

Topics:

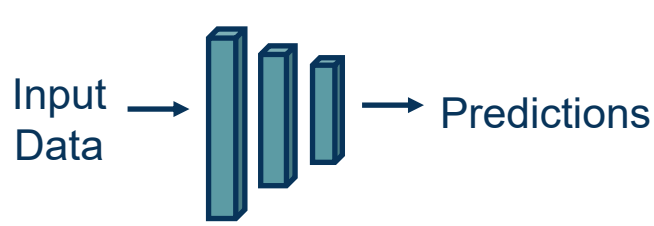
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory

CS 4644-DL / 7643-A
ZSOLT KIRA

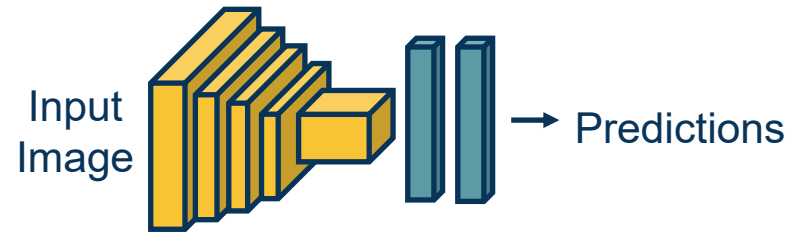
- **Assignment 2 – Due June. 22nd**
 - Implement convolutional neural networks
 - Resources (in addition to lectures):
 - [DL book: Convolutional Networks](#)
 - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
 - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
 - **HW2 Tutorial (@176)**
 - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6)
(https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvlzX0nPa?dl=0)
- **Meta Office Hours – First one on language modeling today 3pm ET**
- **GPU resources:** Google Colab, PACE-ICE
 - Google Cloud posted soon (for projects)

Module 3

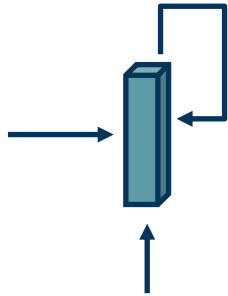
Introduction



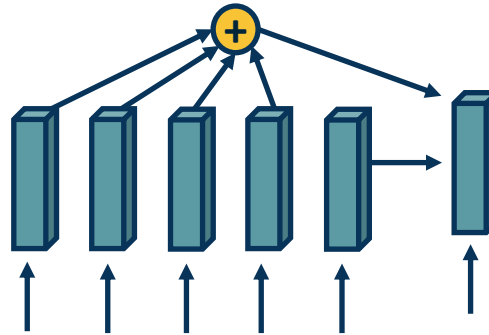
**Fully Connected
Neural Networks**



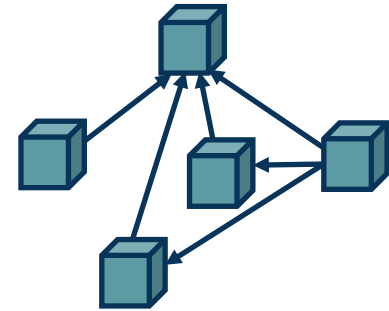
**Convolutional Neural
Networks**



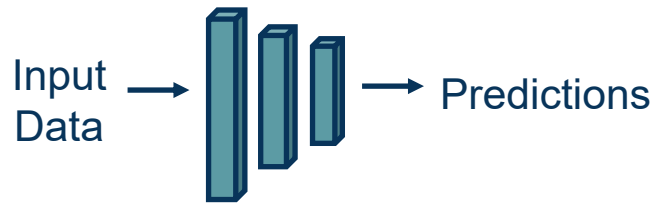
**Recurrent Neural
Networks**



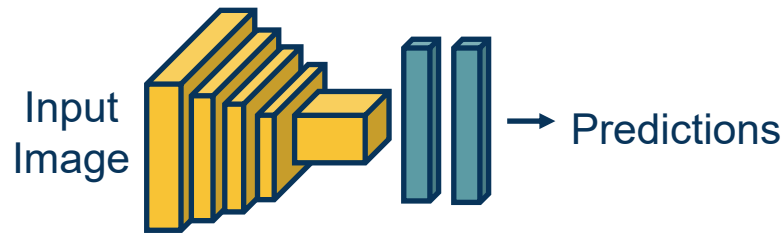
**Attention-Based
Networks**



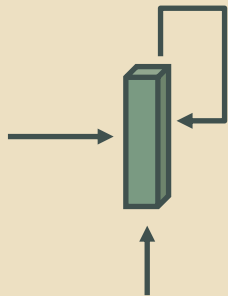
**Graph-Based
Networks**



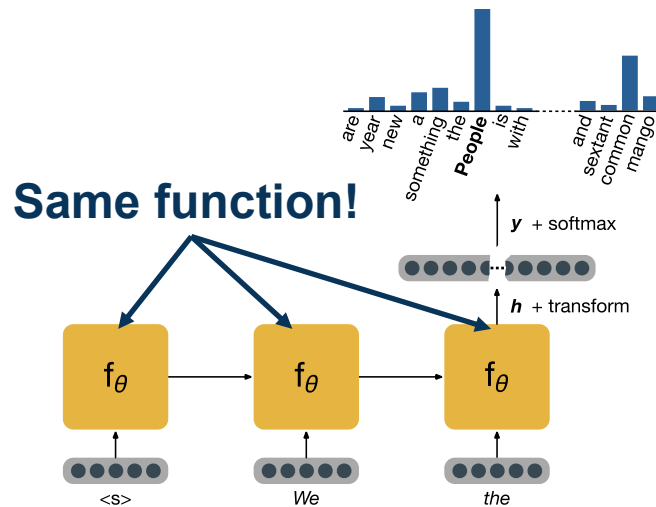
**Fully Connected
Neural Networks**



**Convolutional Neural
Networks**



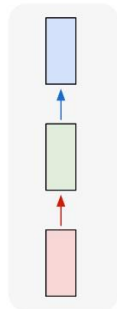
**Recurrent Neural
Networks**



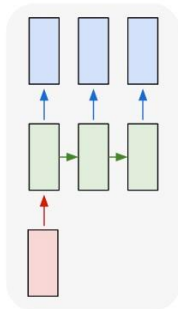
Recurrent Neural Networks & Transformers

New Topic: RNNs

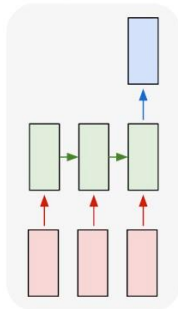
one to one



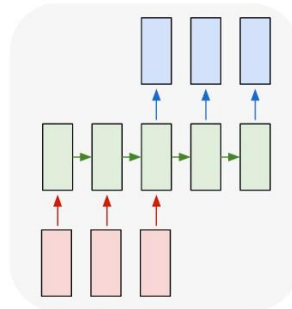
one to many



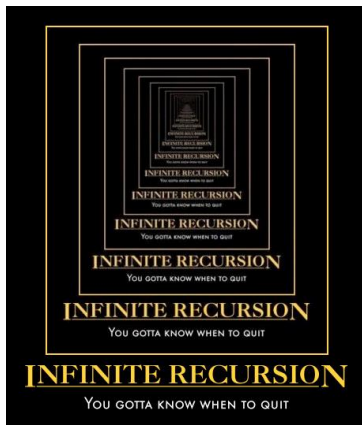
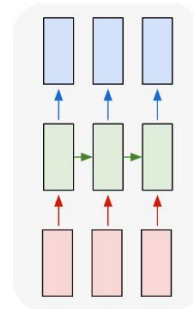
many to one



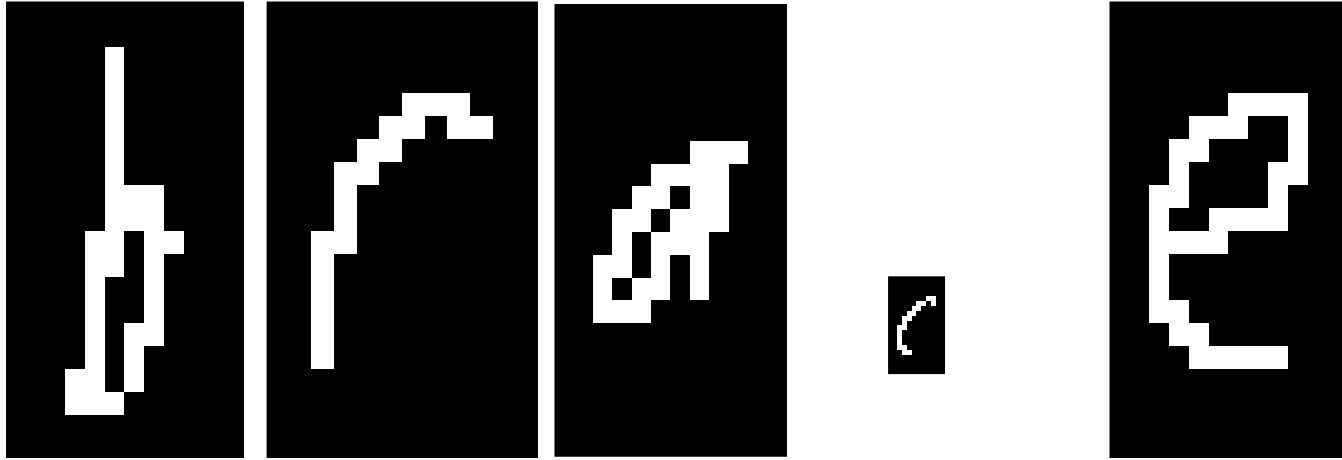
many to many



many to many



Why model sequences?



Sequences are everywhere...

Foreign Minister. → FOREIGN MINISTER.

 → THE SOUND OF

$a_1=2$ $a_2=0$ $a_3=1$ $a_4=3$ $a_5=4$ $a_6=2$ $a_7=5$
 $x =$ bringen sie bitte das auto zurück .
 $y =$ please return the car .

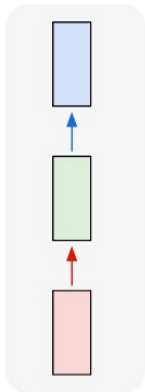
Diagram illustrating a sequence alignment between two sentences, x and y , using indices a_1 through a_7 . The alignment is shown by lines connecting the words in x to the words in y :

- $a_1=2$ connects "bringen" to "please"
- $a_2=0$ connects "sie" to "return"
- $a_3=1$ connects "bitte" to "the"
- $a_4=3$ connects "das" to "car"
- $a_5=4$ connects "auto" to "car"
- $a_6=2$ connects "zurück" to "return"
- $a_7=5$ connects "." to "."

Sequences in Input or Output?

- It's a spectrum...

one to one



Input: No sequence

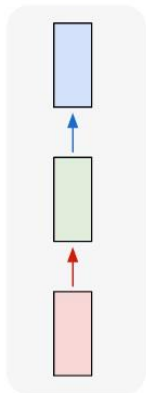
Output: No sequence

Example: "standard"
classification /
regression problems

Sequences in Input or Output?

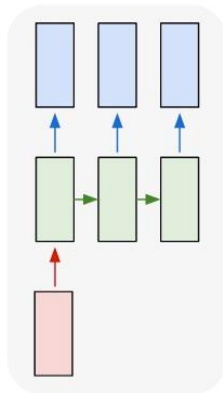
- It's a spectrum...

one to one



Input: No sequence
Output: No sequence
Example: "standard"
classification /
regression problems

one to many

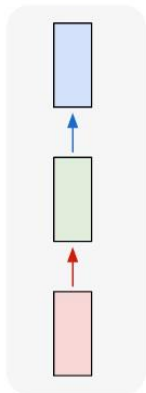


Input: No sequence
Output: Sequence
Example: Im2Caption

Sequences in Input or Output?

