Topics:

- Recurrent Neural Networks
- Long Short-Term Memory

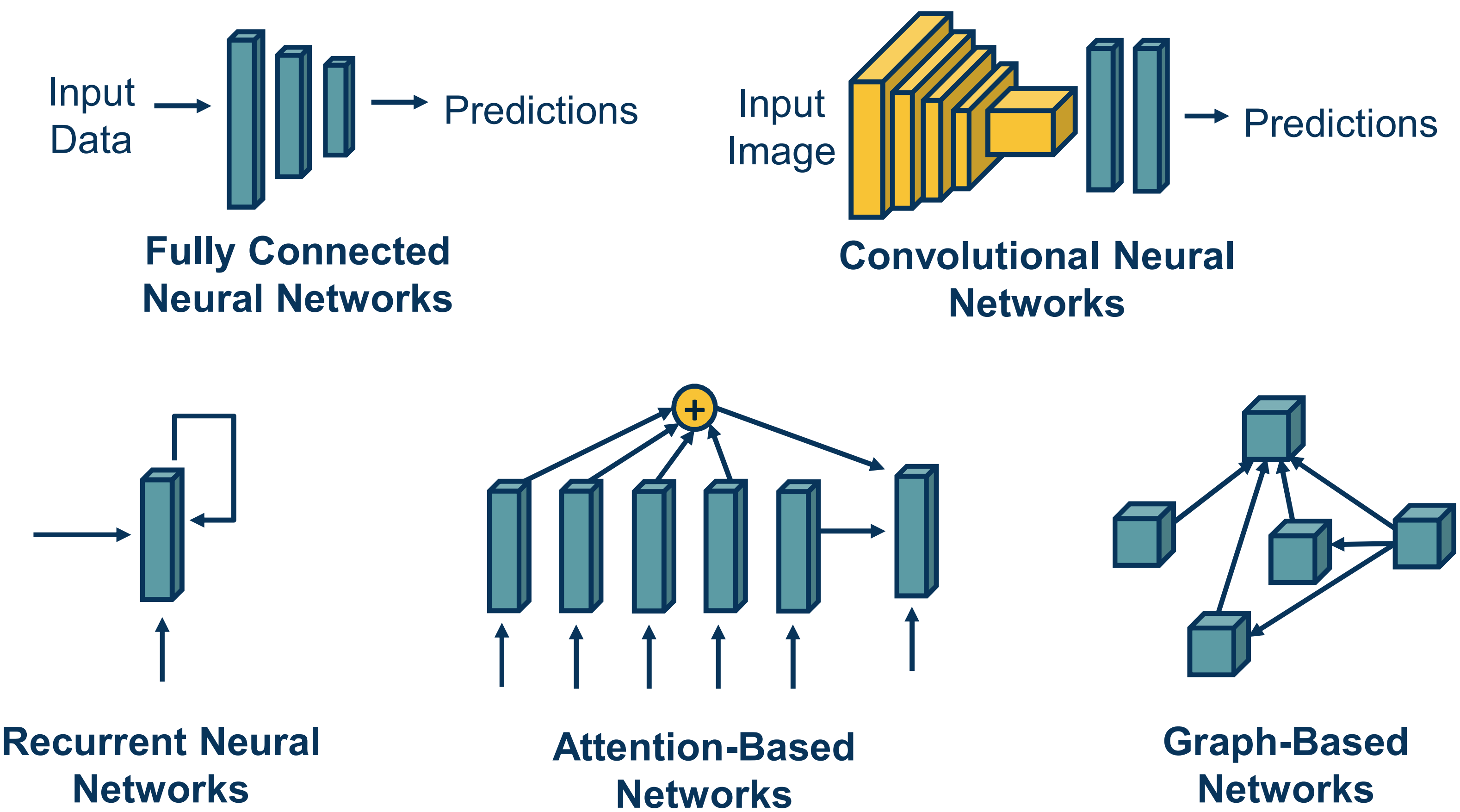
CS 4644-DL / 7643-A

ZSOLT KIRA

- **Assignment 3**
  - **UPDATE:** Now due **March 16th 11:59pm EST.**

- **Projects**
  - Project proposal due **March 17th** (into grace period)

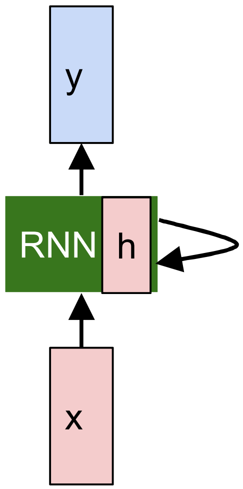
- Meta office hours on language models!



