

Recurrent Networks

Contains material from:

Andrej Karpathy's blog:

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

Christopher Olah's blog

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

Geoffrey Hinton's lecture:

[https:](https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf)

[//www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf](https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf)

Recurrent Neural Network – vector notation

A simple example of a recurrent MLP:

- ▶ Input: \vec{x}
- ▶ Hidden (state): \vec{h}
- ▶ Output: \vec{y}
- ▶ Matrices U, W, V

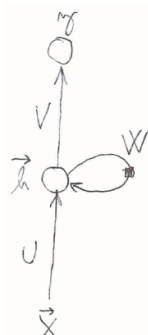
$$\vec{h} = \sigma(U\vec{x} + W\vec{h})$$

Here σ is an activation function (applied component-wise), typically sigmoidal or ReLU.

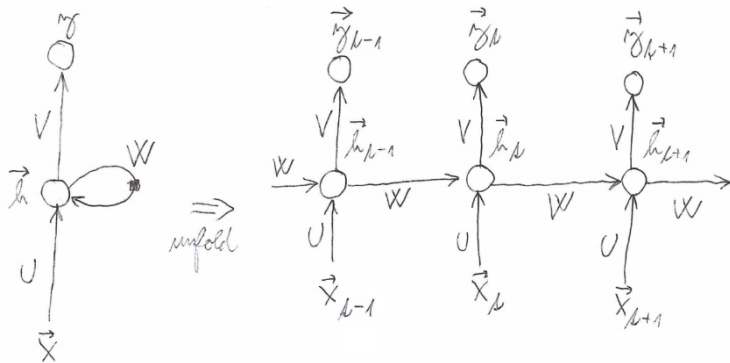
$$\vec{y} = \sigma(V\vec{h})$$

Here σ is typically softmax (so that we get probabilities) or sigmoidal.

In what follows I will use σ to denote arbitrary activation functions, keep in mind that each neuron may have different activation function.



Recurrent Neural Network – sequence modeling



- ▶ Input sequence: $\vec{x}_1, \dots, \vec{x}_T$ of vectors.
- ▶ Output sequence: $\vec{y}_1, \dots, \vec{y}_T$ of vectors obtained by

$$\vec{h}_t = \sigma(U\vec{x}_t + W\vec{h}_{t-1})$$

$$\vec{y}_t = \sigma(V\vec{h}_t)$$

RNN – Component-wise

Denote:

$$\vec{x} = (x_1, \dots, x_M)$$

$$\vec{h} = (h_1, \dots, h_H)$$

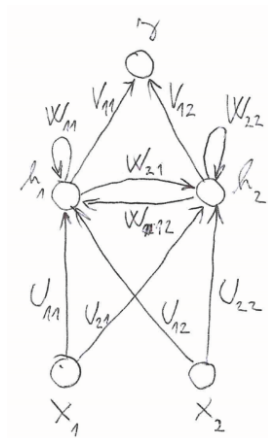
$$\vec{y} = (y_1, \dots, y_N)$$

For all $k = 1, \dots, H$

$$h_k = \sigma \left(\sum_{k'=1}^M U_{kk'} x_{k'} + \sum_{k'=1}^H W_{kk'} h_{k'} \right)$$

For all $k = 1, \dots, N$

$$y_k = \sigma \left(\sum_{k'=1}^H V_{kk'} h_{k'} \right)$$



RNN – Component-wise – Unfolding

- ▶ Input sequence: $\mathbf{x} = \vec{x}_1, \dots, \vec{x}_T$

$$\vec{x}_t = (x_{t1}, \dots, x_{tM})$$

- ▶ Hidden sequence:

