Machine Intelligence & Deep Learning Workshop

Raymond Ptucha, Majid Rabbani, Mark Smith

The Kate Gleason COLLEGE OF ENGINEERING

Recurrent Neural Networks

Raymond Ptucha
June 27-29, 2018
Rochester Institute of Technology
www.rit.edu/kgcoe/cqas/machinelearning

Fair Use Agreement

This agreement covers the use of all slides in this document, please read carefully.

- You may freely use these slides for personal use, if:
  - My name (R. Ptucha) appears on each slide.
- You may freely use these slides externally, if:
  - You send me an email telling me the conference/venue/company name in advance, and which slides you wish to use.
  - You receive a positive confirmation email back from me.
  - My name (R. Ptucha) appears on each slide you use.

Agenda

- Wed, June 27
  - 9-10:30am Regression and Classification
  - 10:30-10:45pm Break
  - 10:45-12:15pm Boosting and SVM
  - 12:15-1:30pm Lunch
  - 1:30-3:30pm Neural Networks and Dimensionality Reduction
  - 3:30-5pm Hands-on Python and Machine Learning
- Thur, June 28
  - 9-10:30am Introduction to deep learning
  - 10:30-10:45pm Break
  - 10:45-12:15pm Convolutional Neural Networks
  - 12:15-1:30pm Lunch
  - 1:30-3:30pm Region and pixel-level convolutions
  - 3:30-5pm Hands-on CNNs
- Fri, June 29
  - 9-10:30am **Recurrent neural networks**
  - 10:30-10:45pm Break
  - 10:45-12:15pm Language and Vision
  - 12:15-1:30pm Lunch
  - 1:30-3:30pm Graph convolutional neural networks; Generative adversarial networks
  - 3:30-5pm Hands-on regional CNNs, RNNs

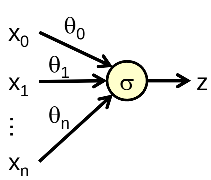
Two Most Important Deep Learning Fields

- Convolutional Neural Networks (CNN)
  - Examine high dimensional input, learn features and classifier simultaneously
- Recurrent Neural Networks (RNN)
  - Learn temporal signals, remember both short and long sequences

Recurrent Neural Networks

- Feed forward Artificial Neural Networks (ANNs) are great at classification, but are limited at predicting future given the past.
- Need framework that determines output based upon current and previous inputs.
- Recurrent or Recursive Neural Networks (RNNs) capture sequential information and are used in speech recognition, activity recognition, NLP, weather prediction, etc.

Adding Recurrence

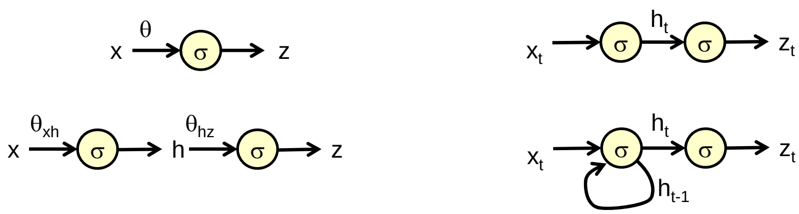


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

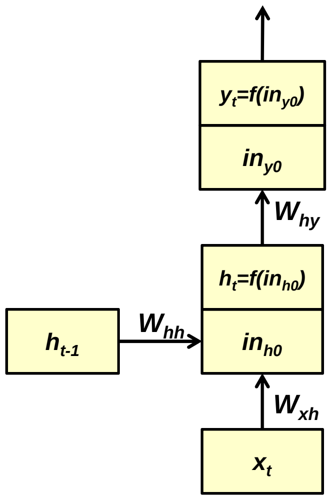
Activation function

$$z = \sigma(x_0\theta_0 + x_1\theta_1 + \ ... + x_n\theta_n) = \sigma\left(\sum_{i=0}^{n} x_i\theta_i\right)$$

$$z = \sigma(\theta^T x)$$



Recurrent Networks

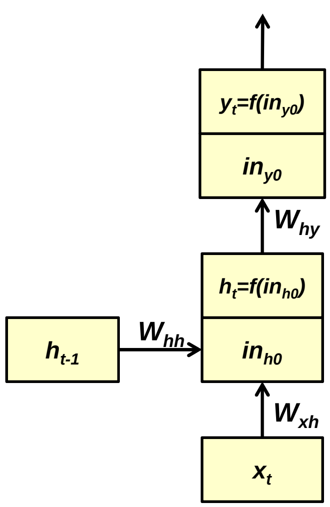


$$in_{h0} = (W_{xh}x_t \qquad\qquad)$$
$$h_t = f(in_{h0})$$

Where:

- f is some activation function
- x_t and h_t are current input and current output values
- W_{xh} is the weight matrix for input, hidden and output stages respectively
- in_{h0} is the input to activation function in hidden and output layers

Recurrent Networks



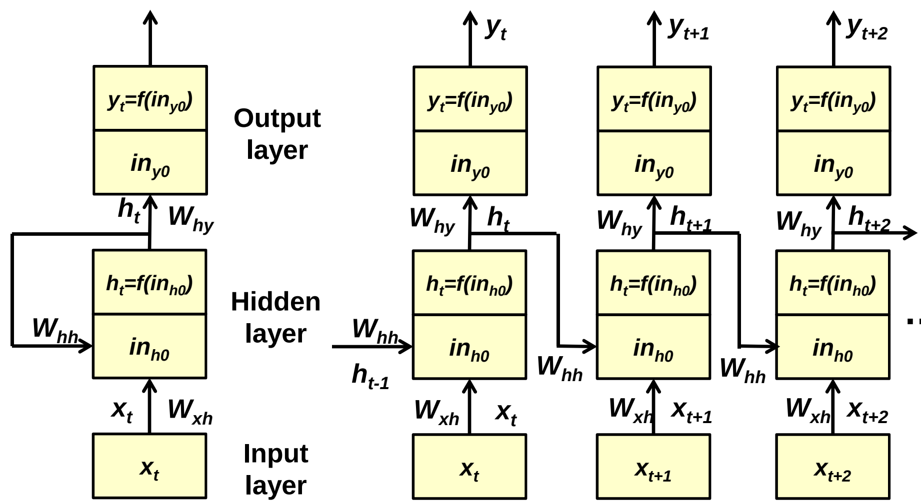
$$in_{h0} = (W_{xh}x_t + W_{hh}h_{t-1})$$
$$h_t = f(in_{h0})$$
$$in_{y0} = W_{hy}h_t$$
$$y_t = f\left(in_{y0}\right)$$

Where:

- f is some activation function
- x_t, h_t, h_{t-1} and y_t are current input, hidden, previous hidden and current output values
- W_{xh}, W_{hh} and W_{hy} are the weight matrices for input, hidden and output stages respectively
- in_{h0} and in_{y0} are the inputs to activation function in hidden and output layers