The Space of Architectures

(Vanilla) Recurrent Neural Network

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



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

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

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Recurrent Neural Network

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

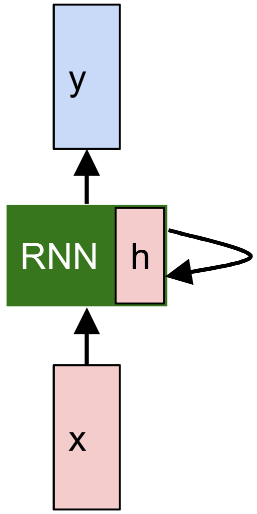
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

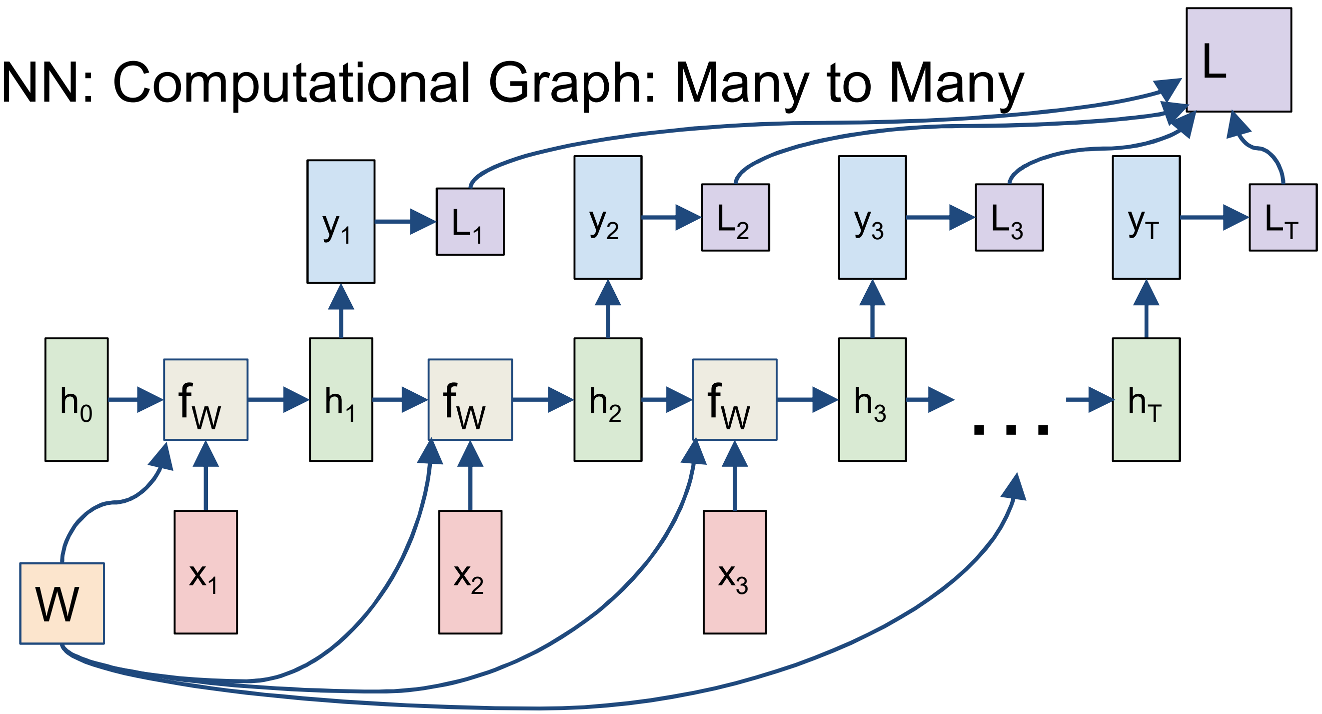
old state

input vector at some time step

some function with parameters W



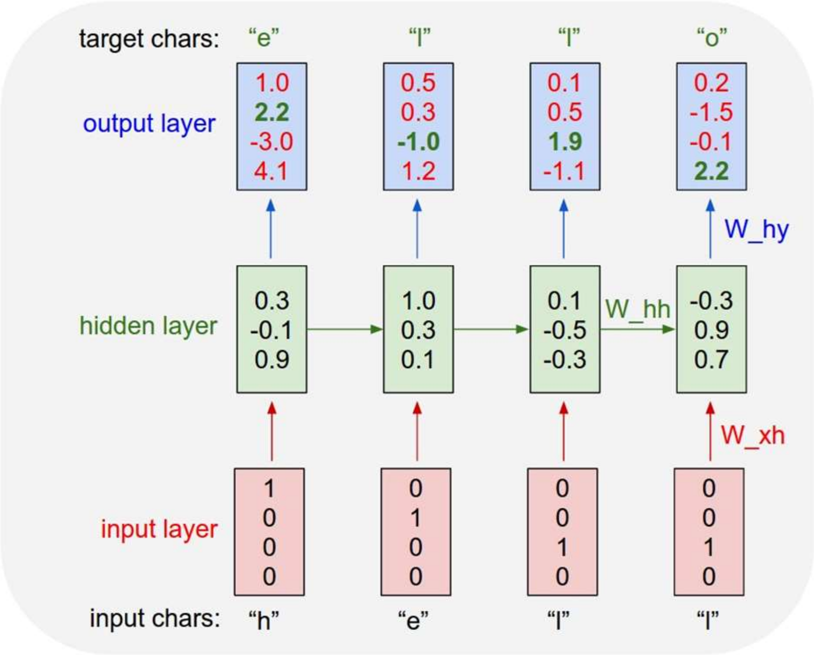
RNN: Computational Graph: Many to Many



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**



```
#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;
}
```