$$\mathbf{h} = \vec{h}_0, \vec{h}_1, \dots, \vec{h}_T$$

$$\vec{h}_t = (h_{t1}, \dots, h_{tH})$$

We have $\vec{h}_0 = (0, \dots, 0)$ and

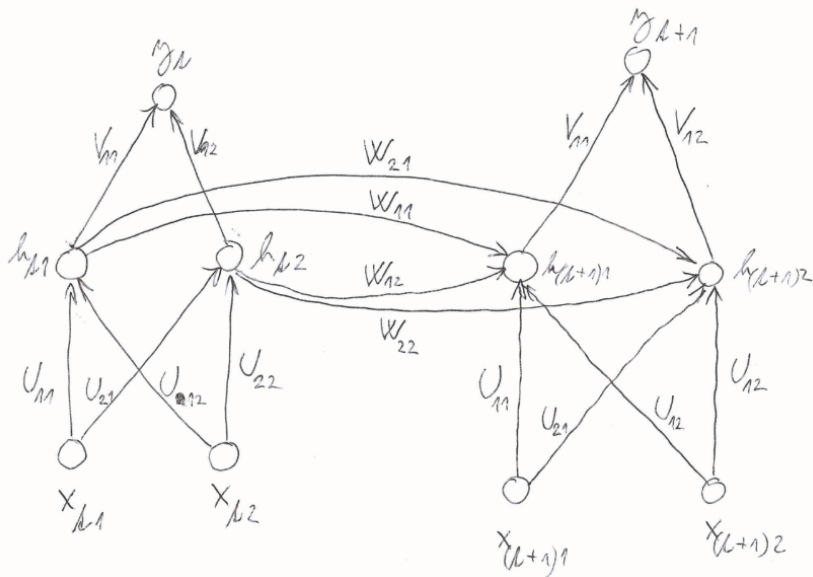
$$h_{tk} = \sigma \left(\sum_{k'=1}^M U_{kk'} x_{tk'} + \sum_{k'=1}^H W_{kk'} h_{(t-1)k'} \right)$$

- ▶ Output sequence: $\mathbf{y} = \vec{y}_1, \dots, \vec{y}_T$

$$\vec{y}_t = (y_{t1}, \dots, y_{tN})$$

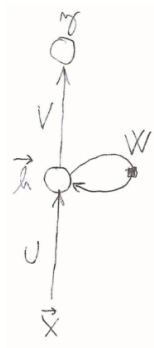
We have $y_{tk} = \sigma \left(\sum_{k'=1}^H V_{kk'} h_{tk'} \right)$

RNN – Component-wise – Unfolding



RNN – Comments

- ▶ \vec{h}_t is the memory of the network, captures what happen in all previous steps (with decaying quality).
- ▶ RNN **shares weights** U, V, W across all steps.
Note the similarity to convolutional networks where the weights were shared spatially over images, here they are shared temporally over sequences.
- ▶ RNN can deal with **sequences of variable length**.
Compare with MLP which accepts only fixed-dimension vectors on input.



RNN – language modelling (toy example)

RNN generating text character by character.

Models the probability distribution of the next character in a given sequence.

Learns the distribution from a huge number of sequences.

For simplicity, assume 4 letter alphabet: h, e, l, o

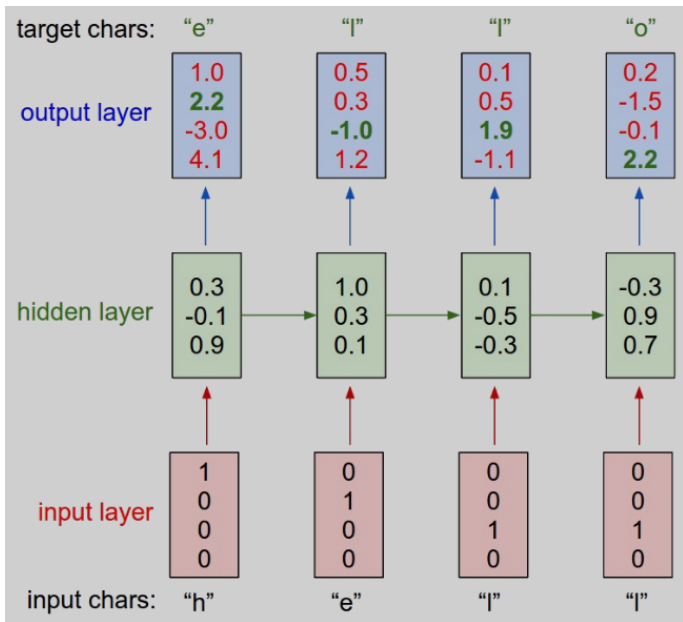
Encode letters using one-hot encoding, e.g. e is $(0, 1, 0, 0)$.