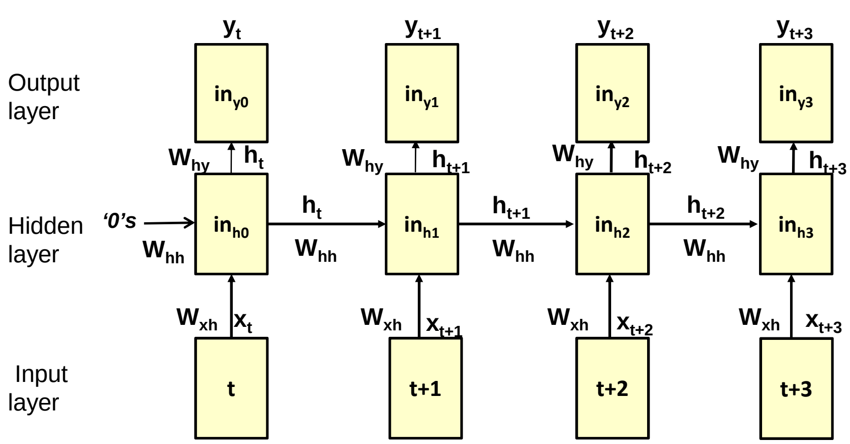
Recurrent Networks

Both figures represent the same architecture



 9

Forward Propagation of Recurrent Networks

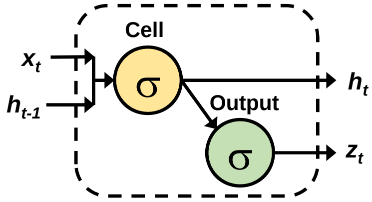


Note: regardless of how many time steps taken, only learning a single W_{xh}, W_{hh}, and W_{hy}. Each are learned via standard back propagation.

 10

Recurrent Networks

Recurrent Neural Network “neuron”

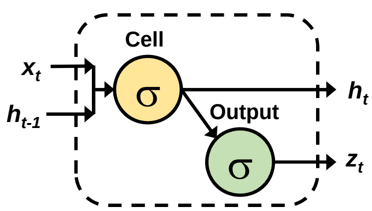


P(next event | previous events)

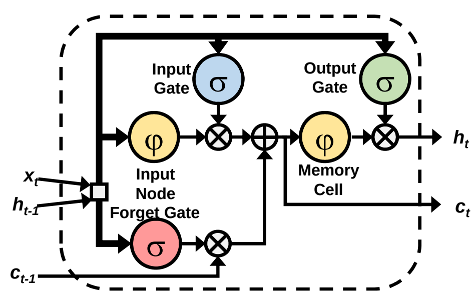
- Unfortunately, these vanilla RNNs don’t always work.
- Can’t store info over long periods of time.
- Suffer from vanishing and/or exploding gradients.

Recurrent Networks

Recurrent Neural Network “neuron”



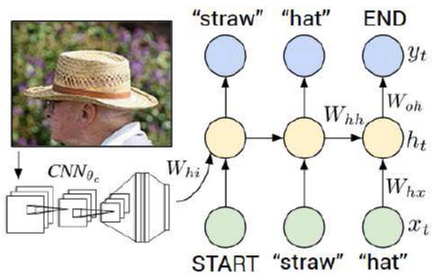
Long Short Term Memory “neuron”



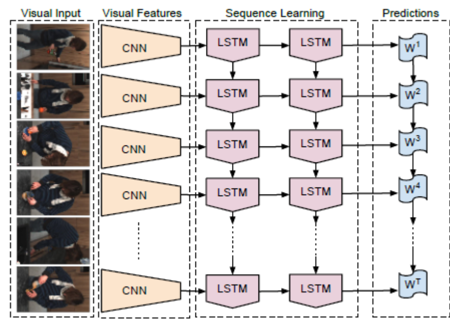
Donahue et al., 2015

- LSTM’s allow read/write/reset functions to neurons.
- Remember past to predict the future- (over long time periods).
- Can have many hidden neurons per layer and many layers.

Recurrent Applications

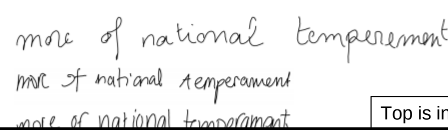


Karpathy, Fei-Fei, 2015



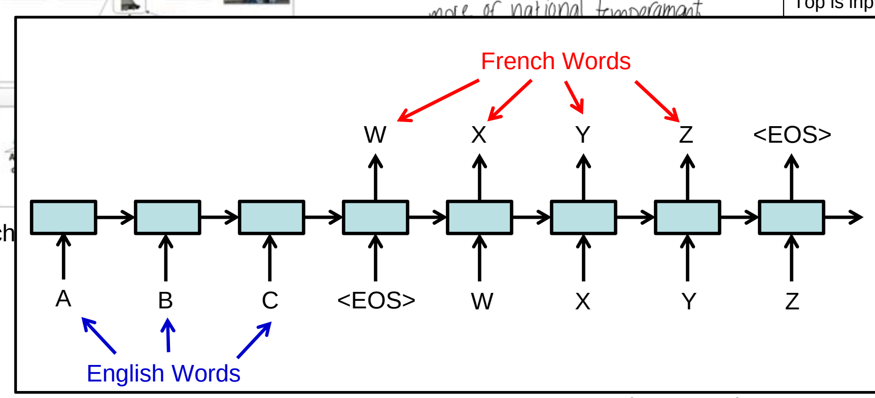
Donahue, et al., 2015

Recurrent Applications



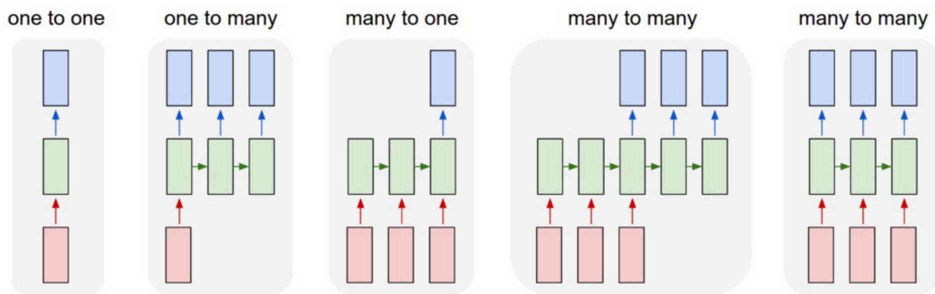
Top is input,

Soch



Sutskever et al., 2014

Many Flavors



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

Vanilla Recurrent Neural Network

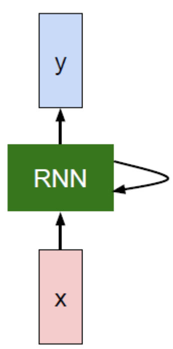
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at
some time step

some function
with parameters W



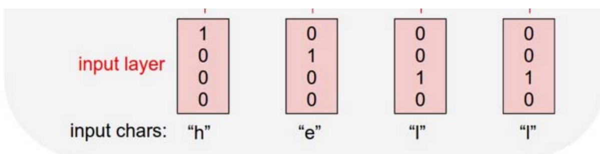
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