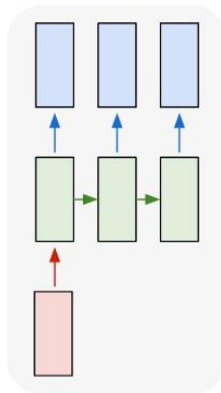
- It's a spectrum...

one to one



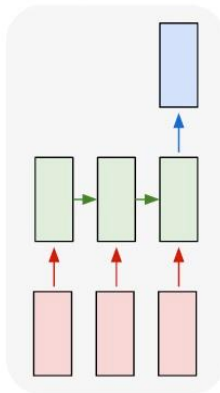
Input: No sequence
Output: No sequence
Example: "standard"
classification /
regression problems

one to many



Input: No sequence
Output: Sequence
Example: Im2Caption

many to one

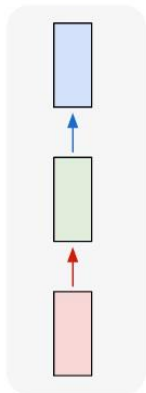


Input: Sequence
Output: No sequence
Example: sentence classification,
multiple-choice question answering

Sequences in Input or Output?

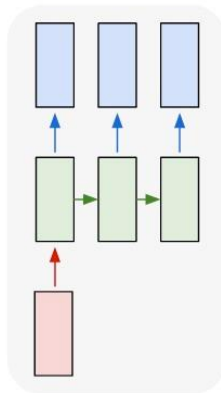
- It's a spectrum...

one to one



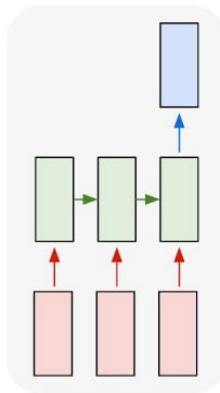
Input: No sequence
Output: No sequence
Example: "standard" classification / regression problems

one to many



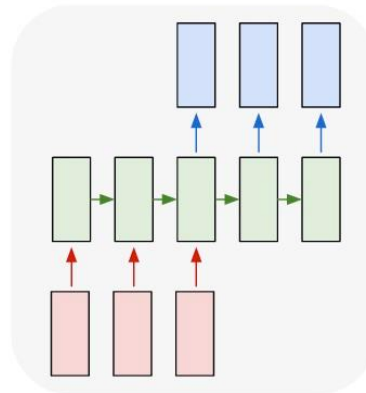
Input: No sequence
Output: Sequence
Example: Im2Caption

many to one



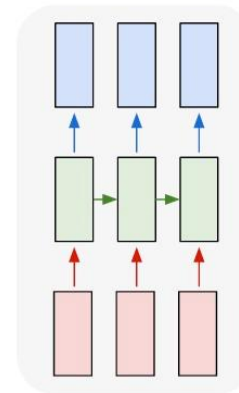
Input: Sequence
Output: No sequence
Example: sentence classification, multiple-choice question answering

many to many



Input: Sequence
Output: Sequence
Example: machine translation, video classification, video captioning, open-ended question answering

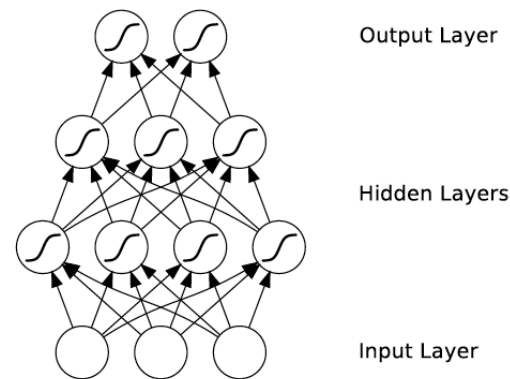
many to many



Input: Sequence
Output: Sequence

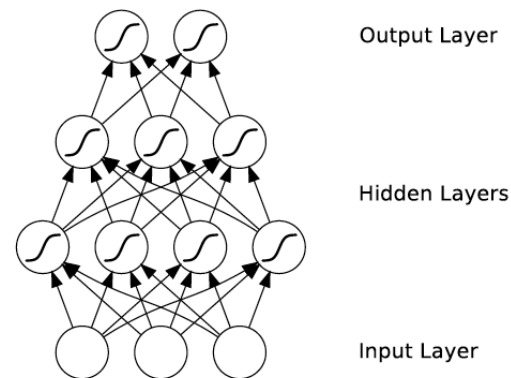
What's wrong with MLPs?

- Problem 1: Can't model sequences
 - Fixed-sized Inputs & Outputs
 - No temporal structure



What's wrong with MLPs?

- Problem 1: Can't model sequences
 - Fixed-sized Inputs & Outputs
 - No temporal structure
- Problem 2: Pure feed-forward processing
 - No “memory”, no feedback



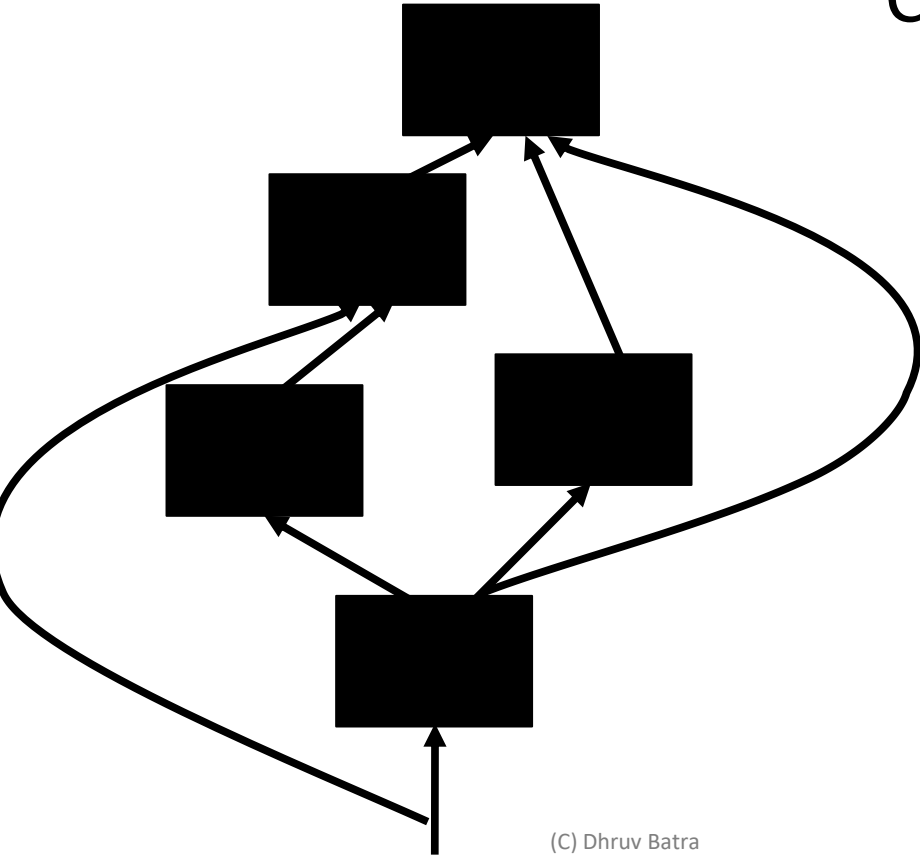
3 Key Ideas

- The notion of memory (state)
 - We want to propagate information across the sequence
 - We will do this with *state*, represented by a vector (embedding/representation)
 - Key idea will be mixing new inputs with this state, to yield a new state
 - All represented as vector operations
 - Just as a CNN represents an image with the final hidden vector/embedding before the final classifier

3 Key Ideas

- The notion of memory (state)
- Parameter Sharing
 - in computation graphs = adding gradients

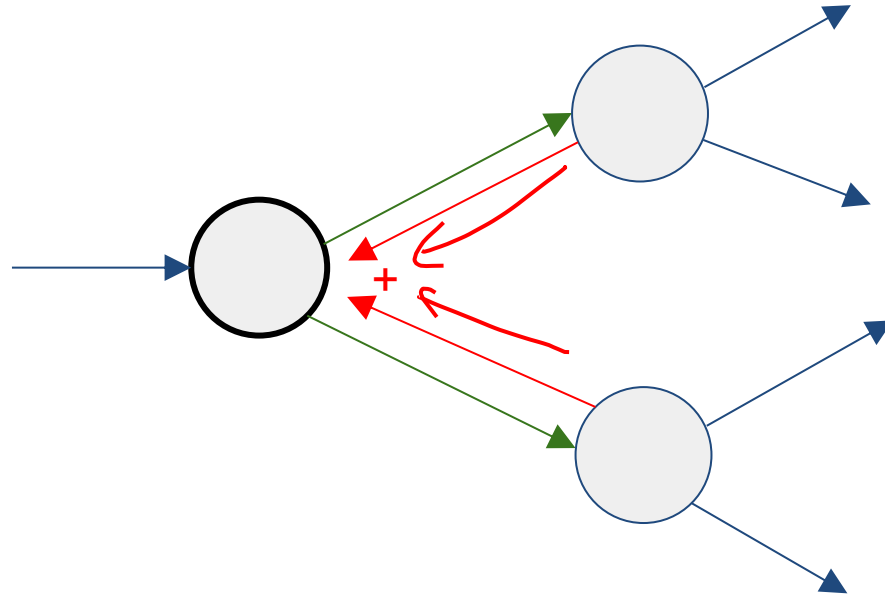
Computational Graph



(C) Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato

Gradients add at branches



3 Key Ideas

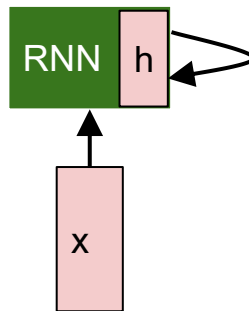
- The notion of memory (state)
- Parameter Sharing
 - in computation graphs = adding gradients
- “Unrolling”
 - in computation graphs with parameter sharing

New Words

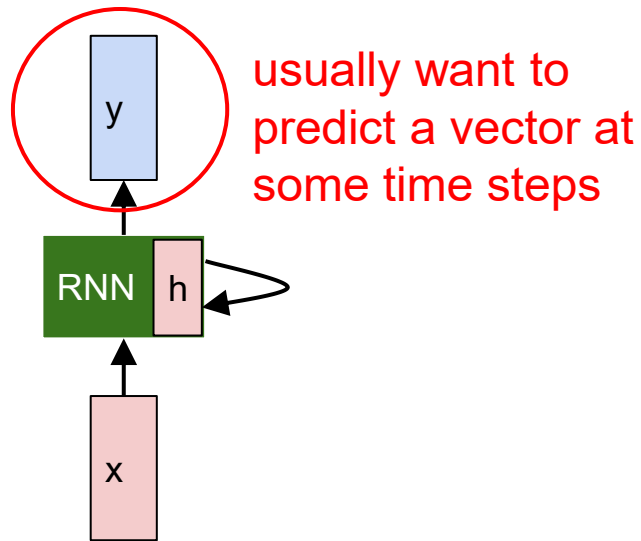
- Recurrent Neural Networks (RNNs)
- Recursive Neural Networks
 - General family; think graphs instead of chains
- Types:
 - “Vanilla” RNNs (Elman Networks)
 - Long Short Term Memory (LSTMs)
 - Gated Recurrent Units (GRUs)
 - ...
- Algorithms
 - BackProp Through Time (BPTT)
 - BackProp Through Structure (BPTS)