Output layer: Softmax

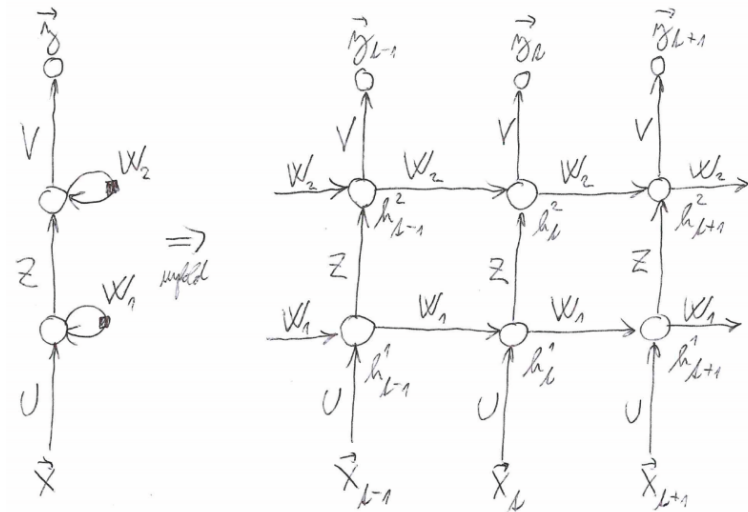
Error: Cross-entropy

Training: Gradient descent (simply unfold in time, see later)

RNN – language modelling (toy example)

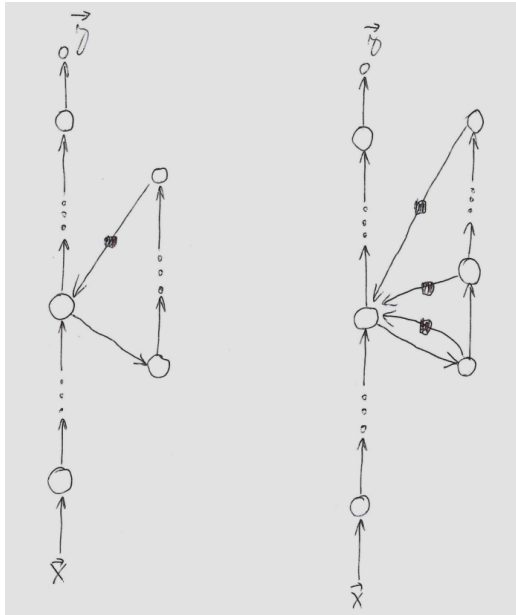


Deeper RNN



Two hidden layers ... may be arbitrary number.

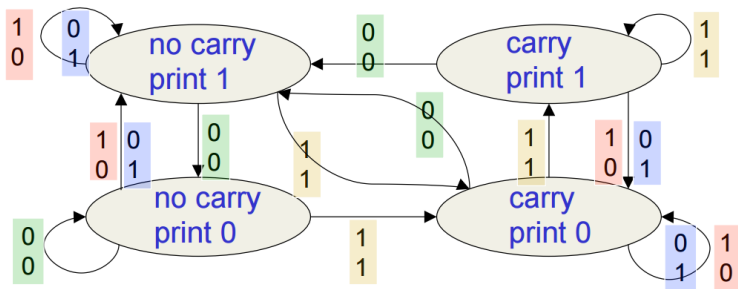
... and deeper



Binary addition – another toy

- ▶ MLP can be trained to do binary addition, but there are obvious regularities that it cannot capture efficiently:
 - ▶ We must decide in advance the maximum number of digits in each number.
 - ▶ The processing applied to the beginning of a long number does not generalize to the end of the long number because it uses different weights.
- ▶ As a result, feedforward nets do not generalize well on the binary addition task.

Binary addition – another toy



A finite transducer, in every step reads a pair of numbers of $\{0, 1\}^2$ and prints an output number of $\{0, 1\}$.

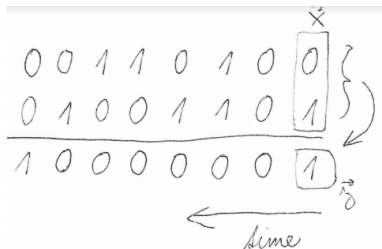
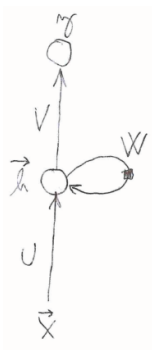
The network should imitate activity of the automaton.

Three hidden neurons should be enough.

Binary addition – another toy

The network has two input neurons and one output neuron.