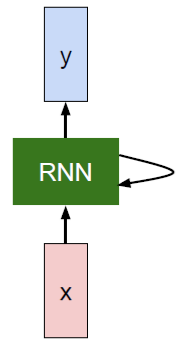
$$y_t = W_{hy}h_t$$

Toy Example

- Character level language model
- Vocabulary: [h,e,l,o]
- Input training example “hello”

Embedding layer: For words, can use one-hot encoding, word2vec, glove, or solve via backprop

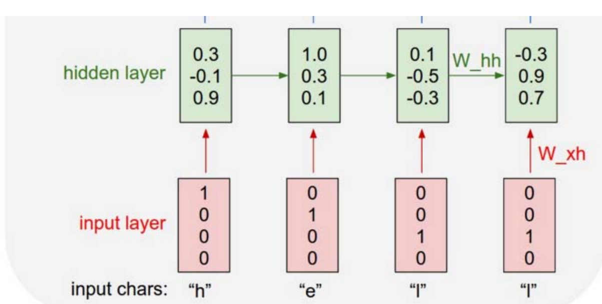


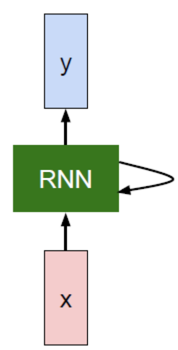


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Toy Example

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

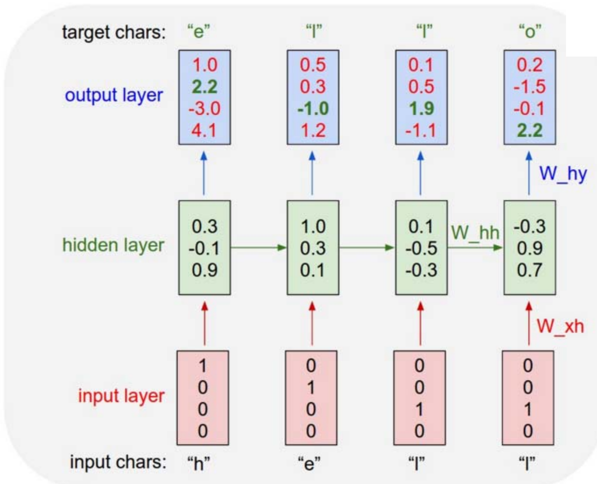


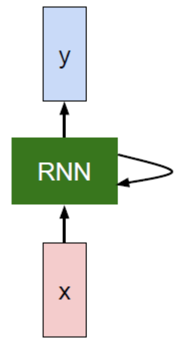


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Toy Example

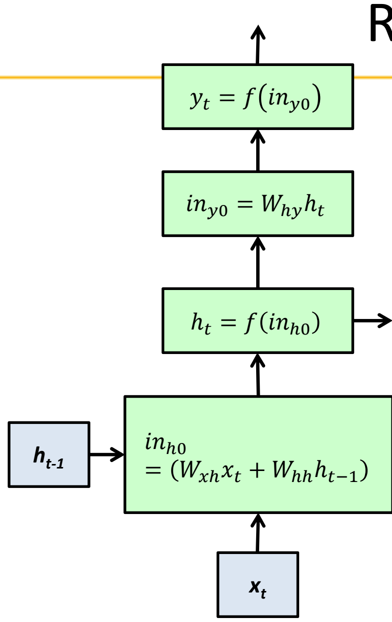
$$y_t = W_{hy} h_t$$





Example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). Shown are the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); want the green numbers to be high and red numbers to be low http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Networks

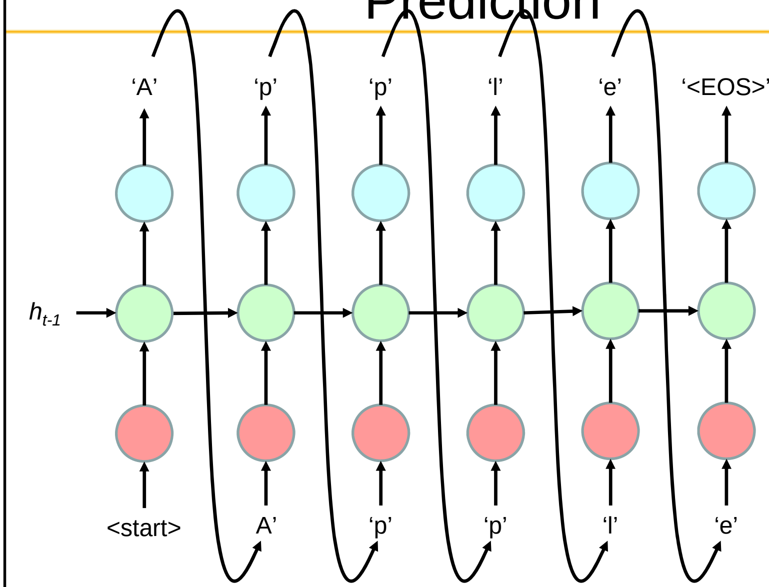


$$in_{h0} = (W_{xh} x_t + W_{hh} h_{t-1})$$
$$h_t = f(in_{h0})$$
$$in_{y0} = W_{hy} h_t$$
$$y_t = f(in_{y0})$$

Where:

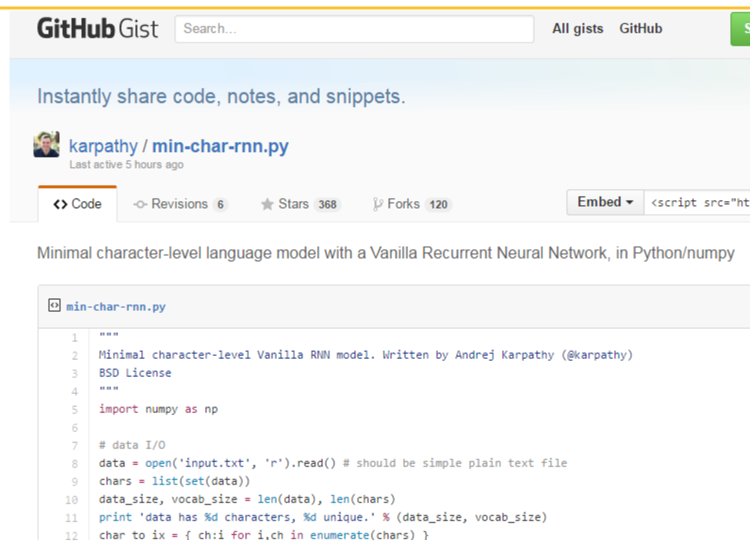
- f is some activation function
- x_t, h_t, h_{t-1} and y_t are current input, hidden, previous hidden and current output values
- W_{xh}, W_{hh} and W_{hy} are the weight matrices for input, hidden and output stages respectively
- in_{h0} and in_{y0} are the inputs to activation function in hidden and output layers

Recurrent Networks for Character Prediction



For this to work, we need to represent characters as some latent vector numerical representation.

