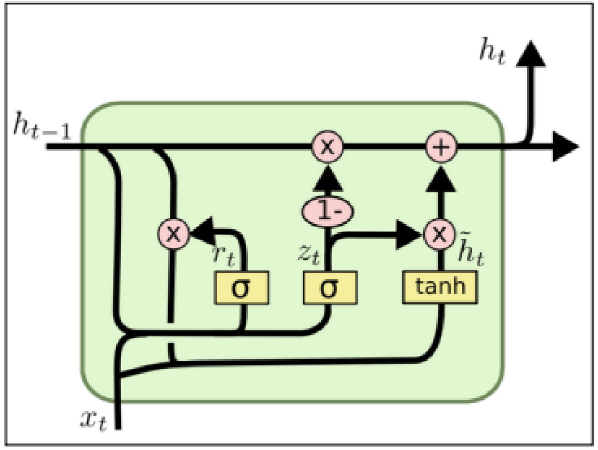
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

GRU: a LSTM variant

Gated Recurrent Unit (GRU)

- A dramatic variant of LSTM
  - It combines the forget and input gates into a single update gate
  - It also merges the cell state and hidden state, and makes some other changes
  - The resulting model is simpler than LSTM models
  - Has become increasingly popular



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

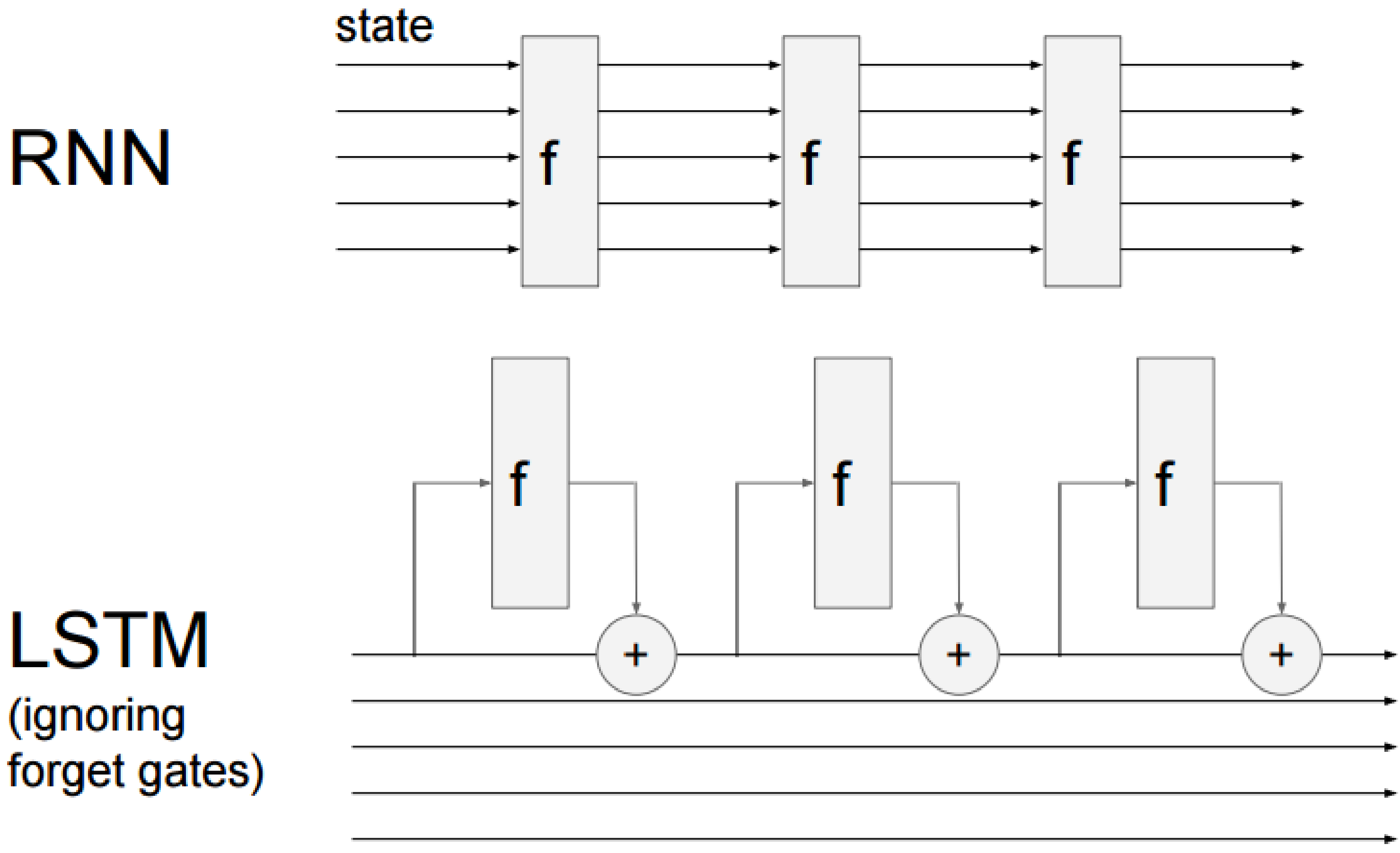
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