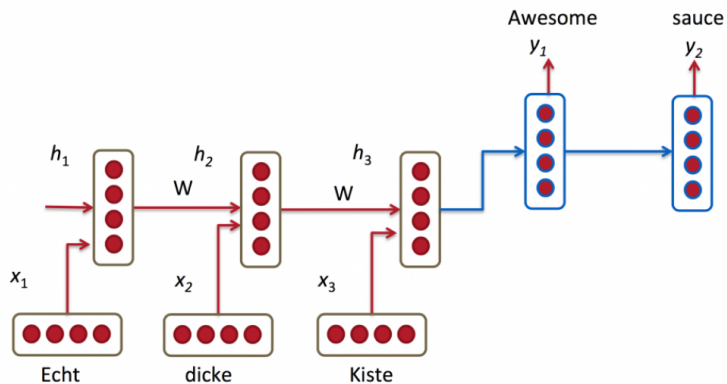
Three hidden neurons are sufficient for binary addition.



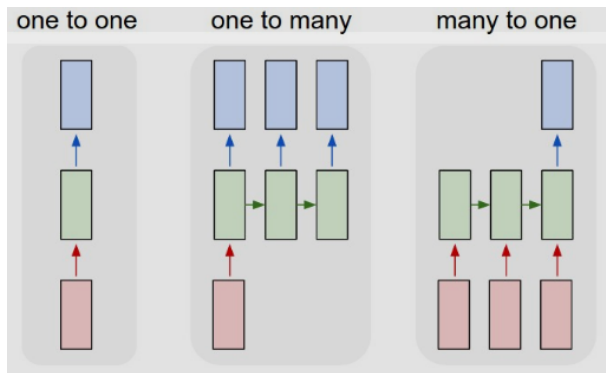
Binary addition – another toy

- ▶ RNN learns four distinct patterns of activity for the 3 hidden neurons. These patterns correspond to the nodes in the finite state automaton.
 - ▶ Do not confuse units in a neural network with nodes in a finite state automaton. Nodes are like activity vectors.
 - ▶ The automaton is restricted to be in exactly one state at each time. The hidden units are restricted to have exactly one vector of activity at each time.
- ▶ A recurrent network can emulate a finite state automaton, but it is exponentially more powerful. With N hidden neurons it has 2^N possible binary activity vectors (but only N^2 weights)
 - ▶ This is important when the input stream has two separate things going on at once.
 - ▶ A finite state automaton needs to square its number of states.
 - ▶ An RNN needs to double its number of units.

Machine translation with RNN

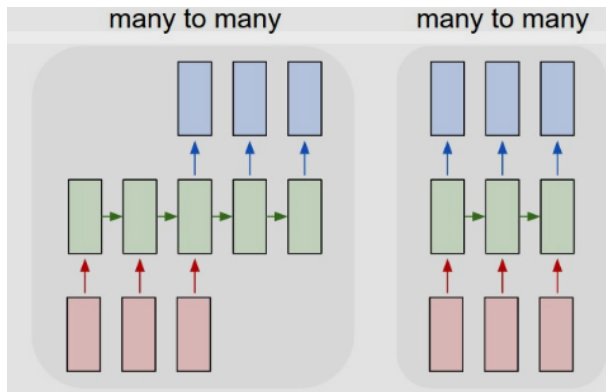


Variants of RNN



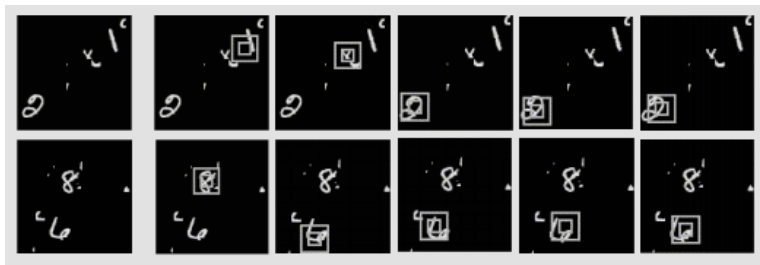
- ▶ one to one: Standard MLP, single vector in, single out.
- ▶ one to many: Single vector in, sequence out.
Image captioning: image in, sentence out.
- ▶ many to one: Sequence in, single vector out.
Sentiment analysis: sentence in, sentiment (positive/negative) out.

Variants of RNN



- ▶ many to many: Sequence in, sequence out.
Machine translator: English sentence in, Czech out (may have different lengths).
- ▶ many to one: Synced sequences in and out.
Video classification, where we wish to label each frame of the video.