Karpathy's minimal character-level language model with a vanilla recurrent neural network in python. This is his RNN in 112 lines of code, https://gist.github.com/karpathy/d4dee566867f8291f086




```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
```

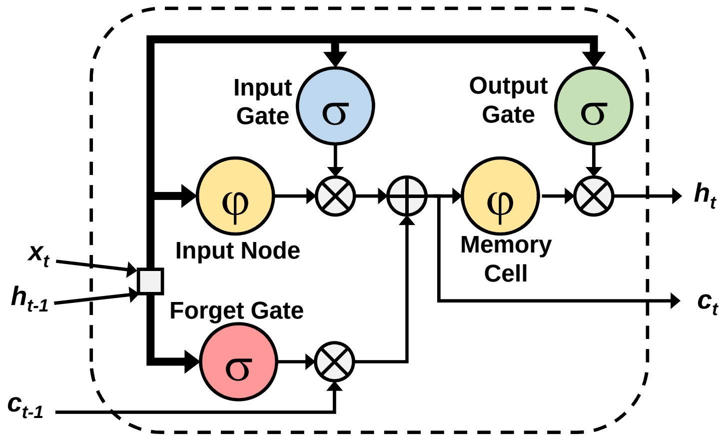
Karpathy's minimal character-level language model with a vanilla recurrent neural network in python. This is his RNN in 112 lines of code, https://gist.github.com/karpathy/d4dee566867f8291f086

- Reads in a training set (say a novel from your favorite author), then uses RNN's to learn character prediction.
- Given the past characters, what is the next most likely character.
- We will see, this can learn words, sentences, phrases, etc. which look like they were written by the author of the novel!
- Given a 't', 'h' is the next most likely char; given 'th', 'e' is the most likely char; given 'the' , ' ' is the most likely char, …

Vanishing gradient problem

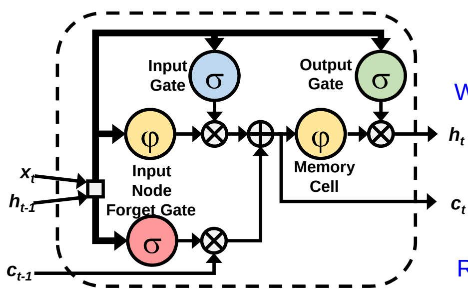
- If the product of weights and derivatives of the activation function is less than 1 the gradients vanish rapidly with the time, else they explode.
- The longer the time lag the greater the chance of gradients exploding or vanishing.
- However weights tend to be initialized with a mean value of 0 and standard deviation of 1.
- Derivatives of common activation functions like sigmoid and tanh also tend to have a value less than or equal to 1 in most cases.
- Hence vanishing gradient problem is observed more commonly.

LSTMs



LSTMs

Convert standard neuron into a complex memory cell



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Write, read, reset governors:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Real input to memory cell:

Input node: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Looks just like our RNN cell!

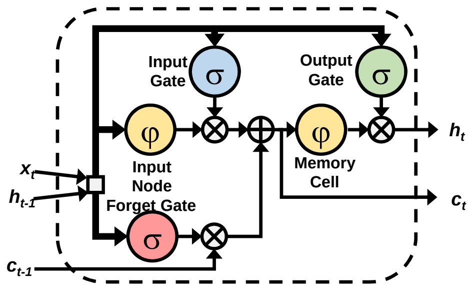
Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

The input node summarizes the input and past output, which will be governed by the input gate.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1}\right)$

Input node: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

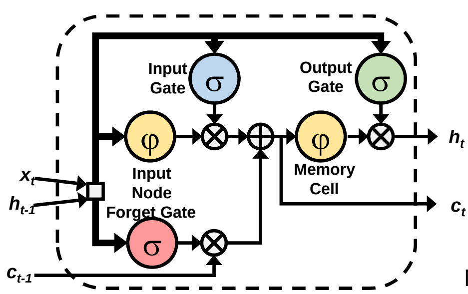
Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

Write: The input gate gives the provision to determine importance of current input and past hidden state.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1}\right)$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

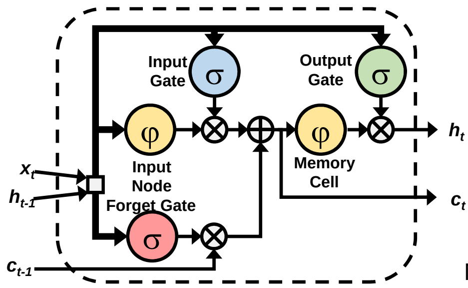
Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

Read: The output gate determines what parts of the cell output are necessary for the next time step.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1}\right)$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

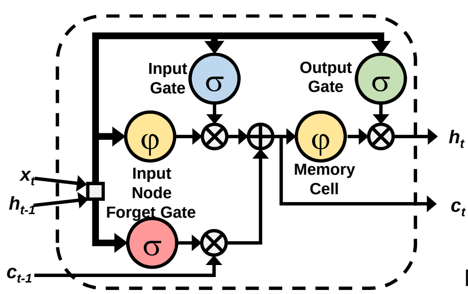
Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

Reset: The forget gate gives the provision for the hidden layer to discard or forget the historical data



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1}\right)$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

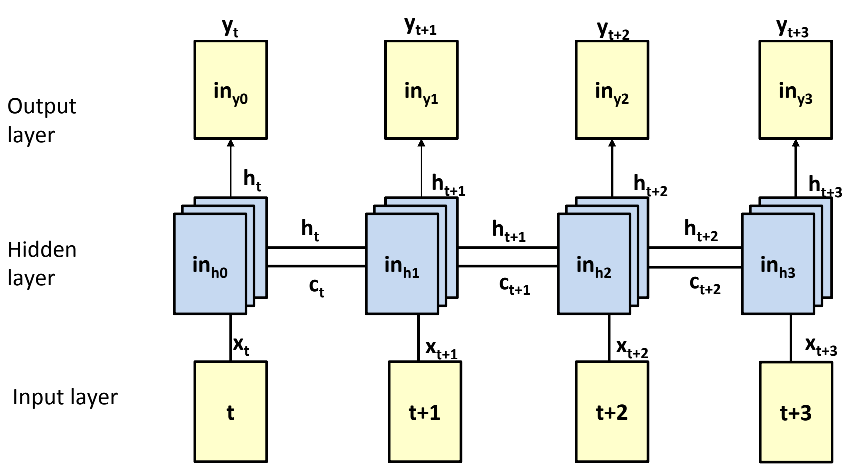
Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