Recurrent Neural Network

- Idea: Input is a **sequence** and we will process it sequentially through a neural network module with *state*
- For each timestep (element of sequence):

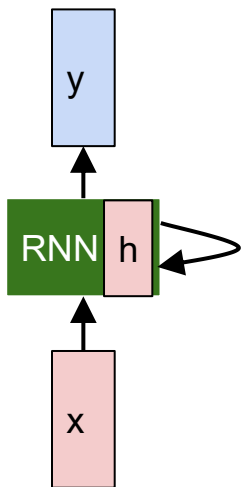


Recurrent Neural Network



(Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector \mathbf{h} :



$$y_t = W_{hy}h_t + b_y$$

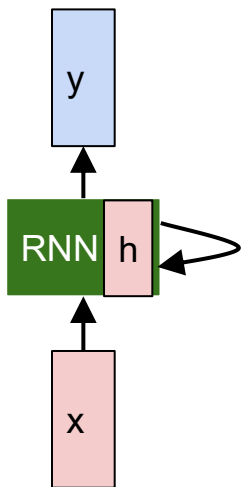
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector \mathbf{h} :



$$y_t = W_{hy}h_t + b_y$$

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

Recurrent Neural Network

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

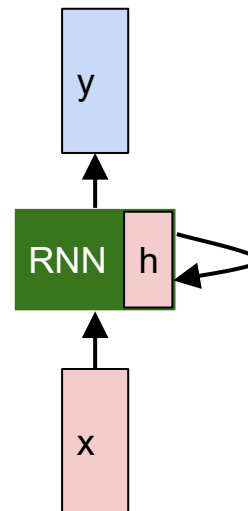
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step

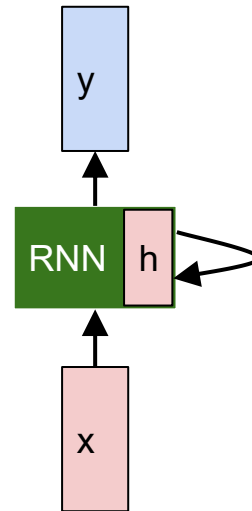


Recurrent Neural Network

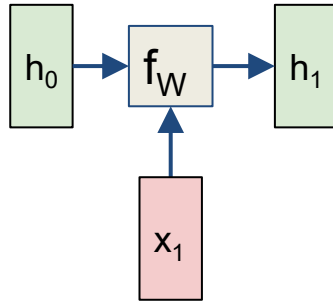
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

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

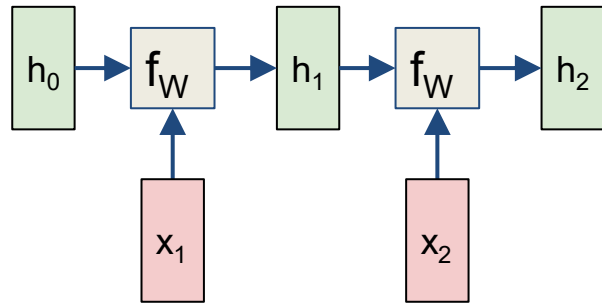
Notice: the same function and the same set of parameters are used at every time step.



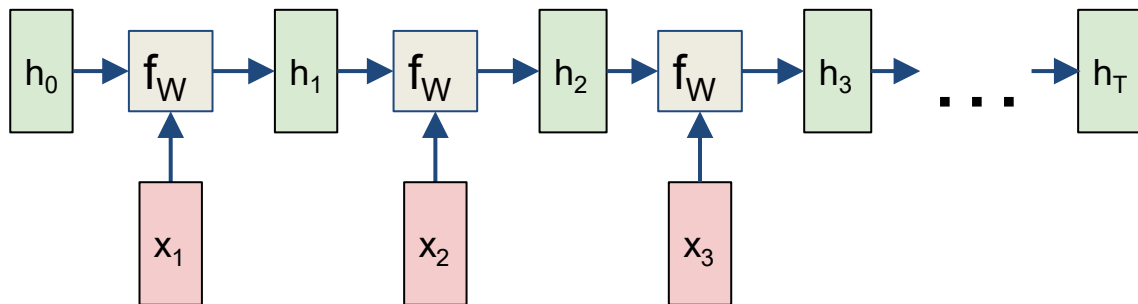
RNN: Computational Graph



RNN: Computational Graph

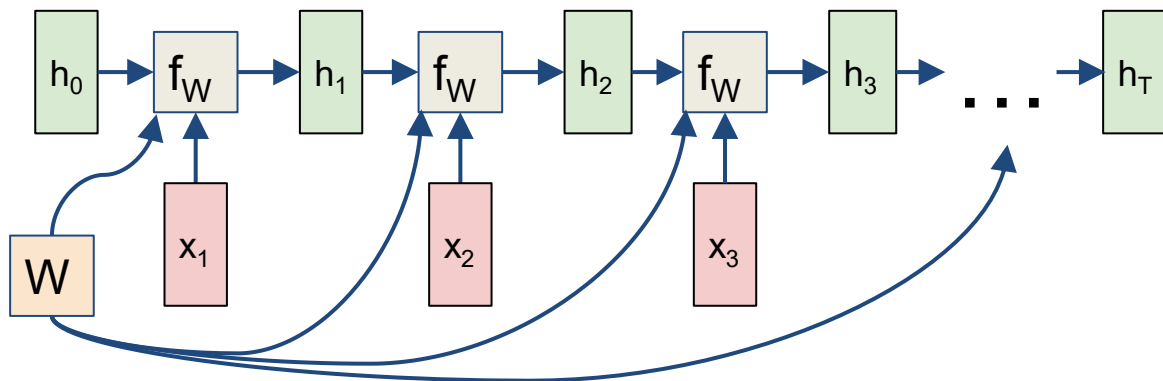


RNN: Computational Graph

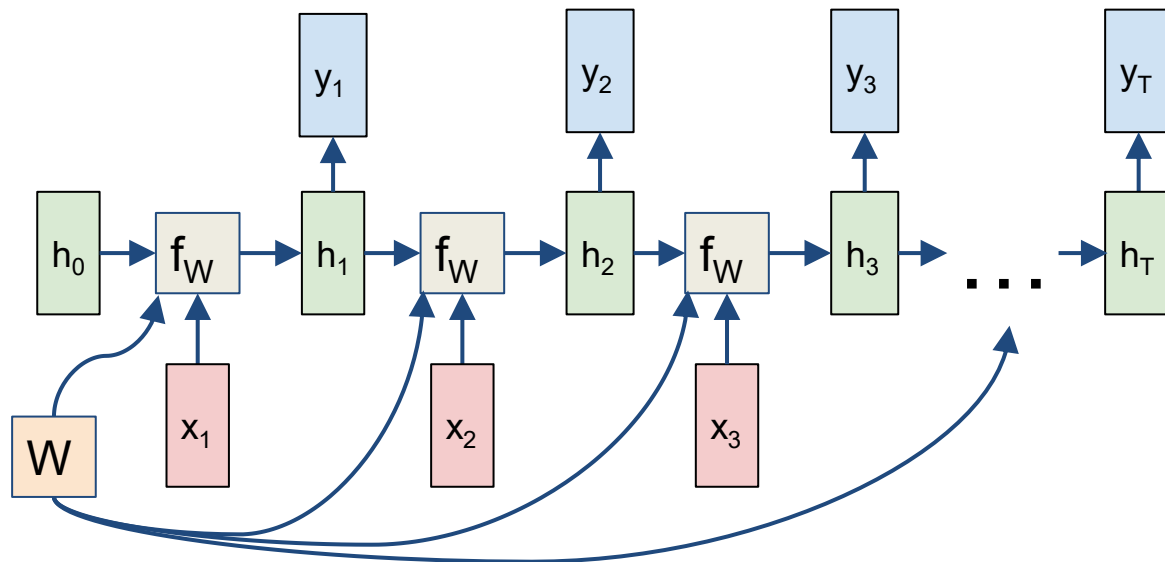


RNN: Computational Graph

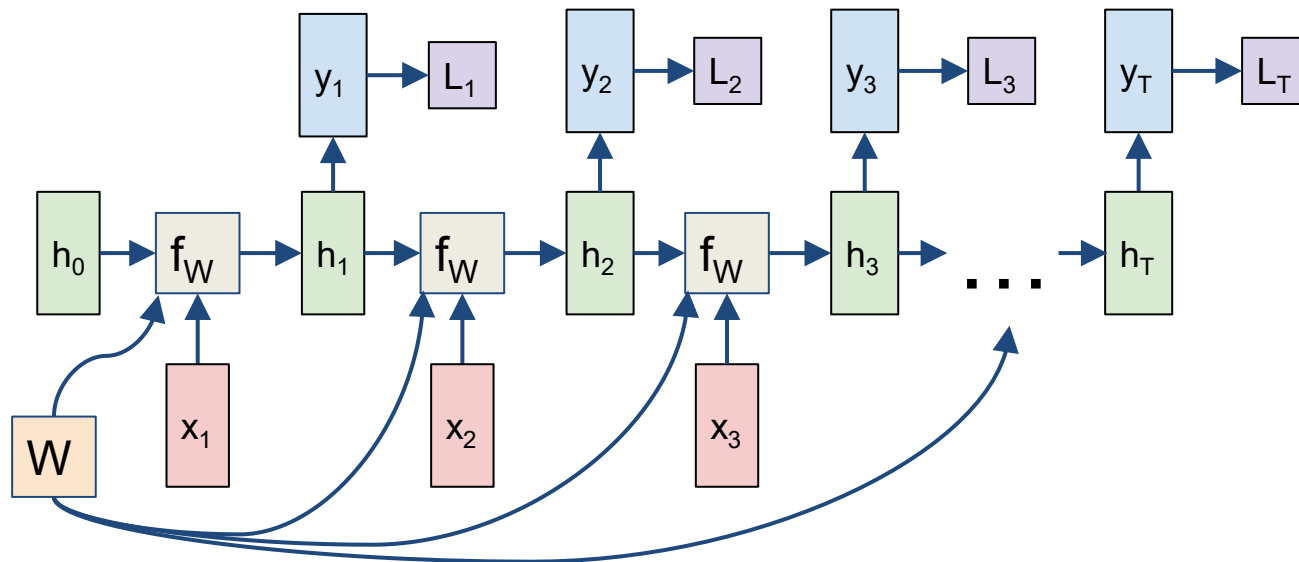
Re-use the same weight matrix at every time-step