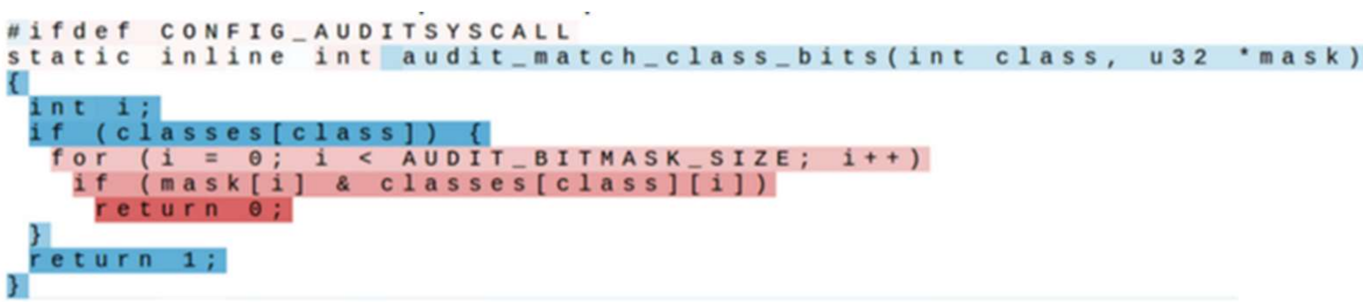
Searching for interpretable cells

Cell that turns on inside comments and quotes:

quote/comment cell

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

Searching for interpretable cells



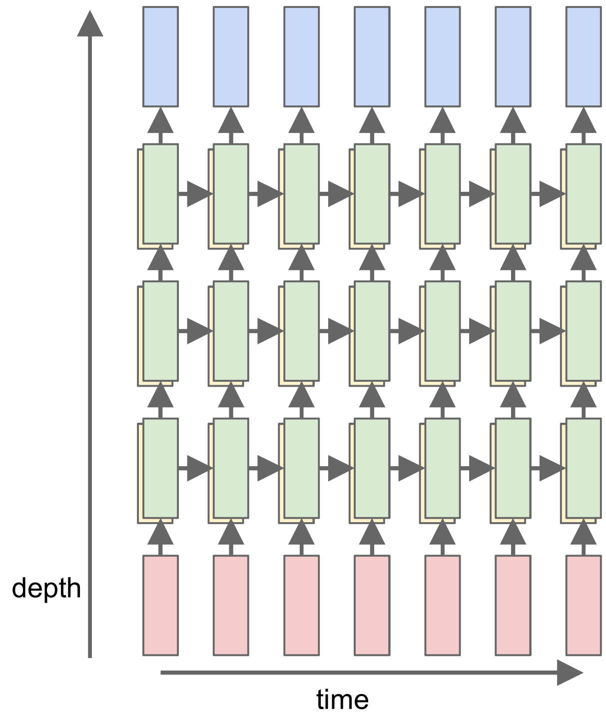
code depth cell

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

Multilayer RNNs

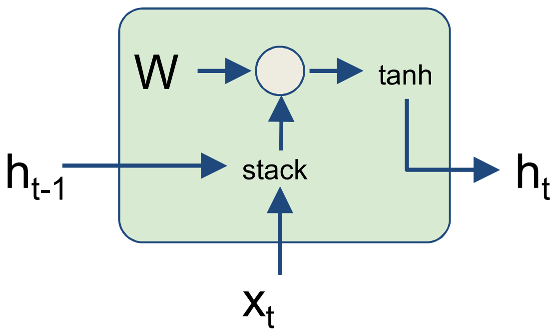
$$\boxed{h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}}$$

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



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

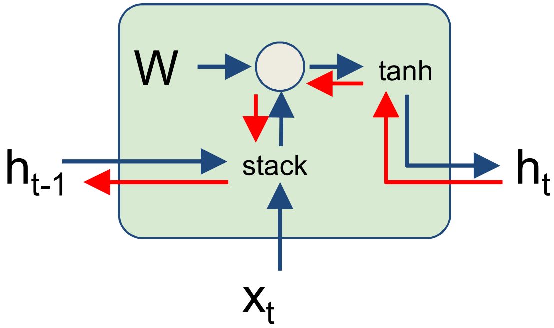


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

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

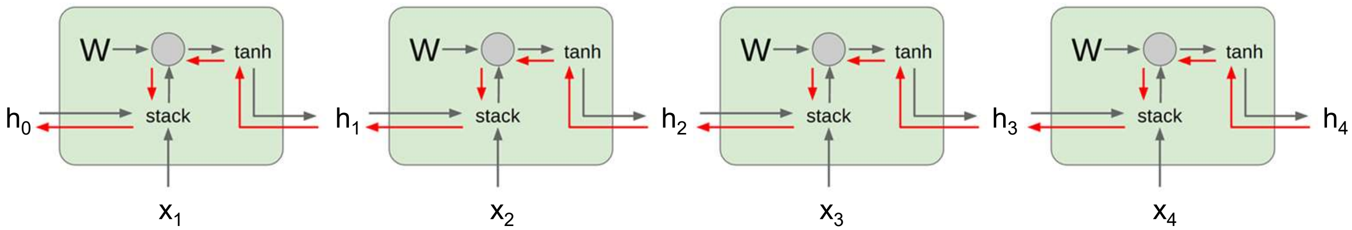
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh})