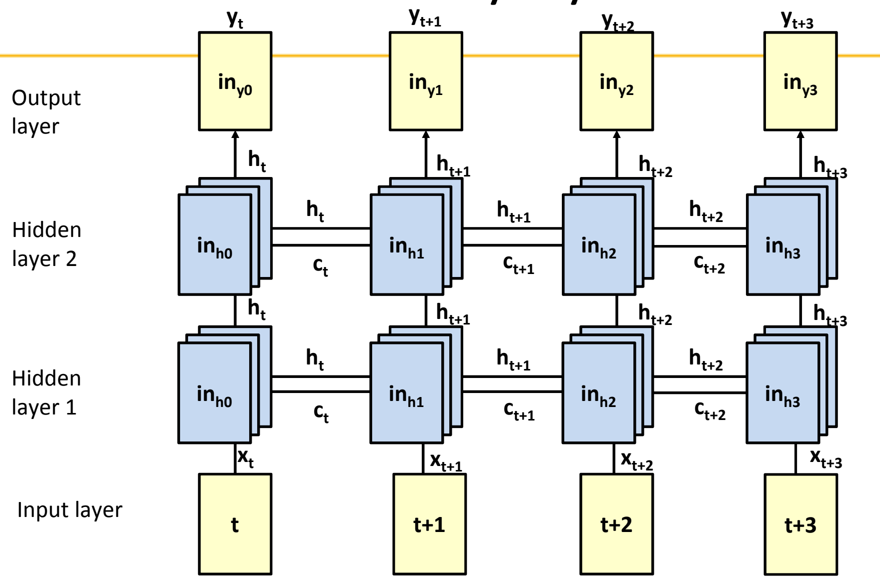
Using LSTMs

- The LSTM memory cells are analogous to a single neuron.
- As such many hundreds of these memory cells are used in a layer, each of which passes its output h_t to the next time step, h_{t+1}.

Same architecture as RNNs, but middle neurons are now LSTM memory cells

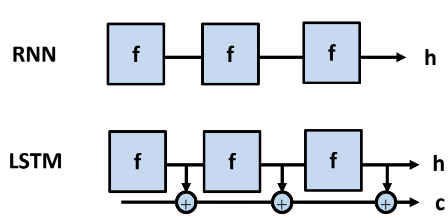


Can do many layers…



Advantages of LSTM over RNN

- As RNN cell step through time, the output of one cell is a transformed version of its input: $h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$
- With LSTMs, if we set the forget gate to 1: $c_t = (f_t c_{t-1} + i_t g_t)$ the memory cell output can pass the entire previous input.
- Very similar to "skip layer" concept of ResNets



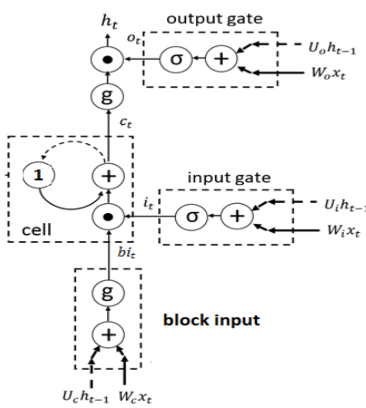
During backpropagation, all gradients forced to go through inverse gradient of *f*.

During backpropagation, gradients of *c* can skip *f*.

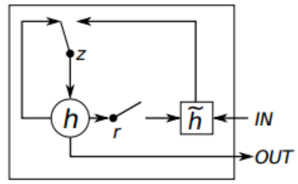
RNN gradients can vanish or explode, LSTM can still explode, so always use gradient clipping (restrict to +/-5).

Many Variants of LSTMs

LSTM with CEC and input, output gates



Simplified variants: Gated Recurrent Unit

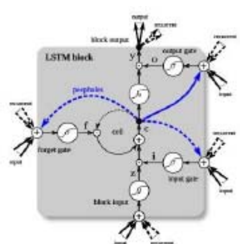


(b) Gated Recurrent Unit

K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014
J. Chung, C. Gulcehre, K. Cho, Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv:1412.3555 (2014)

Many Variants of LSTMs

LSTM: A Search Space Odyssey
Greff et al., 2015



An Empirical Exploration of RNNs
Jozefowicz et al., 2015

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + b_z) \\ r &= \mathrm{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\ r &= \mathrm{sigm}(x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z) \\ r &= \mathrm{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

- Both tried many variants of LSTMs.
- Neither paper was able to find an architecture with substantial advantages over baseline Hochreiter et al. '97 formulation.

GRU vs LSTM

Note: 1 update gate

LSTM uses separate update (i_t) and forget (f_t) gate values; then separate output h_t and memory states c_t.

Gated Recurrent Unit

[Cho et al., EMNLP2014; Chung, Gulcehre, Cho, Bengio, DLUFL2014]

$$h_t = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$
$$\tilde{h} = \tanh(W\,[x_t] + U(r_t \odot h_{t-1}) + b)$$
$$u_t = \sigma(W_u\,[x_t] + U_u h_{t-1} + b_u)$$
$$r_t = \sigma(W_r\,[x_t] + U_r h_{t-1} + b_r)$$

Long Short-Term Memory

[Hochreiter & Schmidhuber, NC1999; Gers, Thesis2001]

$$h_t = o_t \odot \tanh(c_t)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$\tilde{c}_t = \tanh(W_c\,[x_t] + U_c h_{t-1} + b_c)$$
$$o_t = \sigma(W_o\,[x_t] + U_o h_{t-1} + b_o)$$
$$i_t = \sigma(W_i\,[x_t] + U_i h_{t-1} + b_i)$$
$$f_t = \sigma(W_f\,[x_t] + U_f h_{t-1} + b_f)$$

Although computationally simpler, GRUs have similar performance to LSTMs
Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014)