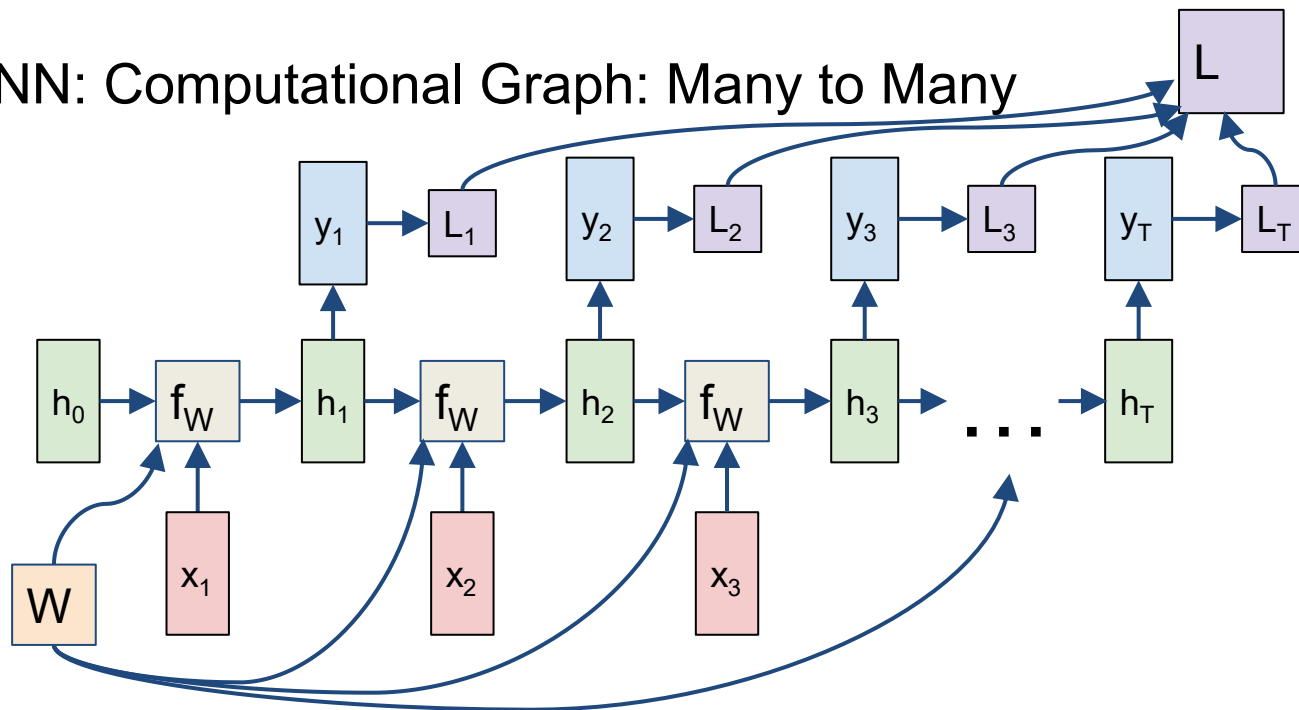
RNN: Computational Graph: Many to Many



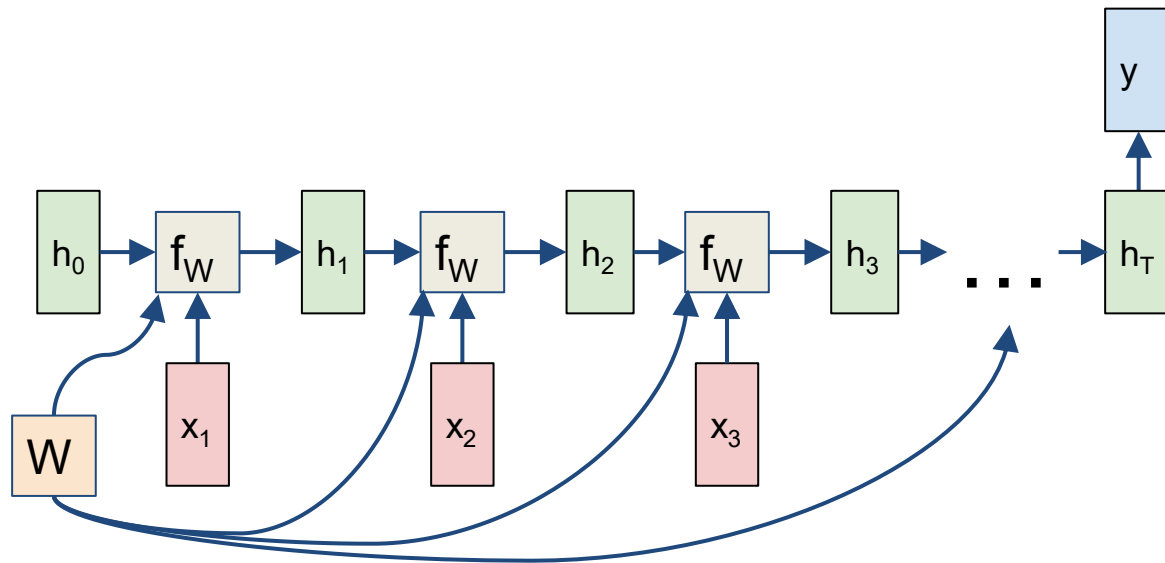
RNN: Computational Graph: Many to Many



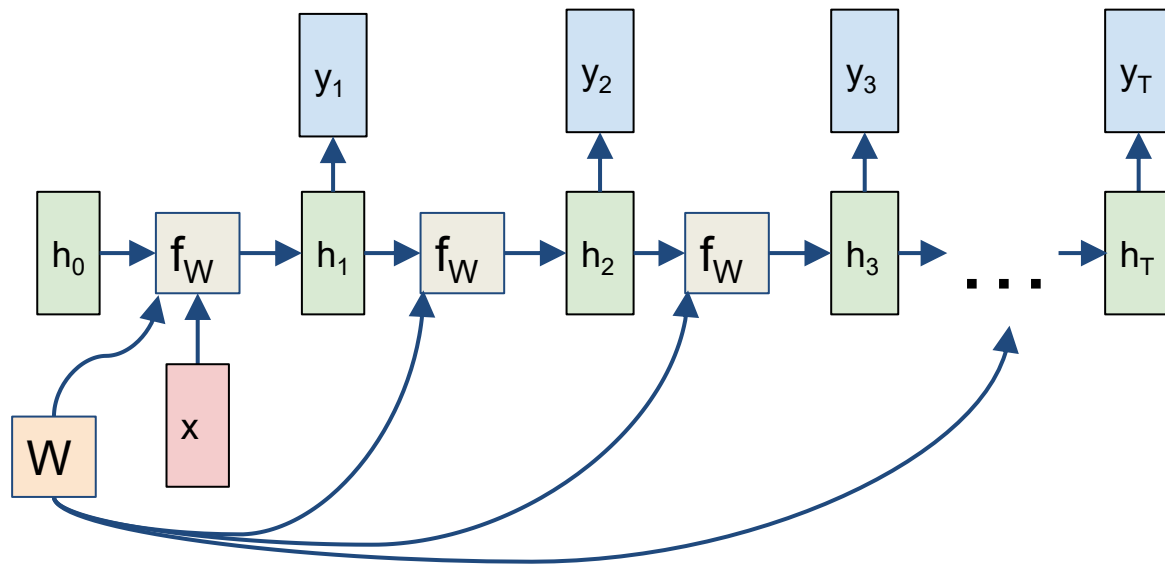
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

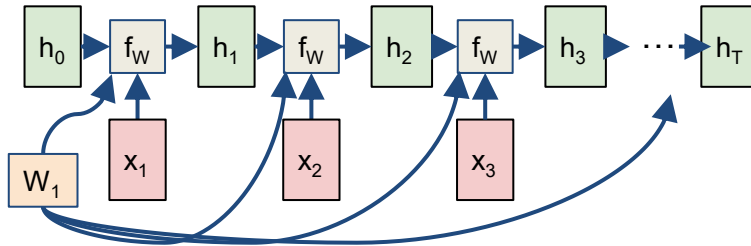


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

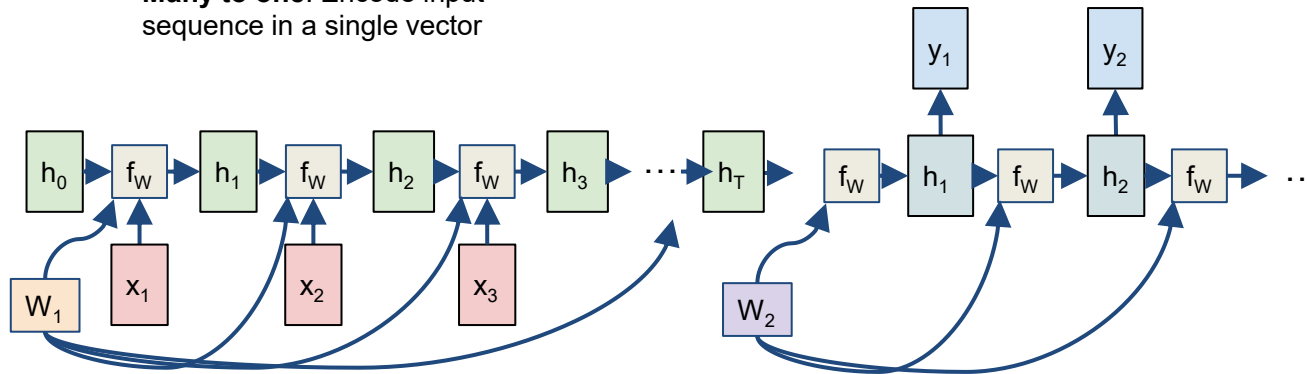
Many to one: Encode input sequence in a single vector



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



How can we train this on language?

- We have a generic sequence-in, sequence-out model
- How do we train this on language?
- Supervised Learning:
 - Sentiment analysis (sentence -> negative/neutral/positive) labeled by humans
 - Translation -> English and equivalent other language
- Self-supervised: Predict the next letter or word!
 - This is **extremely powerful!!**
 - In order to predict what's next, it needs to really understand not just language statistics but world knowledge!
 - Of course, we need scale for this level of loss reduction / understanding

- **Training:** A large corpus of text from the web
 - Note: No annotation required! It's just "the text"
- **Inference:** Just generate me new text
 - Can condition on some initial input (**prompt**)

```
#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)

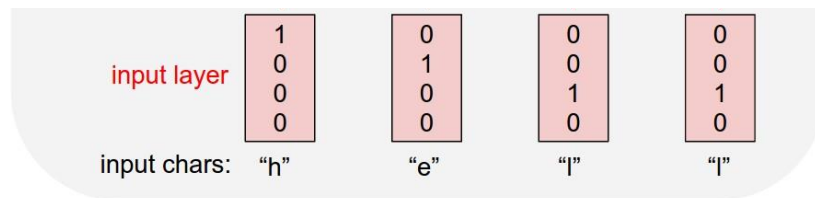
static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

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;
}
}
```

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

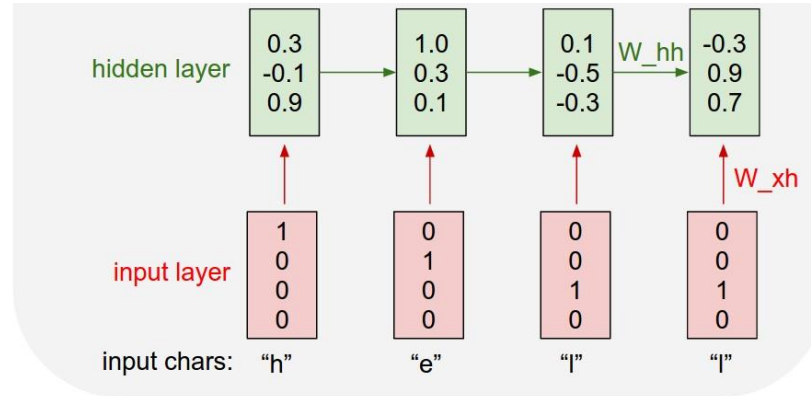


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

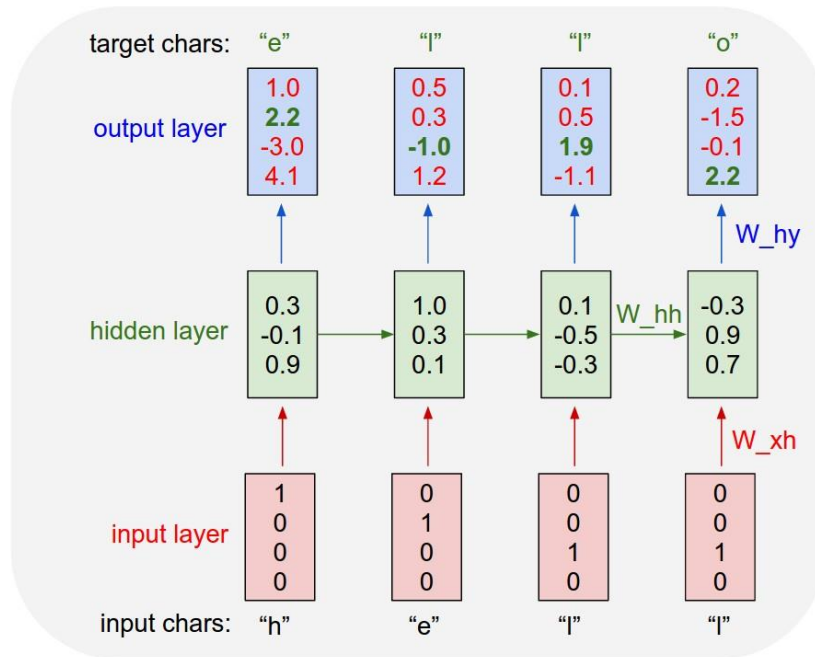
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