Image recognition: recurrent attention model



The recurrent network tells a "glimpse" network, where to look.
The state of the recurrent network changes based on location and actual perception in the location.

RNN – Learning

We consider a fixed training example (\mathbf{x}, \mathbf{d}) where

- ▶ $\mathbf{x} = \vec{x}_1, \dots, \vec{x}_T$ is a given input sequence, here

$$\vec{x}_t = (x_{t1}, \dots, x_{tN})$$

- ▶ $\mathbf{d} = \vec{d}_1, \dots, \vec{d}_m$ is a given sequence of desired values.

$$\vec{d}_t = (d_{t1}, \dots, d_{tM})$$

Unfolding the RNN for \mathbf{x} gives a sequence of hidden states:

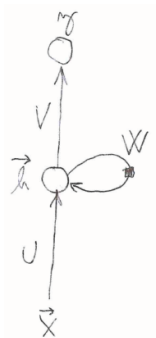
$$\vec{h}_0, \vec{h}_1, \dots, \vec{h}_T \quad \text{each } \vec{h}_t = (h_{t1}, \dots, h_{tH})$$

here $\vec{h}_0 = (0, \dots, 0)$ and a sequence of output values:

$$\vec{y}_1, \dots, \vec{y}_t \quad \text{each } \vec{y}_t = (y_{t1}, \dots, y_{tM})$$

Error function (e.g. squared error): $E_{(\mathbf{x}, \mathbf{d})} = \sum_{t=1}^T \sum_{k=1}^M (y_{tk} - d_{tk})^2$

Learning – backpropagation through time



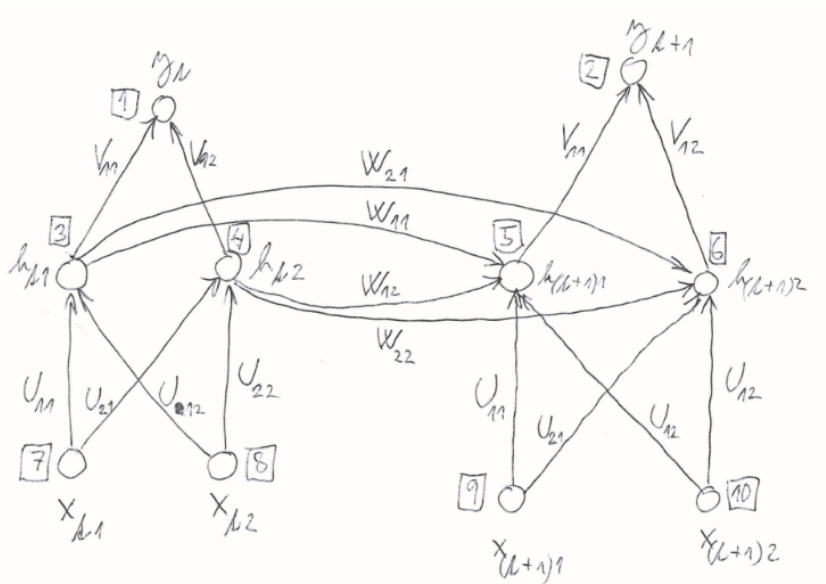
RNN training algorithm is easy to obtain:

- ▶ Unfold the RNN for several time steps.
- ▶ Consider it to be a (deep) MLP.
- ▶ Train with gradient descent.

However, one has to keep in mind that U, W, V are **shared** in all time instants.

To simplify training and to connect with MLP and convolutional networks, we abstract all neurons in the **unfolding**, and denote them by indices i, j , etc. as before for MLP.

Learning – backpropagation through time



Recurrent networks – unfolding

Let us make neurons of the **unfolding** anonymous:

- ▶ Denote
 - ▶ X a set of *input* neurons of the unfolding
 - ▶ Y a set of *output* neurons of the unfolding
 - ▶ Z a set of *all* neurons of the unfolding ($X, Y \subseteq Z$)
- ▶ individual neurons of the unfolding denoted by indices i, j
 - ▶ ξ_j is the inner potential of the neuron j *after the computation stops*
 - ▶ σ_j is the activation function of j
 - ▶ y_j is the output of the neuron j *after the computation stops*
- ▶ w_{ji} is the weight of the connection **from i to j**
- ▶ j_{\leftarrow} is a set of all i such that j is adjacent from i (i.e. there is an arc **to** j from i)
- ▶ j_{\rightarrow} is a set of all i such that j is adjacent to i (i.e. there is an arc **from** j to i)
- ▶ j_{share} is a set of neurons sharing weights with j
 j_{share} consists of all incarnations of the same neuron of the RNN in different time instants t .