13

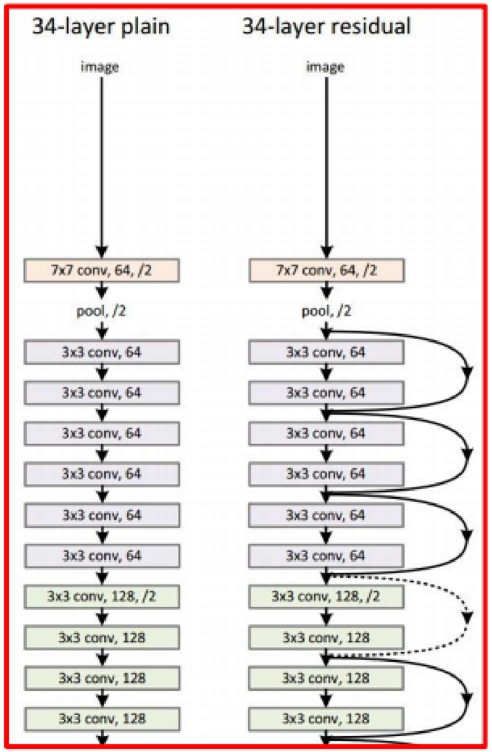
http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Long Short Term Memory (LSTM)



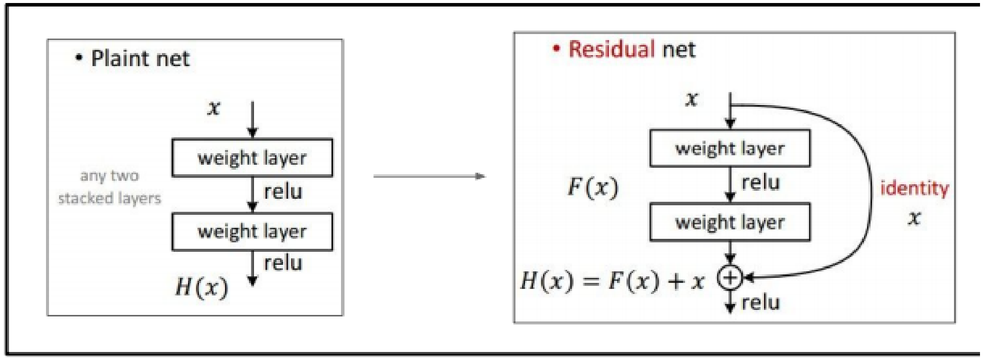
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Long Short Term Memory (LSTM)



Recall: “PlainNets” vs. ResNets

ResNet is to PlainNet what LSTM is to RNN, kind of.