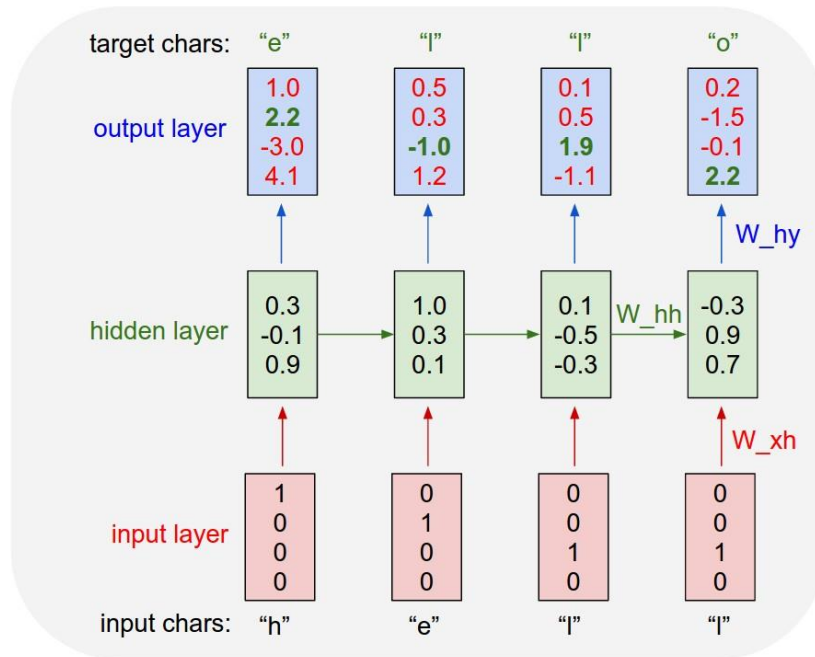


Training Time: MLE / “Teacher Forcing”

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

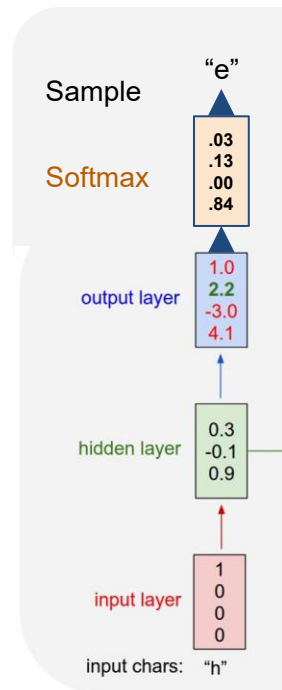


Test Time: Sample / Argmax

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a
time, feed back to
model

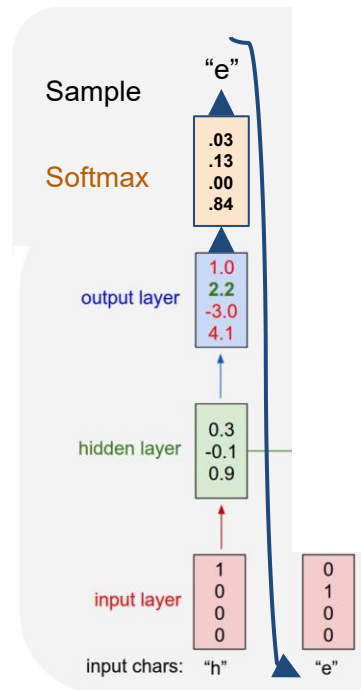


Test Time: Sample / Argmax

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a
time, feed back to
model

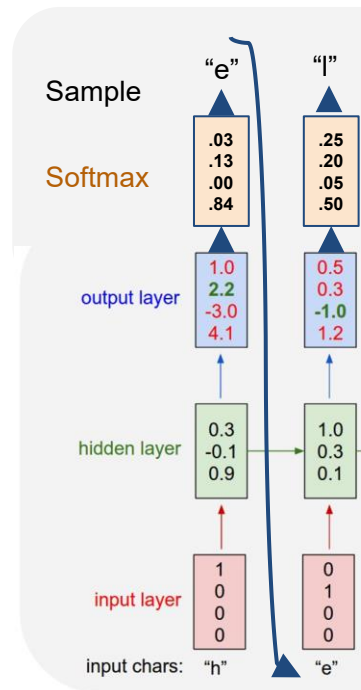


Test Time: Sample / Argmax

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a
time, feed back to
model

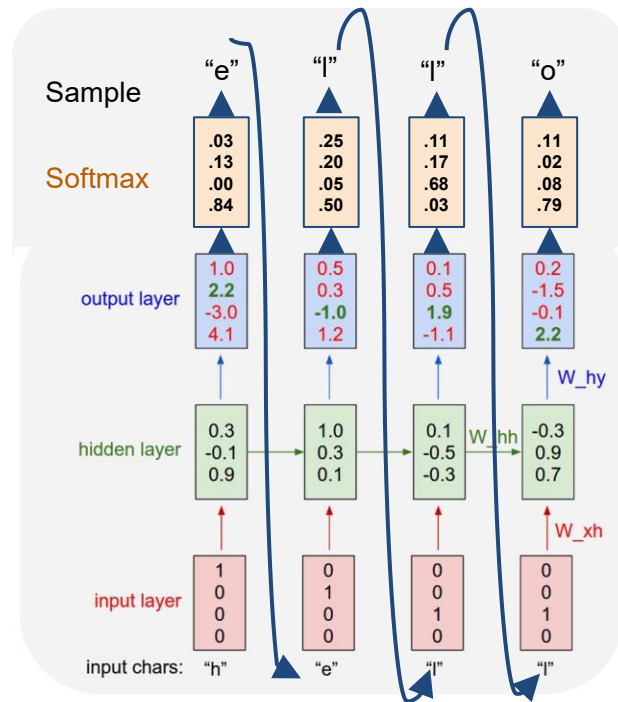


Test Time: Sample / Argmax

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a
time, feed back to
model

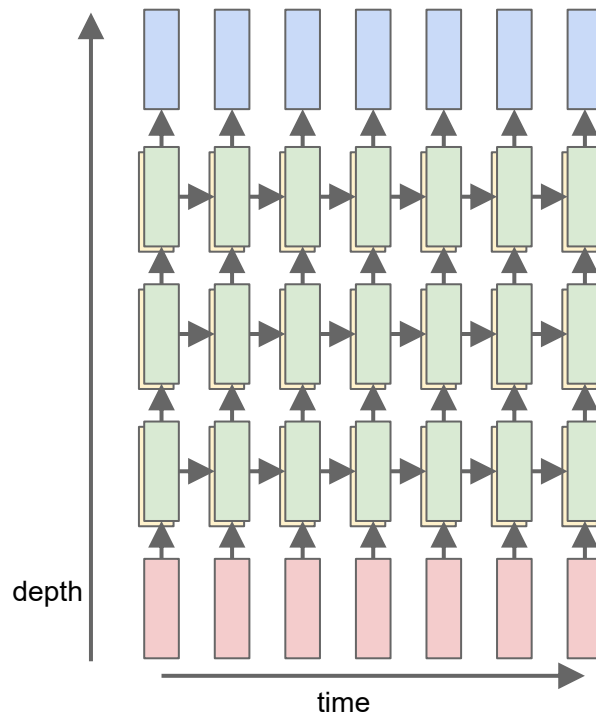


Can also feed in predictions during training (student forcing)

Multilayer RNNs

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$. $W^l [n \times 2n]$



- **Training:** A large corpus of text from the web
 - Note: No annotation required! It's just "the text"
- **Inference:** Just generate me new text
 - Can condition on some initial input (**prompt**)

```
#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>

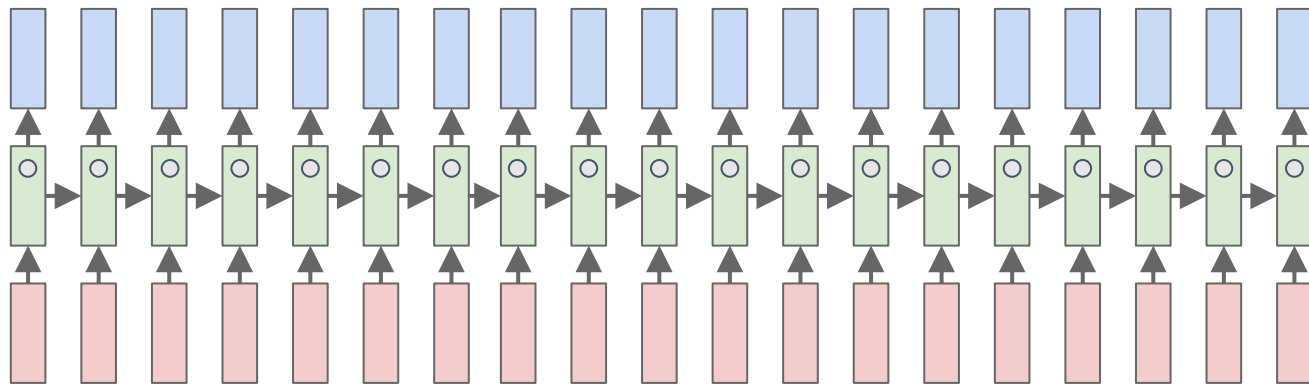
#define REG_PG      vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

#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;
}
}
```

Searching for interpretable cells



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

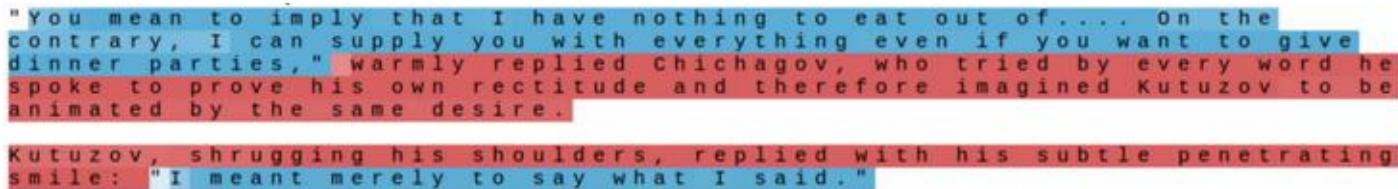
Searching for interpretable cells

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
}
```

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
                           siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

Cell that turns on inside comments and quotes:

```
/* duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                     struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                  (void **)&df->lsm_rule);
    /* keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM '%s' is invalid\n",
                df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

quote/comment cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

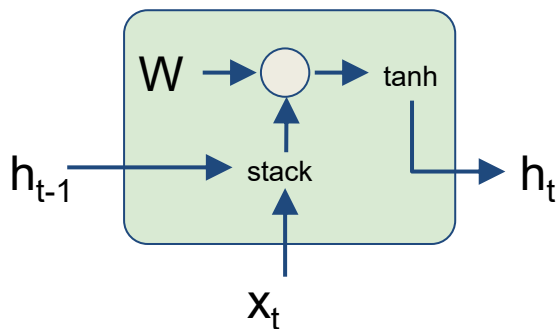
code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Vanilla RNN Gradient Flow

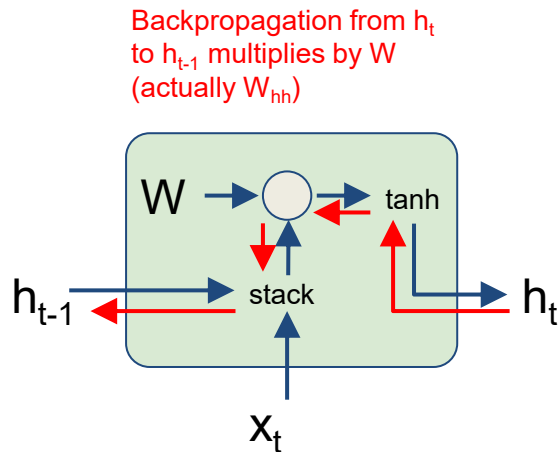
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

Vanilla RNN Gradient Flow

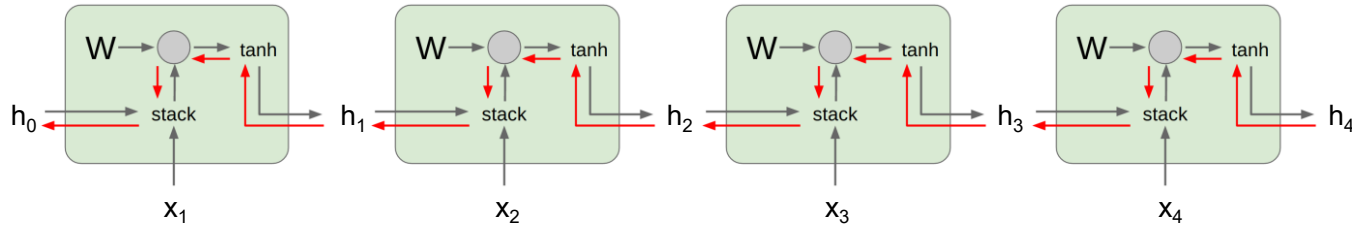
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

Vanilla RNN Gradient Flow

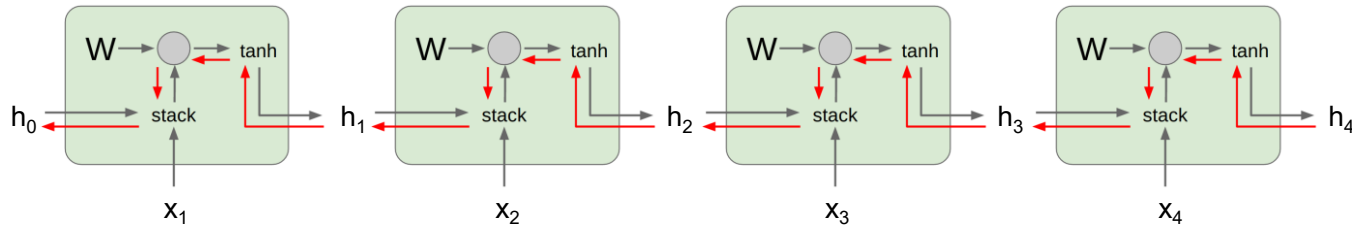
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient
of h_0 involves many
factors of W
(and repeated \tanh)

Vanilla RNN Gradient Flow

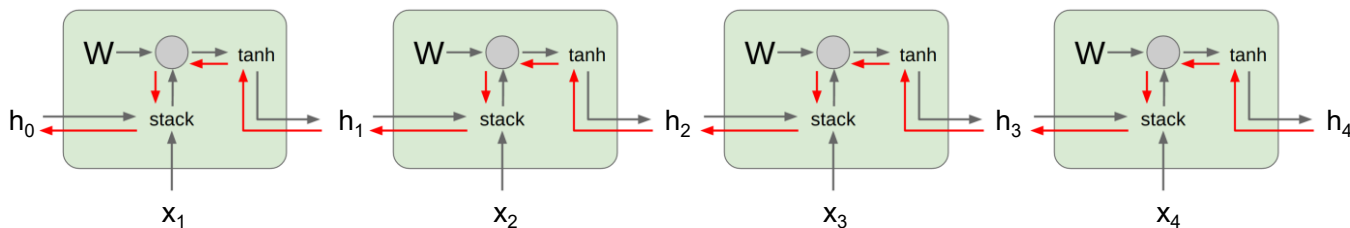
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

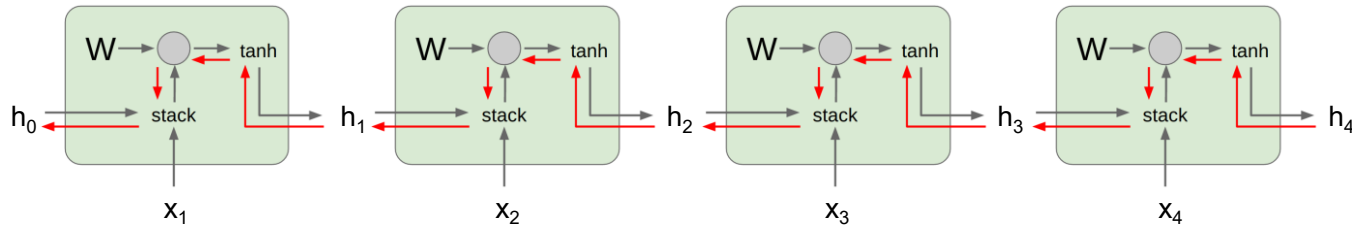


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

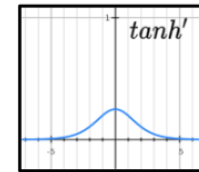
Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
 Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

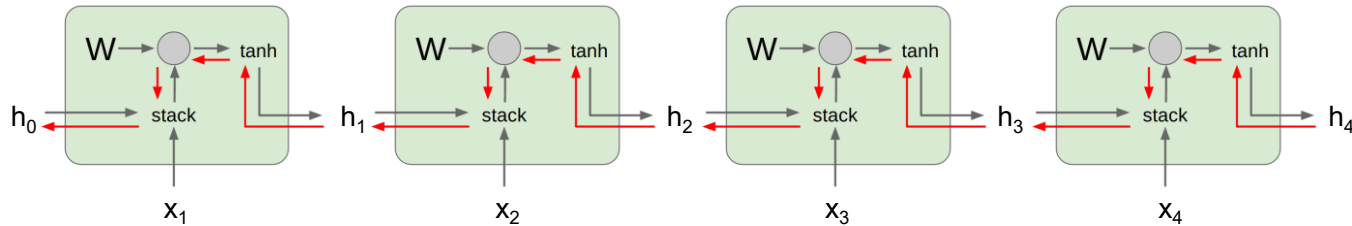
Always < 1
Vanishing gradients



$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient
of h_0 involves many
factors of W
(and repeated tanh)

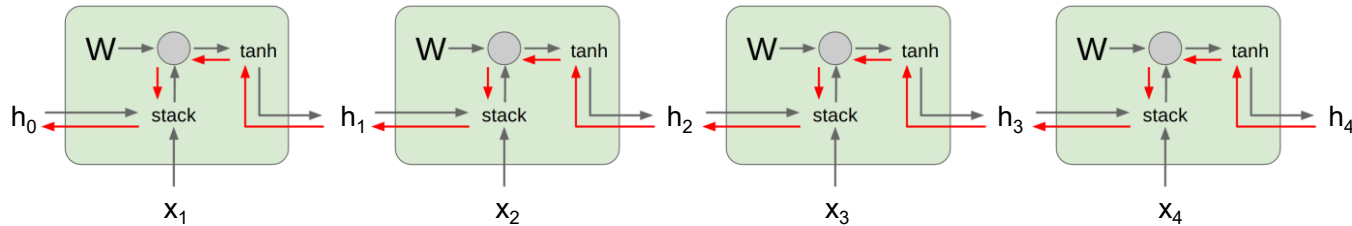
With no non-linearity:

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

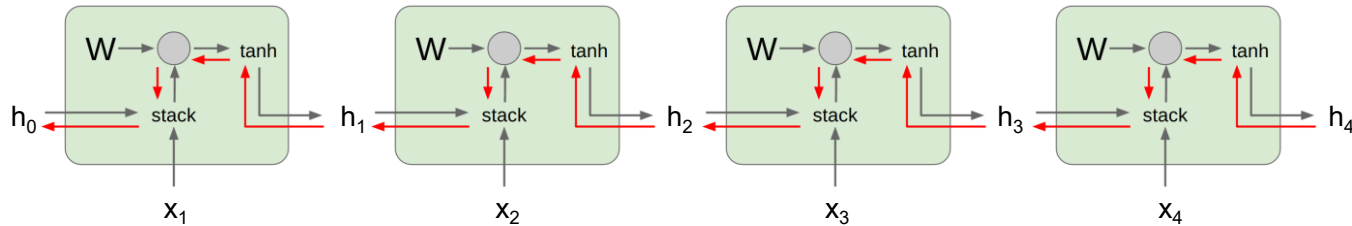
Largest singular value < 1 :
Vanishing gradients

➔ **Gradient clipping:** Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```


Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated \tanh)

Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

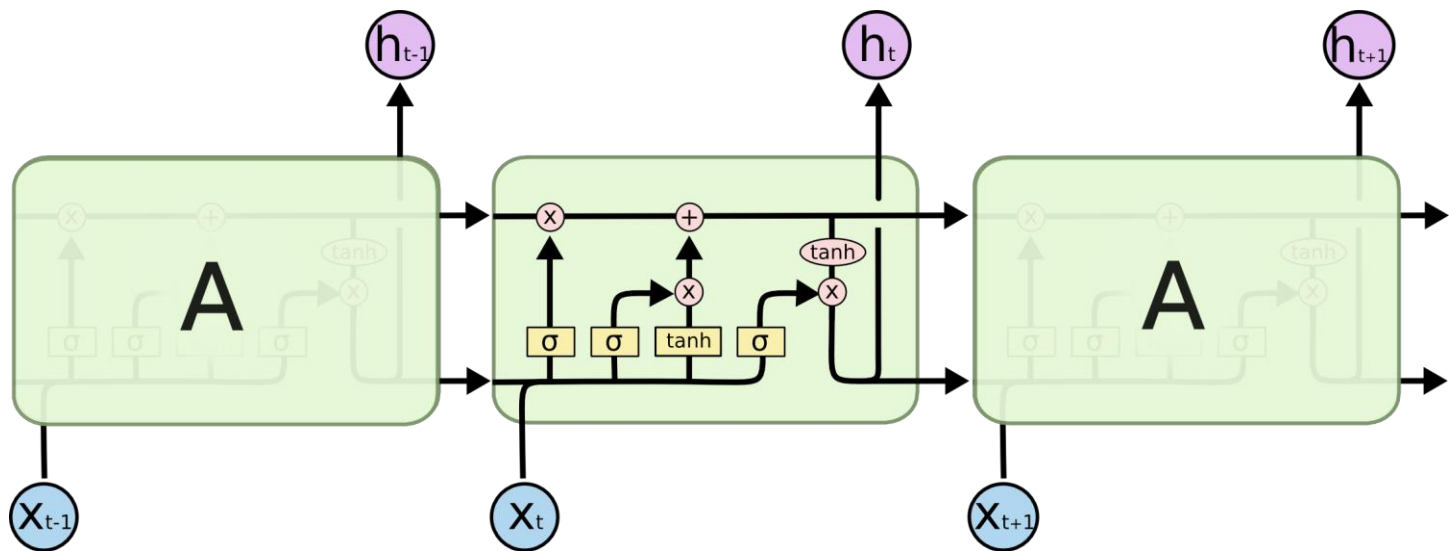
$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