Gradient descent (single training example)

Consider the single training example (\mathbf{x}, \mathbf{d}) .

In the case of SGD, minibatches of such pairs are used, their errors are averaged.

The algorithm computes a sequence of weight vectors $\vec{\mathbf{w}}^{(0)}, \vec{\mathbf{w}}^{(1)}, \vec{\mathbf{w}}^{(2)}, \dots$

Do not forget that these are weights in the RNN, possibly shared by some neurons in the unfolding.

- ▶ weights in $\vec{\mathbf{w}}^{(0)}$ are randomly initialized to values close to 0
- ▶ in the step $\ell + 1$ (here $\ell = 0, 1, 2 \dots$), weights $\vec{\mathbf{w}}^{(\ell+1)}$ are computed as follows:

$$\vec{\mathbf{w}}^{(t+1)} = \vec{\mathbf{w}}^{(t)} + \Delta \vec{\mathbf{w}}^{(t)}$$

where

$$\Delta \vec{\mathbf{w}}^{(t)} = -\varepsilon(t) \cdot \nabla E_{(\mathbf{x}, \mathbf{d})}(\vec{\mathbf{w}}^{(t)})$$

Backprop

$\nabla E_{(\mathbf{x}, \mathbf{d})}(\vec{w}^{(t)})$ is a vector of all partial derivatives $\frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial w_{ji}}$.

- ▶ First, switch from derivatives w.r.t. w_{ji} to derivatives w.r.t. y_j :

$$\frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial w_{ji}} = \sum_{r \in J_{\text{share}}} \frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial y_r} \cdot \sigma'_r(\xi_r) \cdot y_i$$

- ▶ for every $j \in Y$:

$$\frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial y_j} = y_j - d_j$$

This holds for the mean-square error, for other error functions the derivative w.r.t. outputs will be different.

- ▶ for every $j \in Z \setminus Y$:

$$\frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial y_j} = \sum_{r \in J^{\rightarrow}} \frac{\partial E_{(\mathbf{x}, \mathbf{d})}}{\partial y_r} \cdot \sigma'_r(\xi_r) \cdot w_{rj}$$

In the notation of RNN

$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial V_{kk'}} = \sum_{t=1}^T \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial y_{tk}} \cdot \sigma' \cdot h_{tk}$$

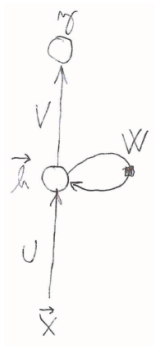
$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial W_{kk'}} = \sum_{t=1}^T \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{tk}} \cdot \sigma' \cdot h_{(t-1)k}$$

$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial U_{kk'}} = \sum_{t=1}^T \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{tk}} \cdot \sigma' \cdot x_{tk}$$

Backprop:

$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial y_{tk}} = y_{tk} - d_{tk}$$

$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{tk}} = \sum_{k'=1}^N \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial y_{tk'}} \cdot \sigma' \cdot V_{kk'} + \sum_{k'=1}^H \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{(t+1)k'}} \cdot \sigma' \cdot W_{kk'}$$



Long-term dependencies

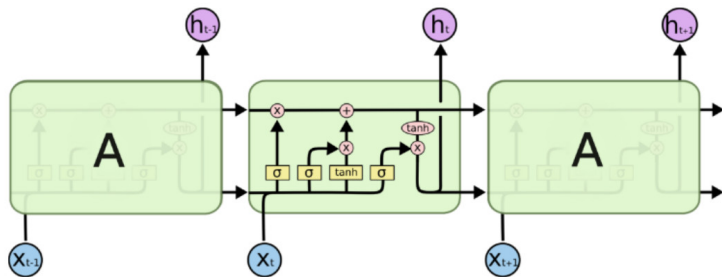
$$\frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{tk}} = \sum_{k'=1}^N \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial y_{tk'}} \cdot \sigma' \cdot V_{kk'} + \sum_{k'=1}^H \frac{\partial E(\mathbf{x}, \mathbf{d})}{\partial h_{(t+1)k'}} \cdot \sigma' \cdot W_{kk'}$$