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

Vanilla RNN Gradient Flow

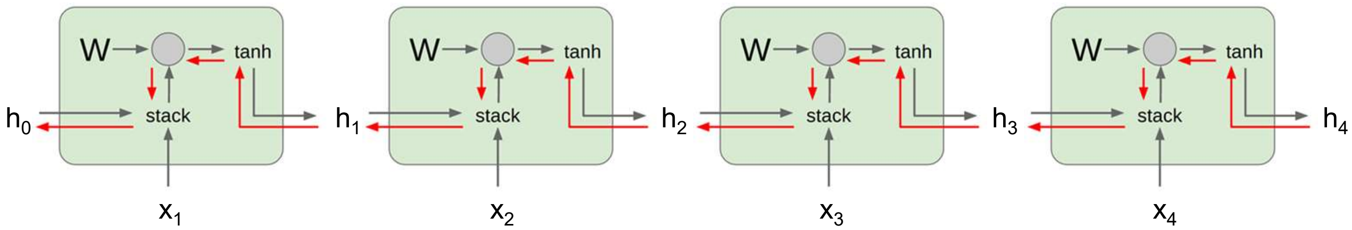
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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



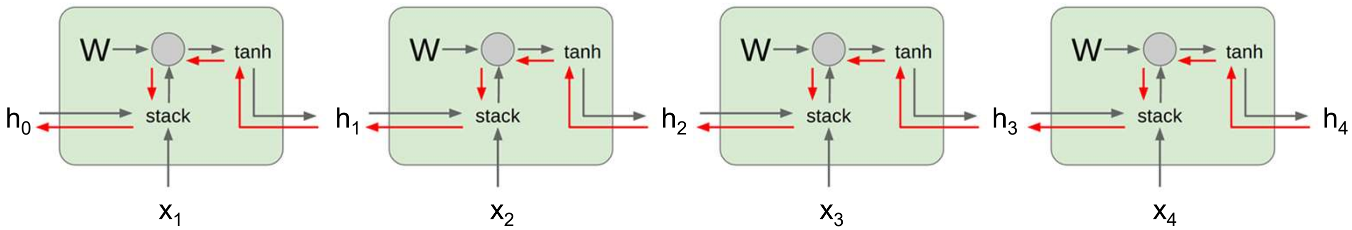
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

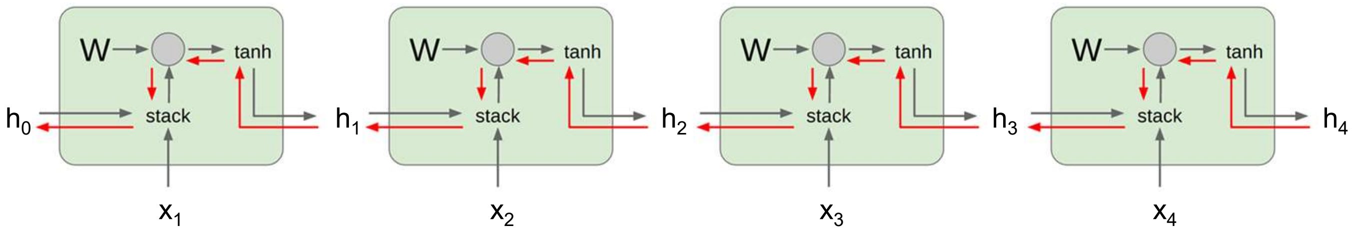
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

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



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

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

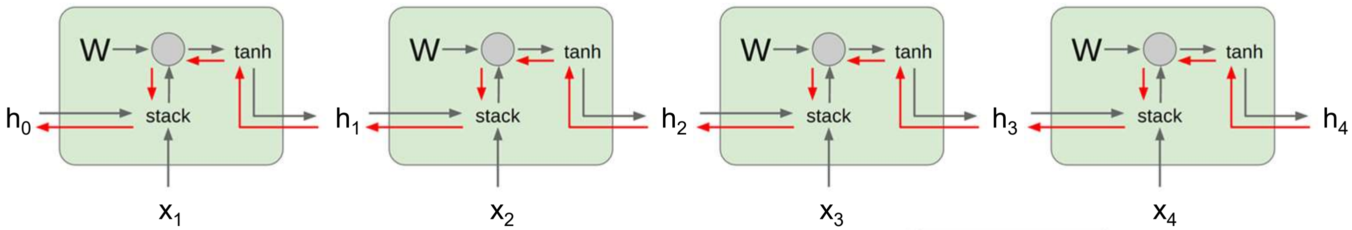
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W}$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Georgia Tech