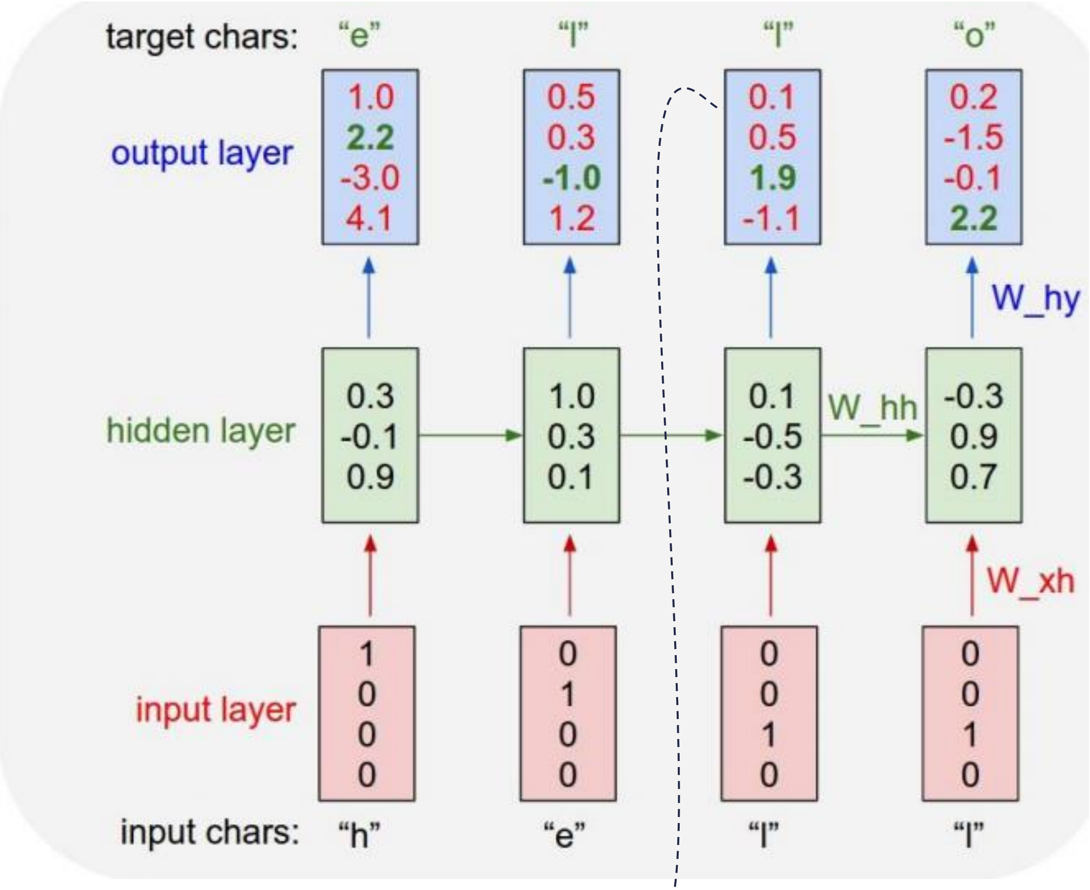
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



$$p(\text{next letter}=\text{“l”} \mid \text{previous letters} = \{\text{“h”}, \text{“e”}, \text{“l”}\})$$

Recurrent Neural Network

Searching for interpretable cells

"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.

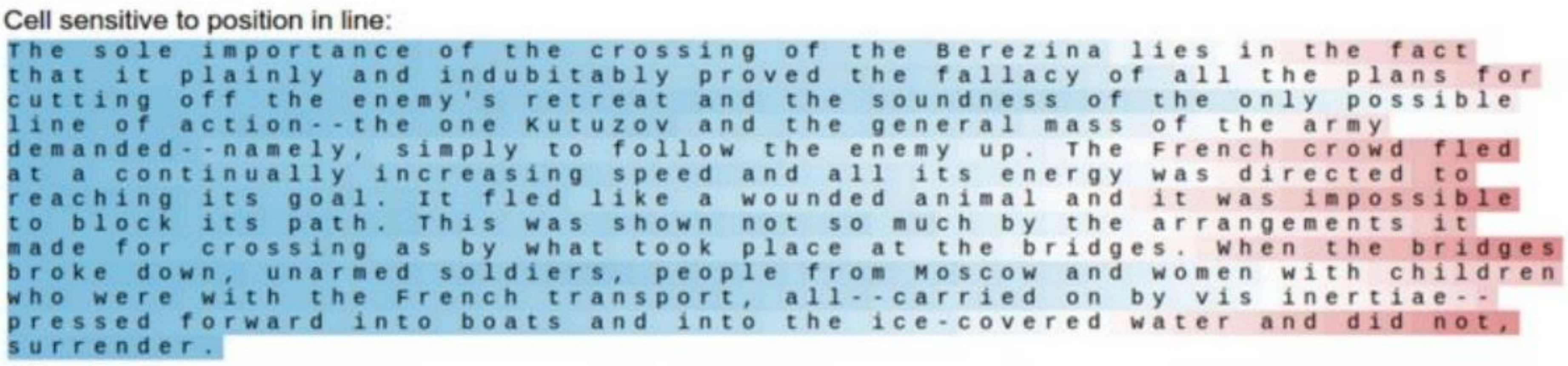
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Recurrent Neural Network