- ▶ Unless $W_{kk'} \cdot \sigma' \approx 1$, the gradient either vanishes, or explodes.
- ▶ For large T (long-term dependency), the "deeper" gradient is too small (large).
- ▶ A solution: LSTM

$(\vec{y}_t =) \vec{h}_t = \vec{o}_t \circ \sigma_h(\vec{c}_t)$	output
$\vec{c}_t = \vec{f}_t \circ \vec{c}_{t-1} + \vec{i}_t \circ \tilde{C}_t$	memory
$\tilde{C}_t = \sigma_c(W_C \cdot \vec{h}_{t-1} + U_C \cdot x_t)$	new memory contents
$\vec{o}_t = \sigma_g(W_o \cdot \vec{h}_{t-1} + U_o \cdot \vec{x}_t)$	output gate
$\vec{f}_t = \sigma_g(W_f \cdot \vec{h}_{t-1} + U_f \cdot \vec{x}_t)$	forget gate
$\vec{i}_t = \sigma_g(W_i \cdot \vec{h}_{t-1} + U_i \cdot \vec{x}_t)$	input gate

- ▶ \circ is the component-wise product
- ▶ σ_h, σ_c original is hyperbolic tangents
(But in my opinion can be whatever you would put into the output and hidden layers, resp.)
- ▶ σ_g original is logistic sigmoid

LSTM




Neural Network
Layer

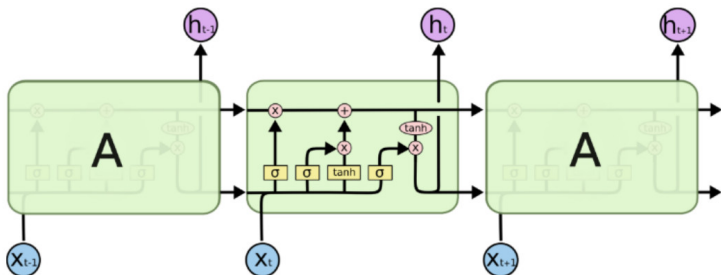
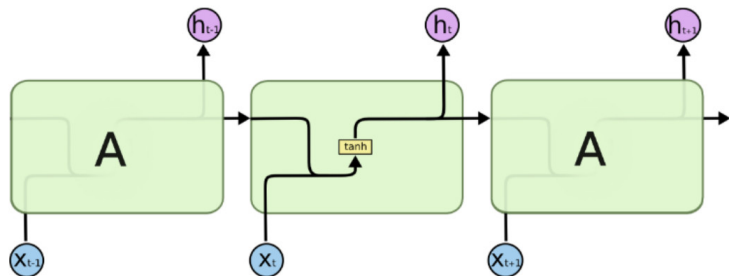