LSTM

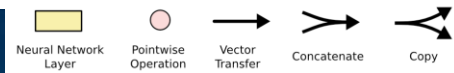
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation
1997

Meet LSTMs



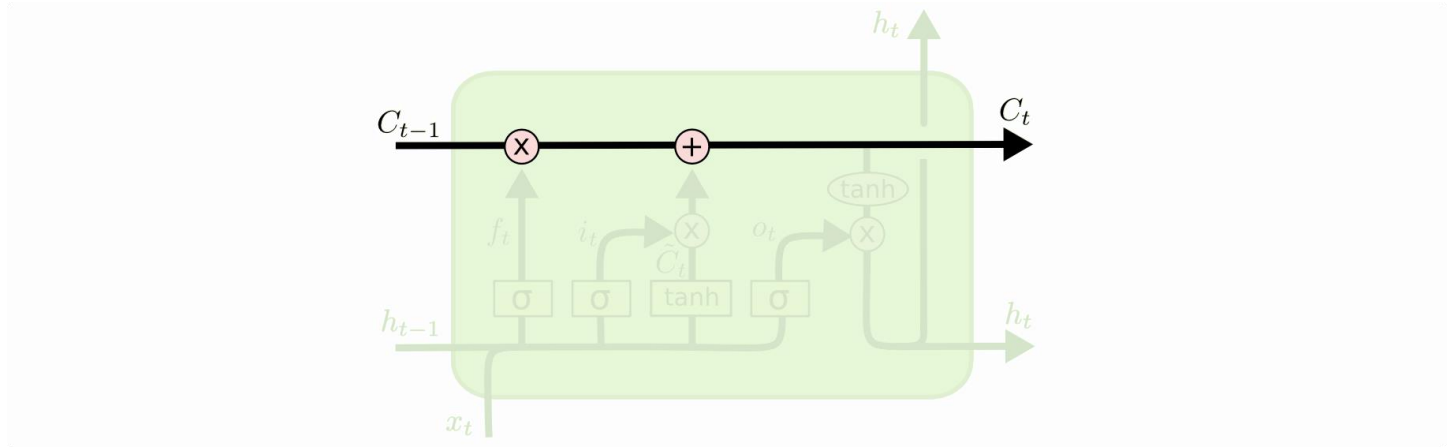
Intuition: “Gating” mechanism similar to residual, but more complex memory operations (forget/read, write)



(C) Dhruv Batra

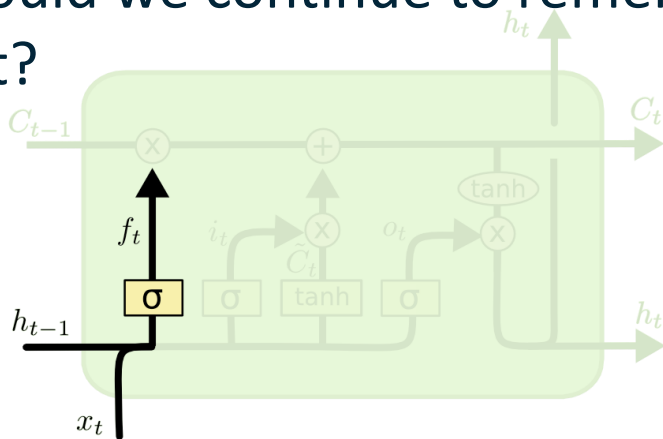
LSTMs Intuition: Memory

- Cell State / Memory



LSTMs Intuition: Forget Gate

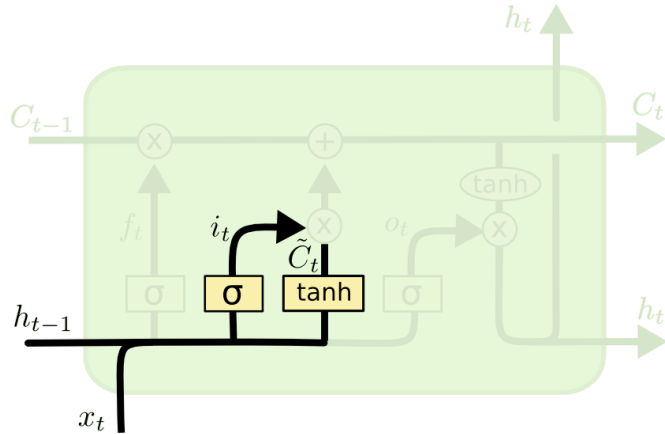
- Should we continue to remember this “bit” of information or not?



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

LSTMs Intuition: Input Gate

- Should we update this “bit” of information or not?
 - If so, with what?

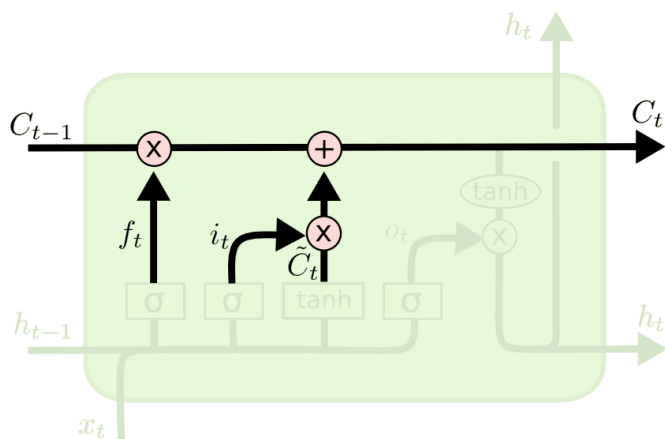


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

LSTMs Intuition: Memory Update

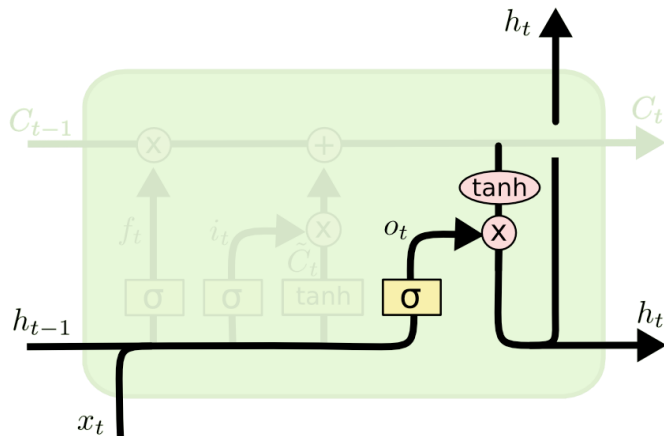
- Forget that + memorize this



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

LSTMs Intuition: Output Gate

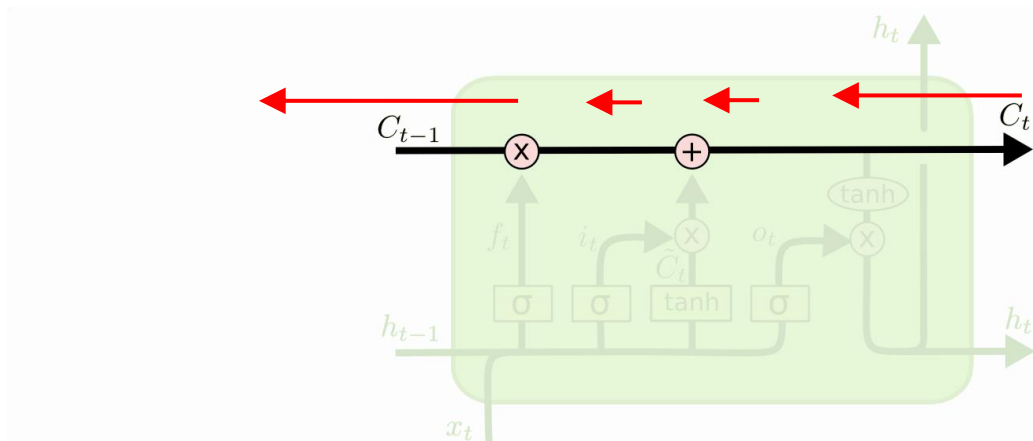
- Should we output this “bit” of information to “deeper” layers?