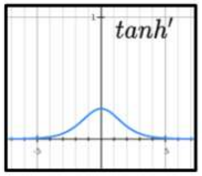
Vanilla RNN Gradient Flow

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



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

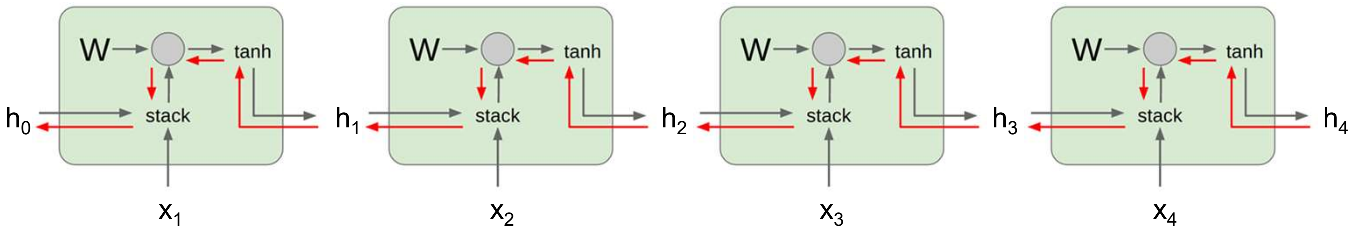
Always < 1
Vanishing gradients



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

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)

Largest singular value > 1: **Exploding gradients**

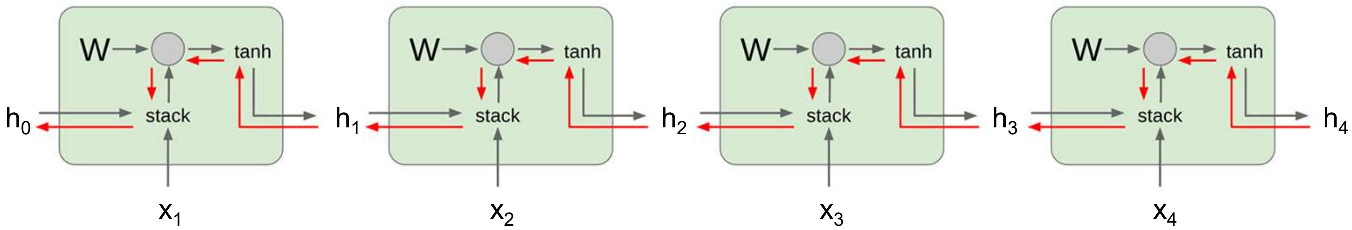
Largest singular value < 1: **Vanishing gradients**

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

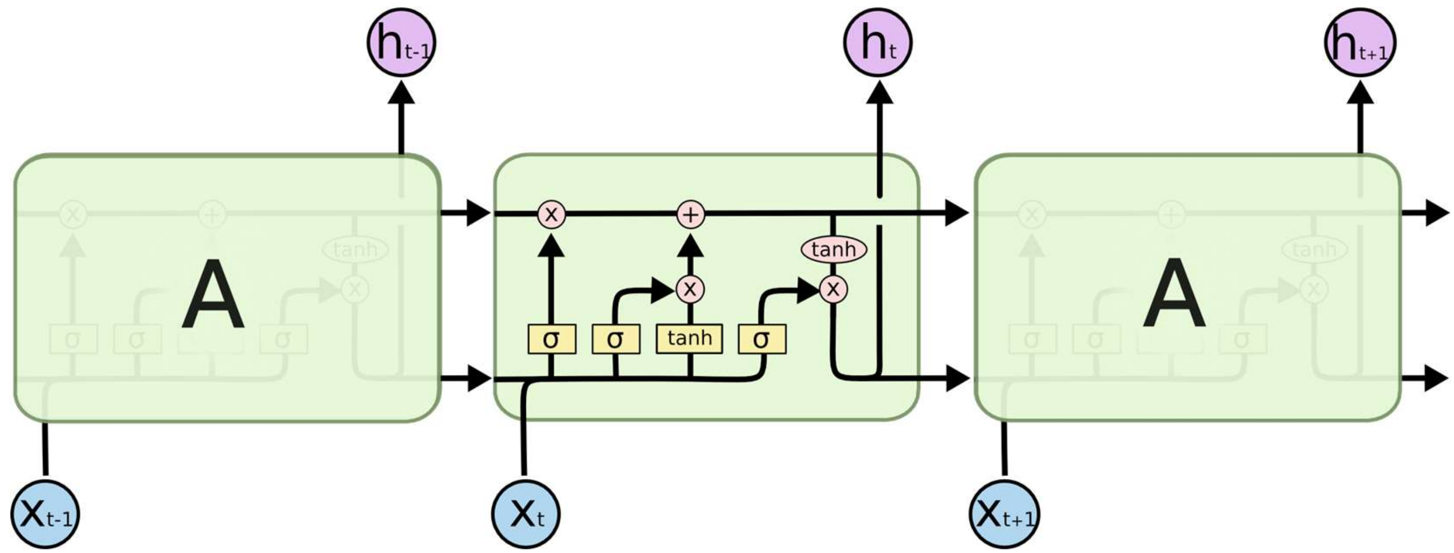
$$\begin{pmatrix}i\\f\\o\\g\end{pmatrix} = \begin{pmatrix}\sigma\\\sigma\\\sigma\\\tanh\end{pmatrix} W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}$$