Pointwise
Operation


Vector
Transfer

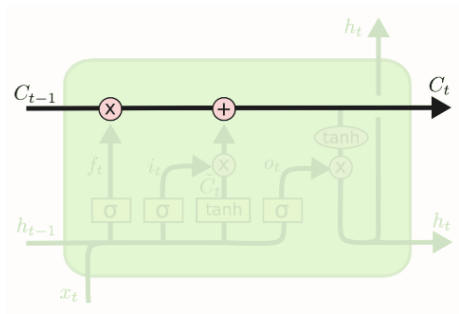

Concatenate

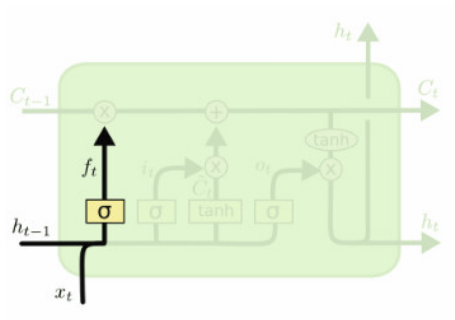

Copy

RNN vs LSTM



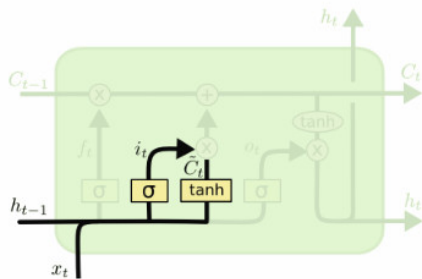
LSTM





$$\vec{f}_t = \sigma_g(W_f \cdot \vec{h}_{t-1} + U_f \cdot \vec{x}_t)$$

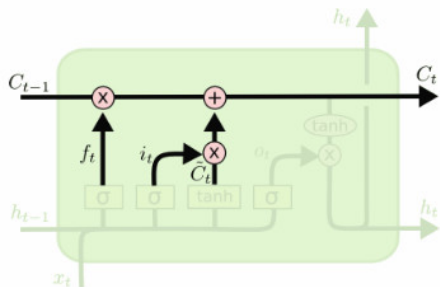
LSTM



$$\tilde{C}_t = \sigma_c(W_C \cdot \vec{h}_{t-1} + U_C \cdot x_t)$$

$$\vec{i}_t = \sigma_g(W_i \cdot \vec{h}_{t-1} + U_i \cdot \vec{x}_t)$$

LSTM

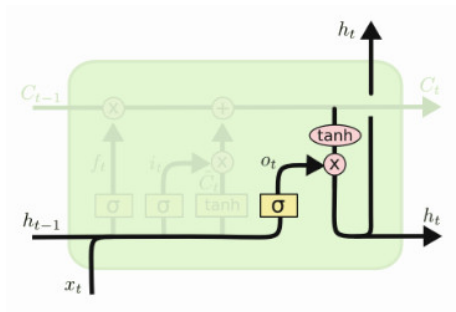


$$\vec{C}_t = \vec{f}_t \circ \vec{C}_{t-1} + \vec{i}_t \circ \tilde{C}_t$$

$$\vec{f}_t = \sigma_g(W_f \cdot \vec{h}_{t-1} + U_f \cdot \vec{x}_t)$$

$$\tilde{C}_t = \sigma_c(W_C \cdot \vec{h}_{t-1} + U_C \cdot x_t)$$

$$\vec{i}_t = \sigma_g(W_i \cdot \vec{h}_{t-1} + U_i \cdot \vec{x}_t)$$



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

$$\vec{o}_t = \sigma_g(W_o \cdot \vec{h}_{t-1} + U_o \cdot \vec{x}_t)$$

$$\vec{c}_t = \vec{f}_t \circ \vec{c}_{t-1} + \vec{i}_t \circ \tilde{c}_t$$

$$\vec{f}_t = \sigma_g(W_f \cdot \vec{h}_{t-1} + U_f \cdot \vec{x}_t)$$

$$\tilde{c}_t = \sigma_c(W_c \cdot \vec{h}_{t-1} + U_c \cdot x_t)$$

$$\vec{i}_t = \sigma_g(W_i \cdot \vec{h}_{t-1} + U_i \cdot \vec{x}_t)$$

Fun with LSTM – Shakespeare

- ▶ A LSTM generating **new** Shakespeare character by character!
- ▶ All works of Shakespeare concatenated into a single (4.4MB) file.
- ▶ 3-layer RNN with 512 hidden neurons in each layer.

VIOLA: Why, Salisbury must find his flesh and thought That which I am not apt, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

- ▶ Hutter Prize 100MB dataset of raw Wikipedia data (96MB for training, the rest for validation)

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[http:

//www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm

Official economic Adjunct for the Nations. Montromery was supports

Randomly hallucinated (correct!!) xml:

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">
      #REDIRECT [[Christianity]]</text>
    </revision>
  </page>
```

- ▶ Train RNN on an algebraic geometry book
<http://stacks.math.columbia.edu/>
- ▶ Raw LaTeX source file (a 16MB file) and trained a multilayer LSTM.
- ▶ The resulting sampled Latex almost compiles!
- ▶ The authors had to step in and fix a few issues manually but then you get plausible looking math.

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 $\mathcal{F} \rightarrow \mathcal{F}$ 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

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

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

- ▶ Trained on all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474MB of C code).
- ▶ 3-layer LSTM with approx. 10 million parameters.

```

/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac) | PFMR_CLOBATHINC_SECONDS << 12];
    return segtable;
}

```

```
/*
 * 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>
```

Evolution of Shakespeare

100 iter.:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

300 iter.:

"Tmont thithey" fomesscerliund

Keushey. Thom here

sheulke, anmerenith ol sivh I lalterthend Bleipile shuwv fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

500 iter.:

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

700 iter.:

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 offer.

1200 iter.:

"Kite vouch!" he repeated by her

door. "But I would be done and quarts, feeling, then, son is people..."

2000 iter.:

"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.