http://web.stanford.edu/class/cs20si/lectures/slides_11.pdf

Learning Shakespeare

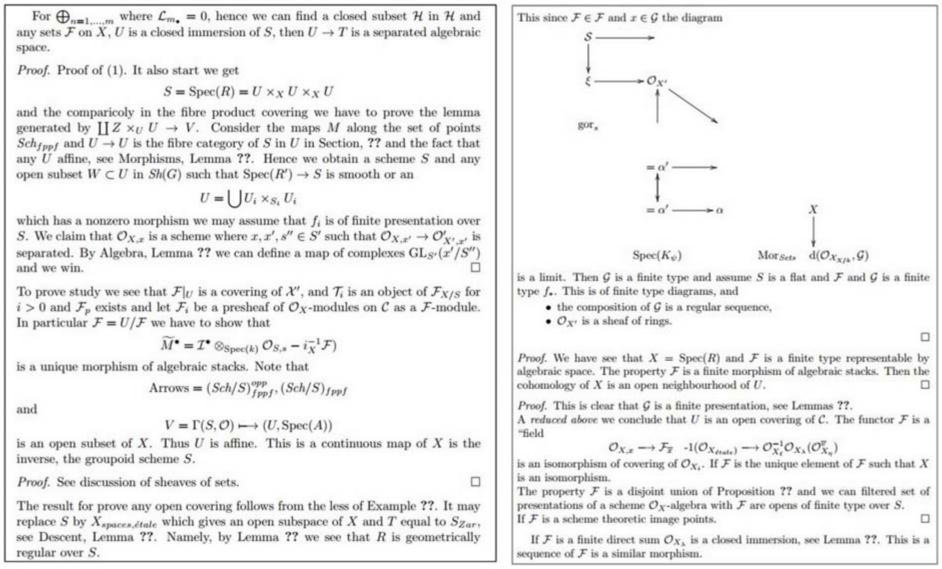
- LSTMs can learn structure and style in the data.
- Karparthy downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file.
- Train a 3-layer LSTM with 512 hidden nodes on each layer.
- After we train the network for a few hours Karpathy obtained samples such as:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.
```

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

Learning LaTeX

- The results above suggest that the model is actually quite good at learning complex syntactic structures.
- Karpathy and Johnston downloaded the raw Latex source file (a 16MB file) of a book on algebraic stacks/geometry and trained a multilayer LSTM.
- Amazingly, the resulting sampled LaTex *almost* compiled.
- They had to step in and fix a few issues manually but then they get plausible looking math:



For $\bigoplus_{n=1,\dots,m}$ 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 $X_{spaces,\acute{e}tale}$ 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.

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

S
$\xi \longrightarrow \mathcal{O}_{X'}$
gor_s
$= \alpha' \longrightarrow$
$= \alpha' \longrightarrow \alpha$
X
$\mathrm{Spec}(K_\psi)$ $\quad$ Mor_{Sets} $\quad$ $\mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})$

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 $f_\bullet$. 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

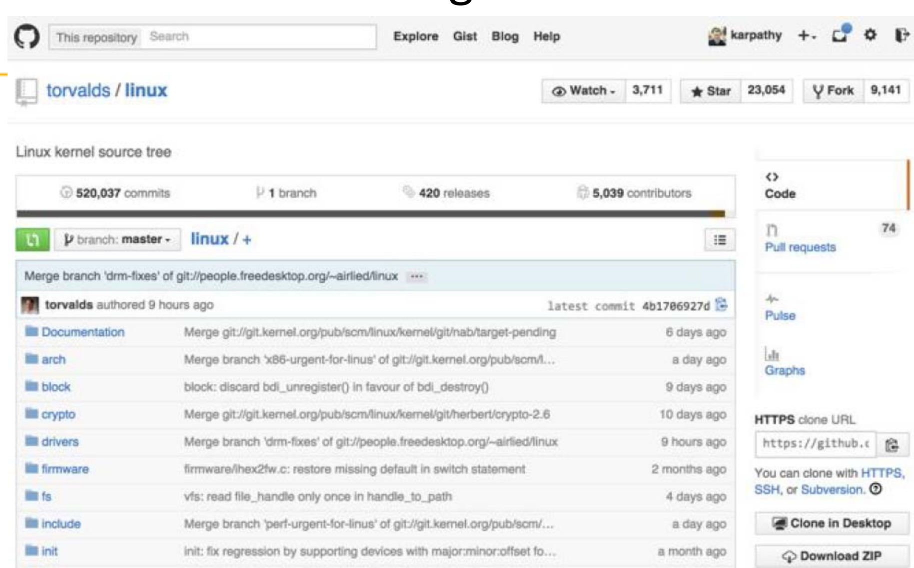
$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \ \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}^{-1}_{X_\ell}\mathcal{O}_{X_\lambda}(\mathcal{O}^{\overline{v}}_{X_\eta})$$

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

Learning C Code



cs231n, lecture 10

```
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 ");
}
```



```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

cs231n, lecture 10

The evolution of samples while training

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

- Let's examine how sampled text evolves while the LSTM model trains.
- For example, Karpathy trained an LSTM of Leo Tolstoy's War and Peace and then generated samples every 100 iterations of training.
- At iteration 100 the model samples random jumbles:

```
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
```

- The model is starting to get an idea about words separated by spaces- except sometimes it inserts two spaces. It also doesn't know that comma is almost always followed by a space.

The evolution of samples while training

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

- At 300 iterations we see that the model starts to get an idea about quotes and periods:

```
"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."
```

- The words are now also separated with spaces and the model starts to get the idea about periods at the end of a sentence.
- At iteration 500:

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```


The evolution of samples while training

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

- The model has now learned to spell the shortest and most common words such as "we", "He", "His", "Which", "and", etc. At iteration 700 we're starting to see more and more English-like text emerge:

```
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.
```

- At iteration 1200 we're now seeing use of quotations and question/exclamation marks. Longer words have now been learned as well:

```
"Kite vouch!" he repeated by her
door. "But I would be done and quarts, feeling, then, son is people...."
```

© 2018 Ray Ptucha, Rochester Institute of Technology

The evolution of samples while training

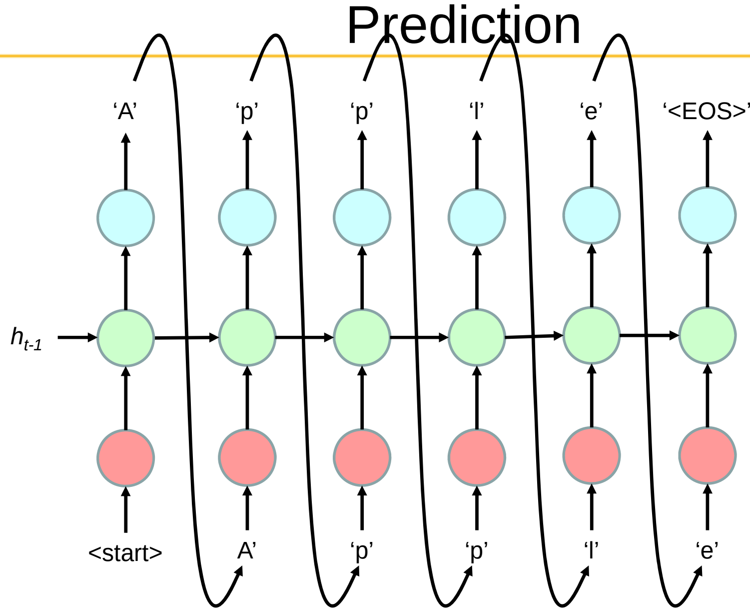
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- Properly spelled words, quotations, names, and so on occur by about iteration 2000:

```
"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.
```

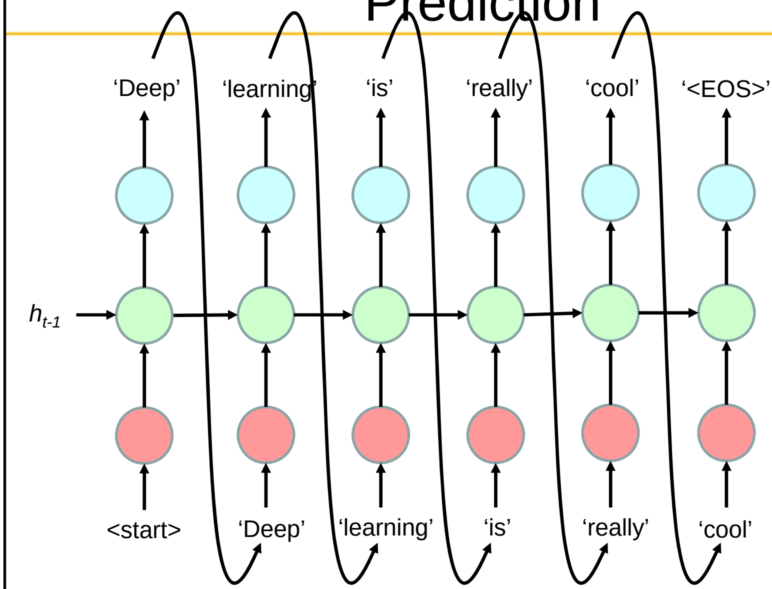
- The model first discovers the general word-space structure and then rapidly starts to learn the words; First starting with the short words and then eventually the longer ones. Topics and themes that span multiple words (and in general longer-term dependencies) start to emerge only much later.