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

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

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Meet LSTMs



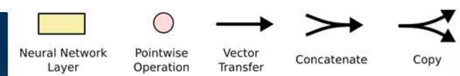


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Memory

- Cell State / Memory

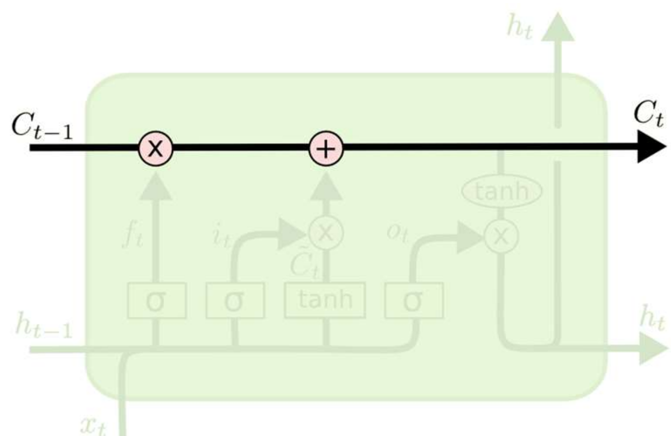
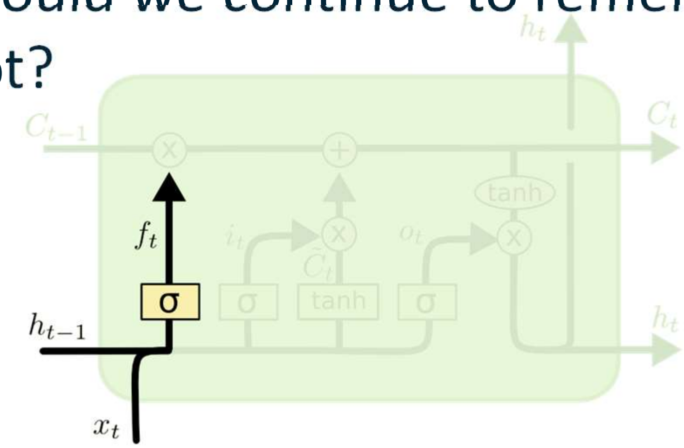


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Forget Gate

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

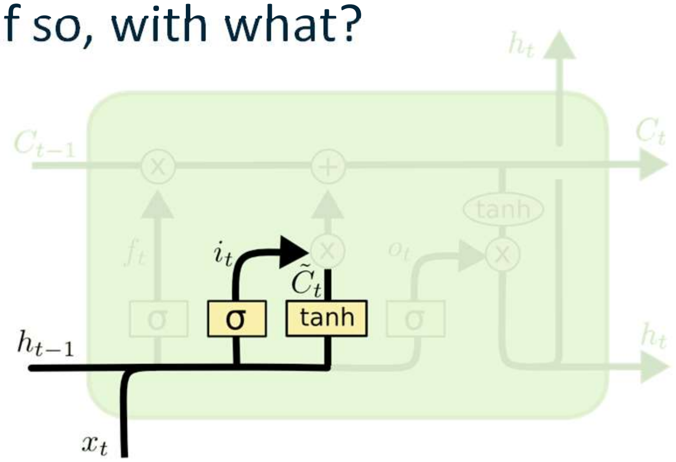


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Input Gate

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



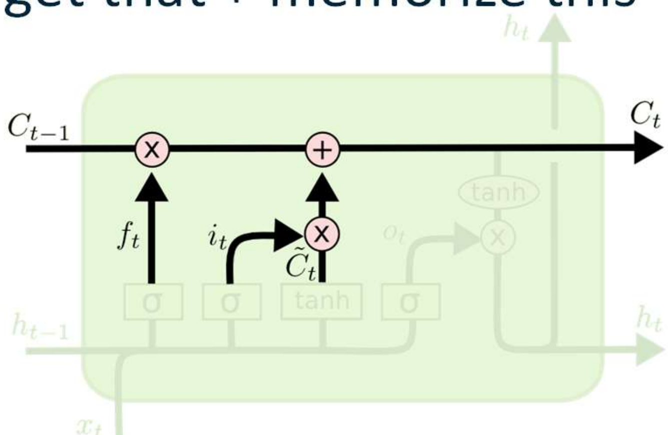
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Memory Update