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

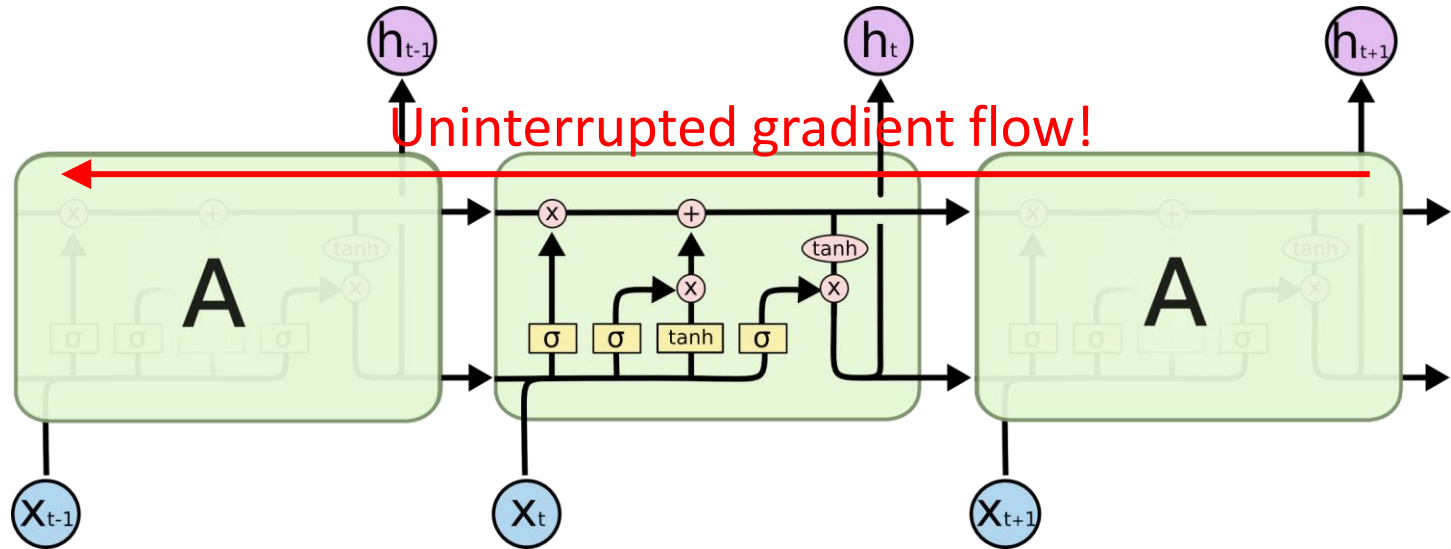
$$h_t = o_t * \tanh (C_t)$$

LSTMs Intuition: Additive Updates

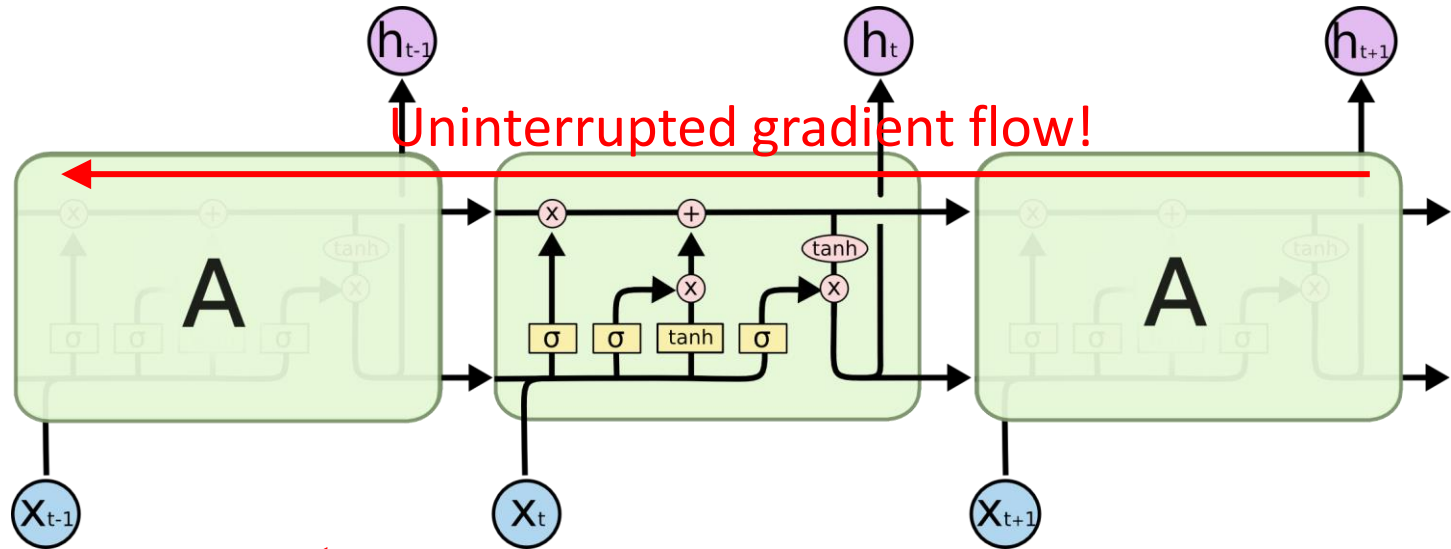


Backpropagation from c_t to c_{t-1} only elementwise multiplication by f , no matrix multiply by W

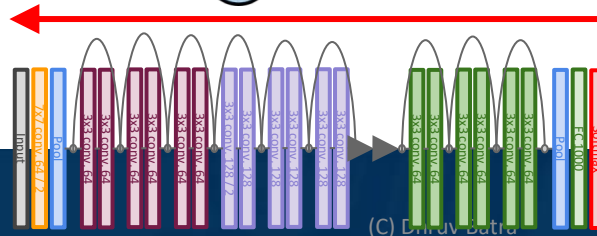
LSTMs Intuition: Additive Updates



LSTMs Intuition: Additive Updates

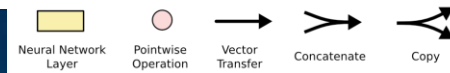
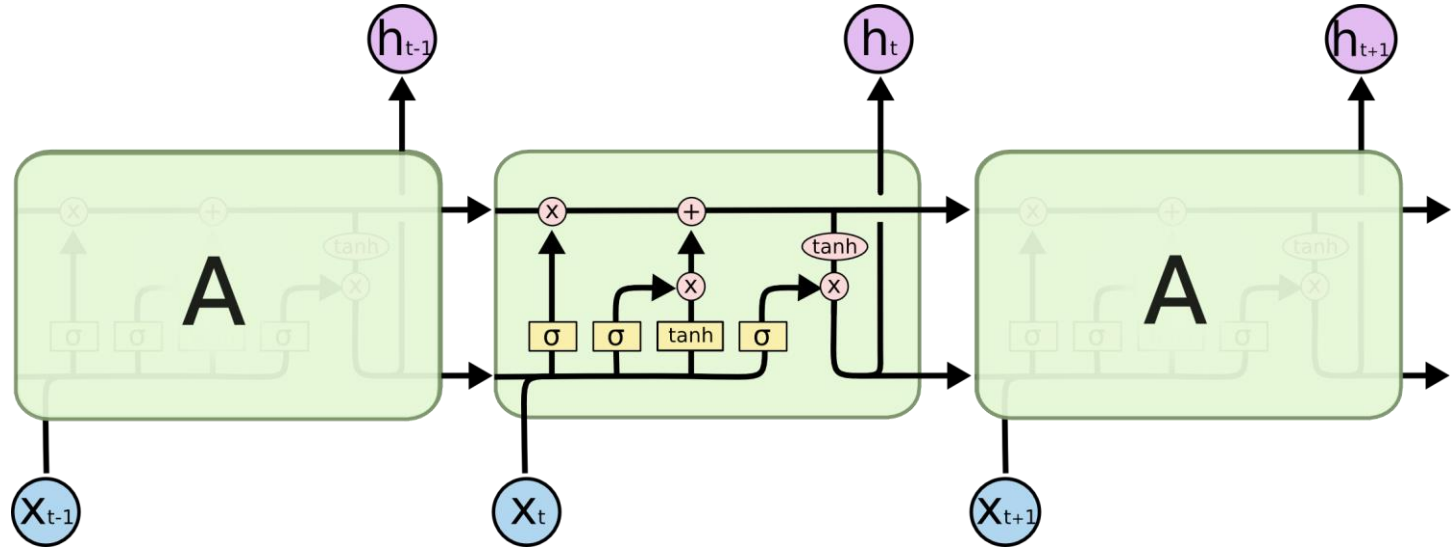


Similar to
ResNet!



LSTMs

- A pretty sophisticated cell



(C) Dhruv Batra

Other RNN Variants

[An Empirical Exploration of
Recurrent Network Architectures,
Jozefowicz et al., 2015]

MUT1:

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

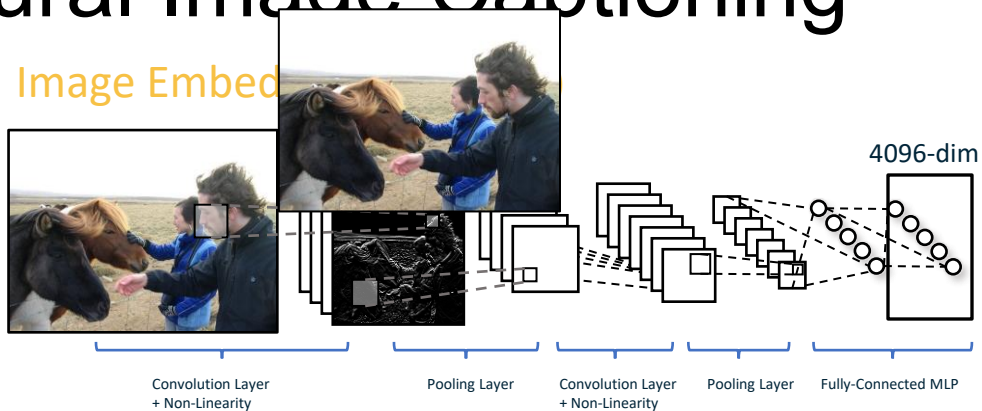
MUT2:

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

MUT3:

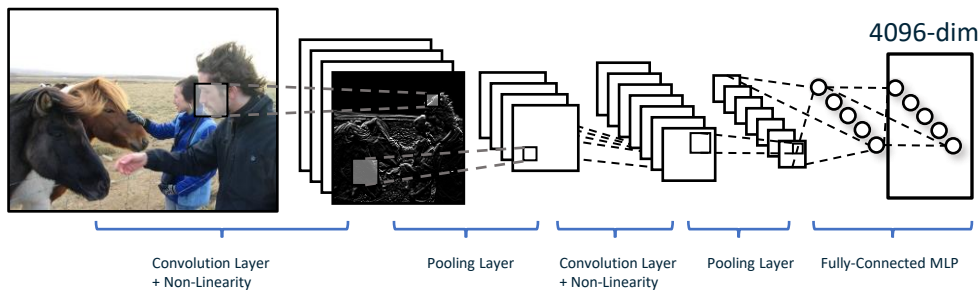
$$\begin{aligned}z &= \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\&+ h_t \odot (1 - z)\end{aligned}$$

Neural Image Captioning

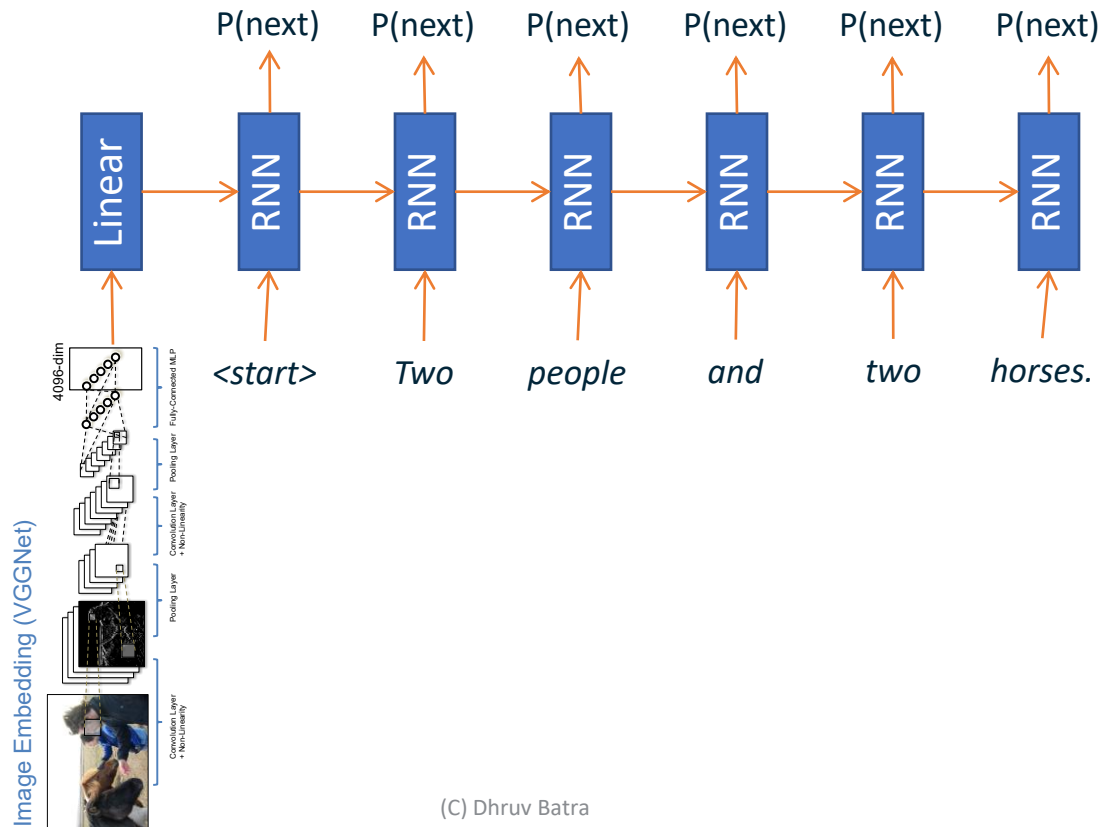


Neural Image Captioning

Image Embedding (VGGNet)



Neural Image Captioning



Neural Image Captioning