Recurrent Networks for Character Prediction



For this to work, we need to represent characters as some latent vector numerical representation.

Recurrent Networks for Word Prediction



For this to work, we need to represent words as some latent vector numerical representation.

Word2vec

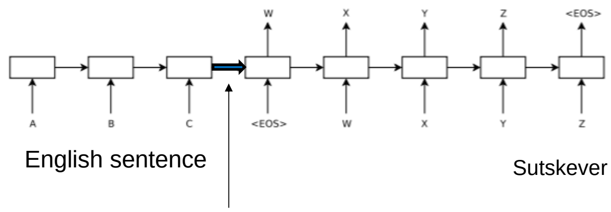
- In the simplest form, we can start with a one-hot encoded vector of all words, and then learn a model which converts to a lower dimensional representation.
- Word2vec, glove, and skip-gram are popular metrics which encode words to a latent vector representation (~300 dimensions).

- Now we have a way to represent images, characters, and words as vectors.

Sent2vec

- In the English to French translation, we have:

French sentence



English sentence

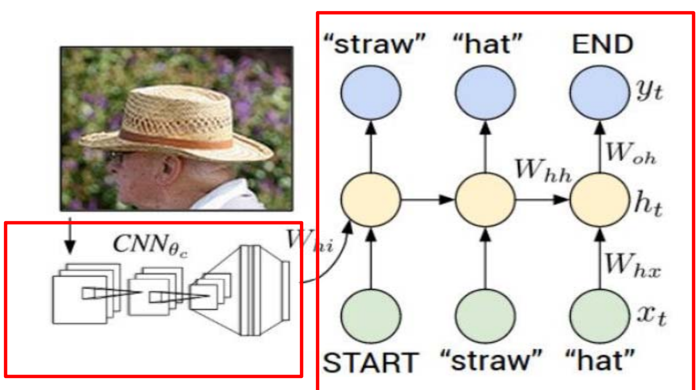
Sutskever et al., NIPS ‘14

…but wait, this point in the RNN is a representation (sent2vec) of all the words in the English sentence!

- Now we have a way to represent images, characters, words, and sentences as vectors…can extend to paragraphs and documents…

Image Captioning

RNN takes in a latent representation of an image, and generates a sequence.



CNN helps represent an image as a numeric value. (image2vec)

Karpathy & Li, CVPR'15

- We may have 50K words. Instead of one-hot encoding, we learn an embedding for each word.

<word1>

$y_t = f(W_{hy}h_t)$

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

<start>

- Glove embedding (300 long vector/word) is very popular.
- Alternately, can learn embedding- learn a matrix which goes from (50K) one-hot to 300, ie: $W_{ix} \in R^{50K\times 300}$
- Embedding and unembedding can be learned or inverses of one another.

Note: Word is sampled from distribution of word probabilities

<word1>

$y_t = f(W_{hy}h_t)$

h_{t-1} → $h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$

v

<start>

v could be FC6, FC7, conv5, conv4, …; or a combination of above

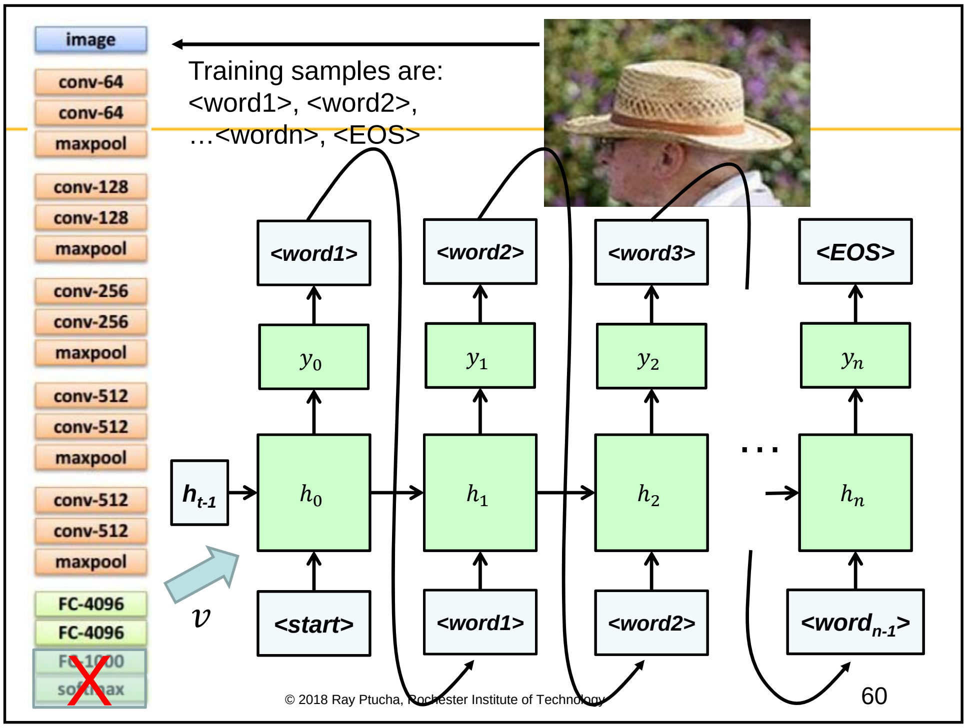


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
Training samples are:
<word1>, <word2>,
…<wordn>, <EOS>
<word1>
<word2>
<word3>
<EOS>
y_0
y_1
y_2
y_n
h_{t-1}
h_0
h_1
h_2
…
h_n
v
<start>
<word1>
<word2>
<word$_{n-1}$>
© 2018 Ray Ptucha, Rochester Institute of Technology
60