- Forget that + memorize this

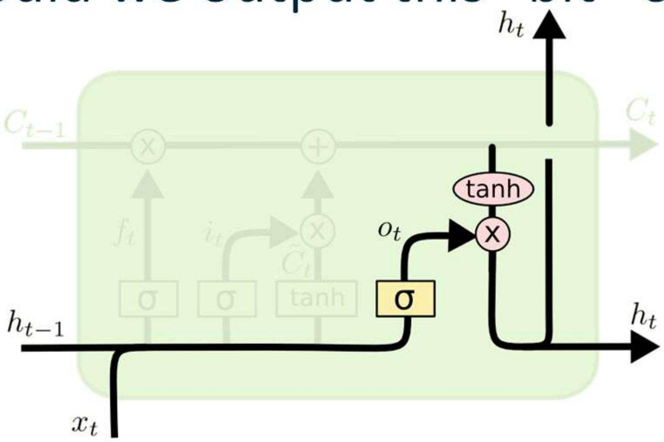


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Output Gate

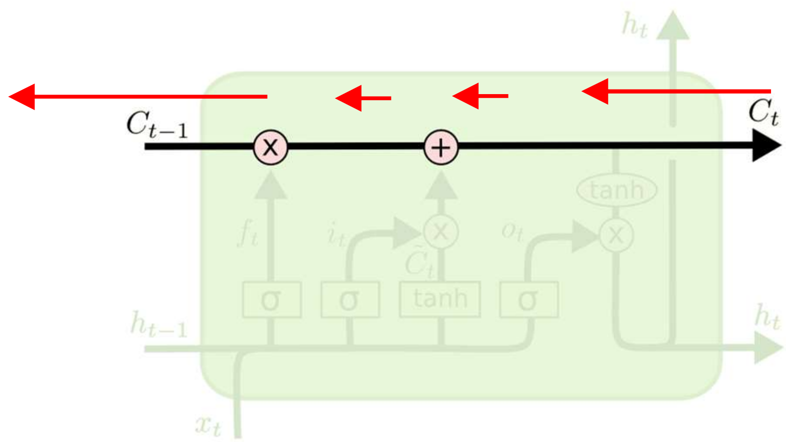
- Should we output this “bit” of information to “deeper” layers?



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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Additive Updates



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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Additive Updates

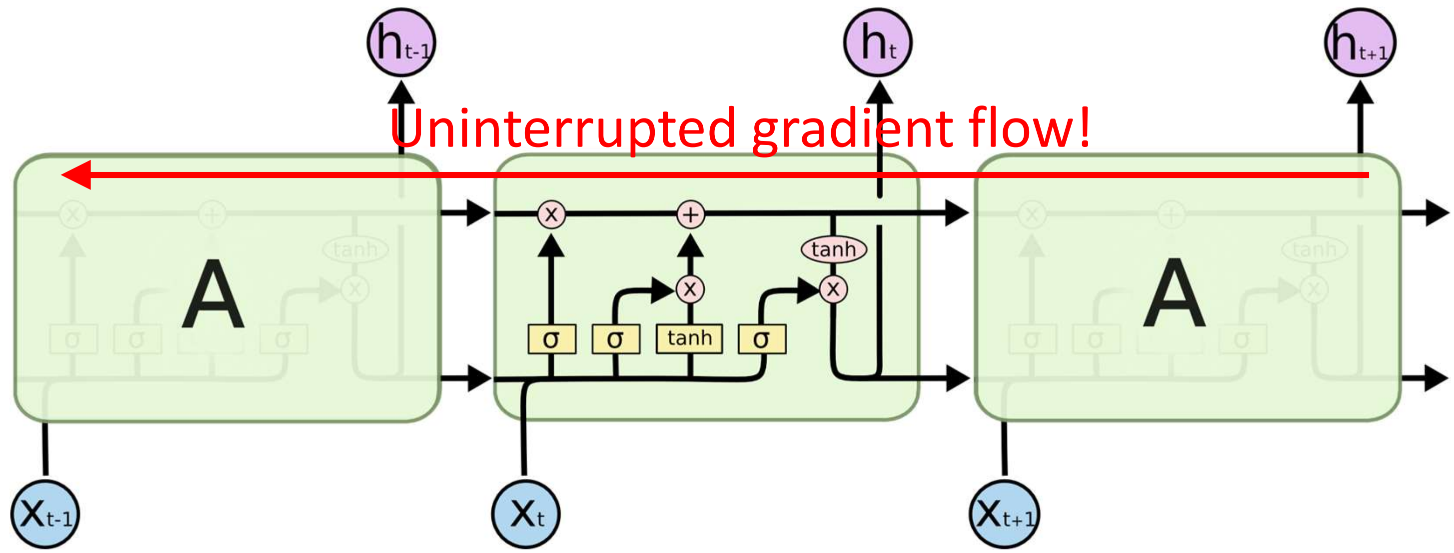
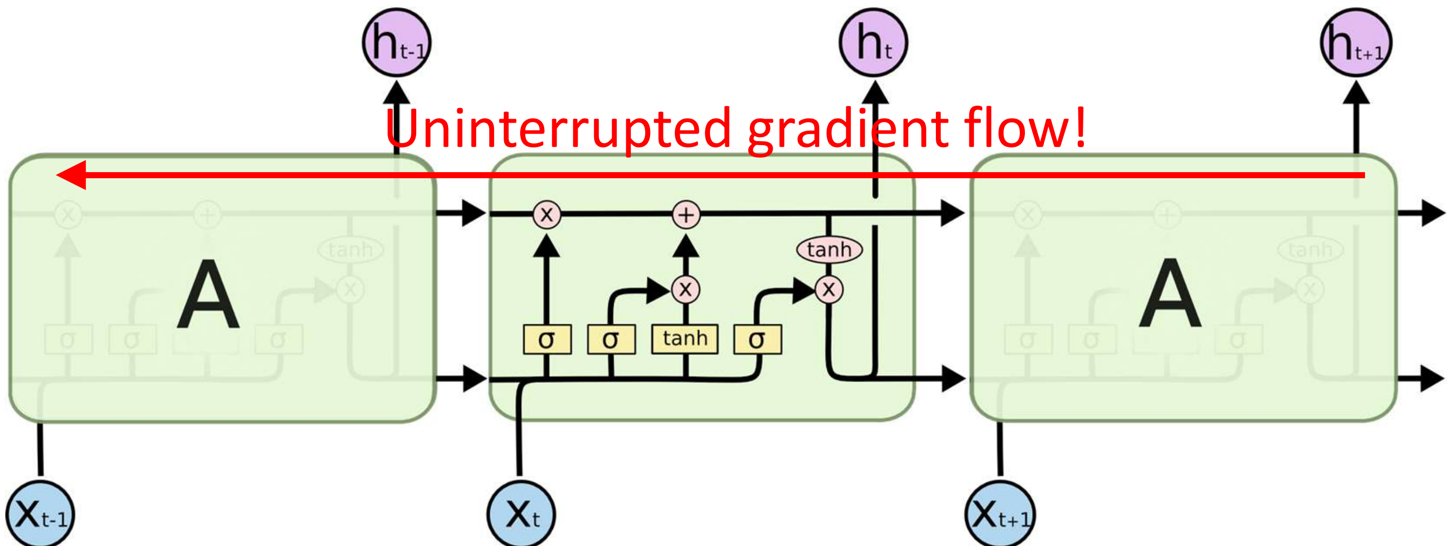
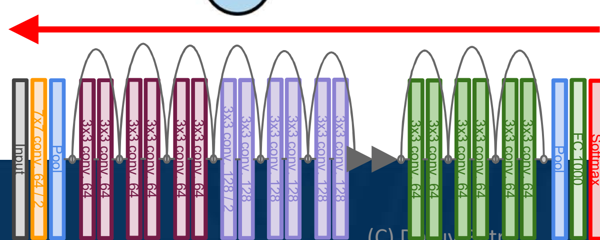