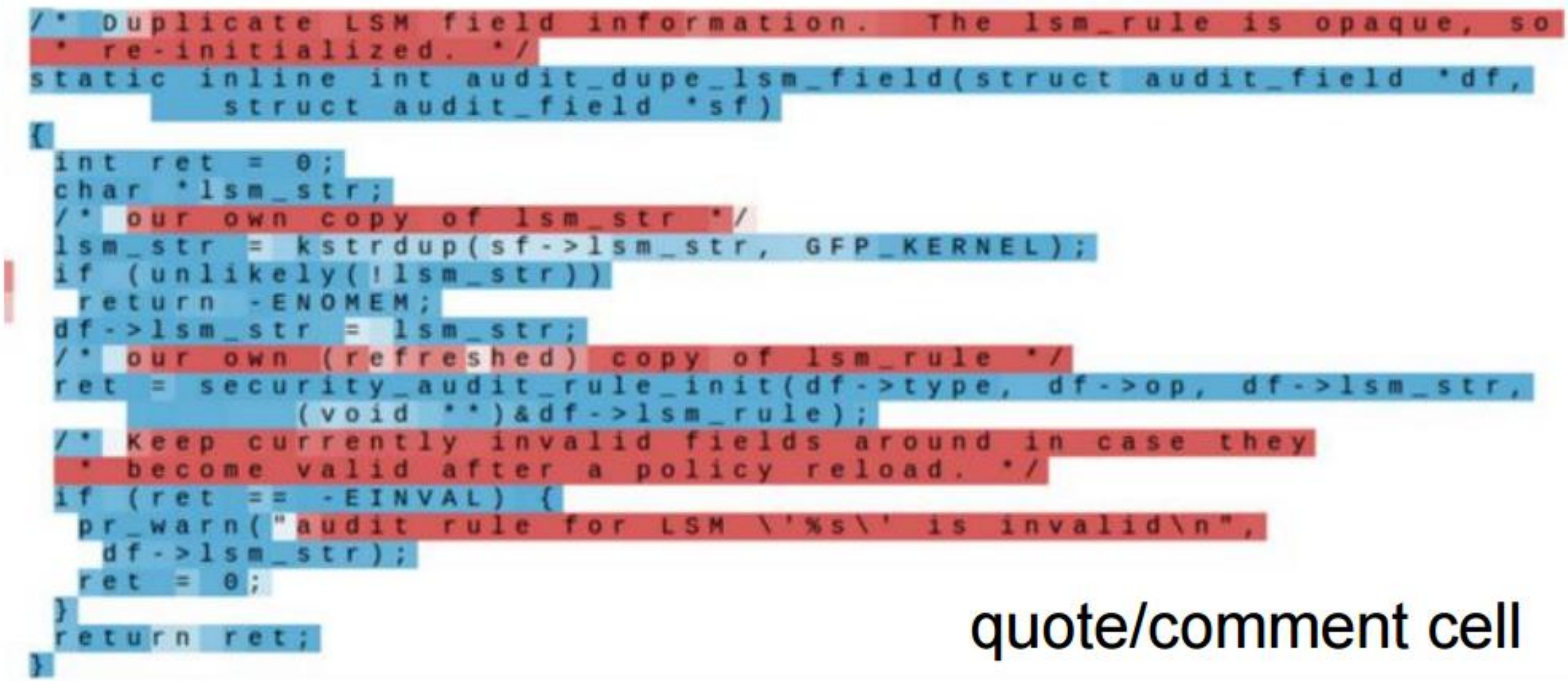
Searching for interpretable cells



line length tracking cell

Recurrent Neural Network

Searching for interpretable cells



quote/comment cell

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Getting started (code ...)

- Karpathy char-rnn : https://gist.github.com/karpathy/d4dee566867f8291f086
- TensorFlow
  - http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html
  - http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
  - https://danijar.com/introduction-to-recurrent-networks-in-tensorflow/
  - http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/
- Lasagne
  - https://github.com/craffel/Lasagne-tutorial/blob/master/examples/tutorial.ipynb