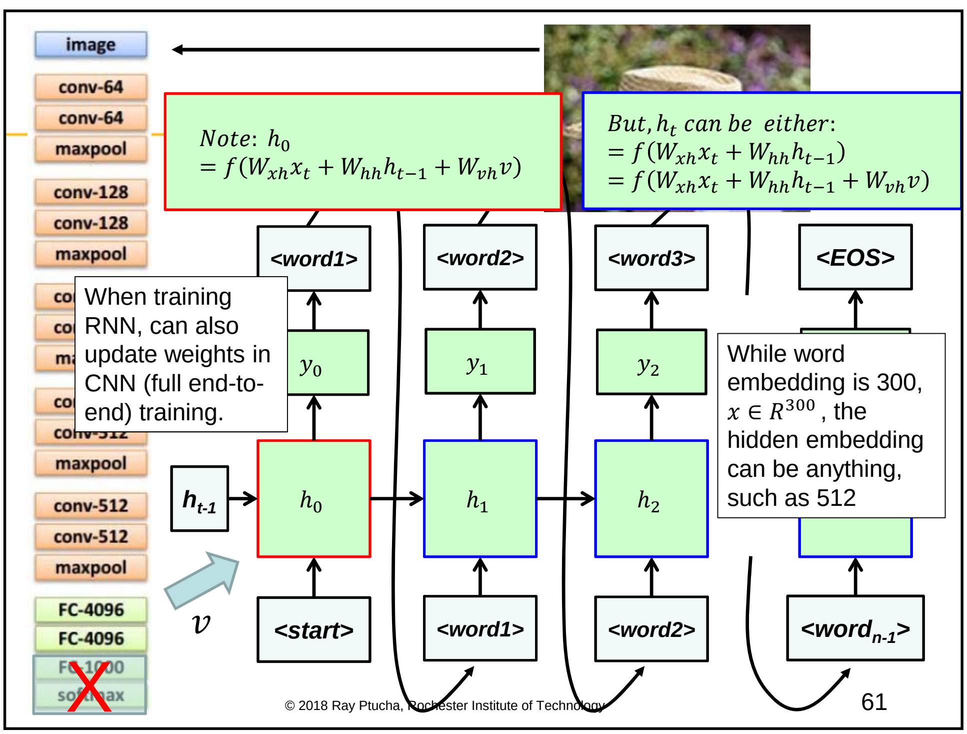


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
Note: $h_0 = f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$
But, h_t can be either: $= f(W_{xh}x_t + W_{hh}h_{t-1})$ $= f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$
<word1>
<word2>
<word3>
<EOS>
When training RNN, can also update weights in CNN (full end-to-end) training.
y_0
y_1
y_2
While word embedding is 300, $x \in R^{300}$, the hidden embedding can be anything, such as 512
h_{t-1}
h_0
h_1
h_2
v
<start>
<word1>
<word2>
<word$_{n-1}$>
© 2018 Ray Ptucha, Rochester Institute of Technology
61

Data for Captioning

- Flickr8K
  - 8,000 images, from Flickr website, each with five captions
  - http://nlp.cs.illinois.edu/HockenmaierGroup/8k-pictures.html
- Flickr30K
  - 31,783 images, from Flickr website, each with five captions
  - http://shannon.cs.illinois.edu/DenotationGraph/
- MSCOCO
  - 80,000 training images, each with five captions
  - http://mscoco.org/

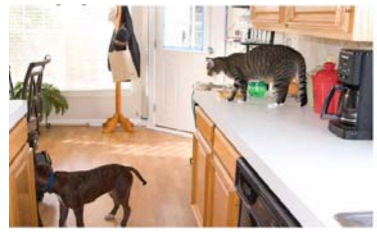
Captioning Datasets

Amazon mechanical turkers do all labeling

https://www.mturk.com/mturk/welcome

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

MSCOCO Dataset



- A pile of wooden boxes filled with fruits and vegetables.
- An assortment of fruit in buckets for sale in a shop.
- An outdoor fruit stand with various types of fruits for sale.
- A display of crates of fruit on a city street.
- There are many crates with fruit and vegetables.

- A cat stands on a counter while a dog stands on the floor.
- A cat on the kitchen counter is looking down at a dog.
- A cat is looking at a dog rummage in the garbage.
- A cat on the counter and a dog on the ground in the kitchen.
- A cat stalking a dog on the kitchen floor.

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

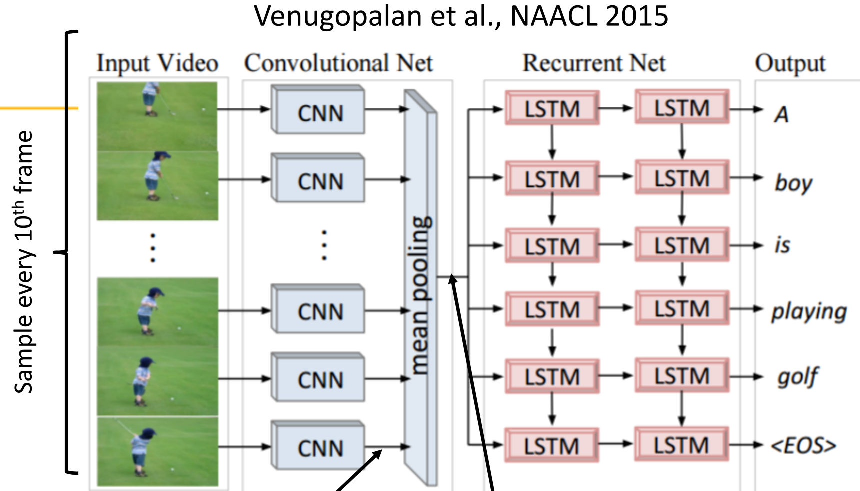
Karpathy'15

Video Data for Captioning

MSVD: Microsoft Video Description Dataset
MSR-VTT: Microsoft Research -Video to Text
M-VAD: Movie description dataset M-VAD

	MSVD	MSR-VTT	M-VAD
#sentences	80,827	200,000	54,997
#sent. per video	~42	20	~1-2
vocab. size	9,729	24,282	16,307
avg. length	10.2s	14.8s	5.8s
#train video	1,200	6,513	36,921
#val. video	100	497	4,651
#test video	670	2,990	4,951

Venugopalan et al., NAACL 2015

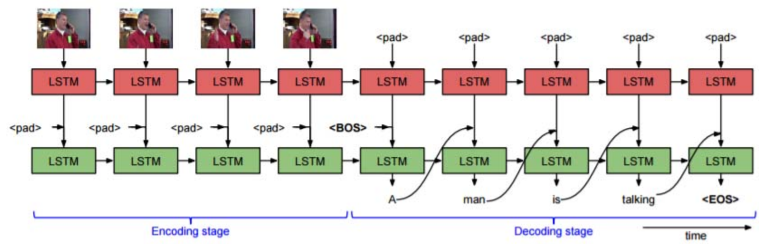


AlexNet 4K FC7

Mean pooling over f frames $V = \frac{1}{f}\sum_{i=1}^{f}[v_1^i, v_2^i, \cdots, v_{4096}^i]$

Pre-train on alternate caption datasets, fine tune to your dataset

Video Captioning



SV2T, Venugopalan, 2015

- Single LSTM for both encode and decode state.
- Two layer LSTM, 1000 hidden units each:
  - First LSTM learns video concepts
  - Second LSTM concentrates on language details.

Thank you!!

Ray Ptucha
rwpeec@rit.edu

https://www.rit.edu/mil