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Additive Updates





Similar to ResNet!

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs

- A pretty sophisticated cell

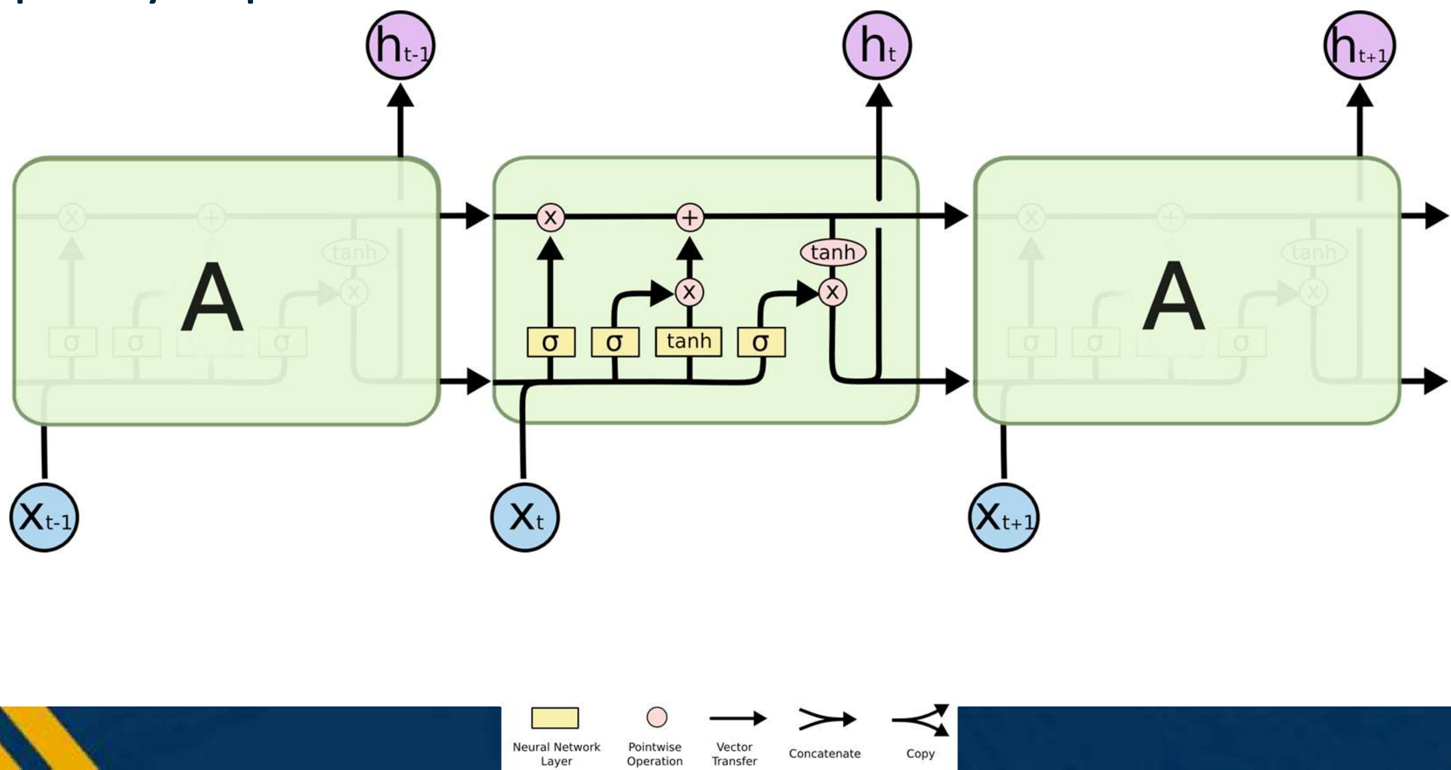
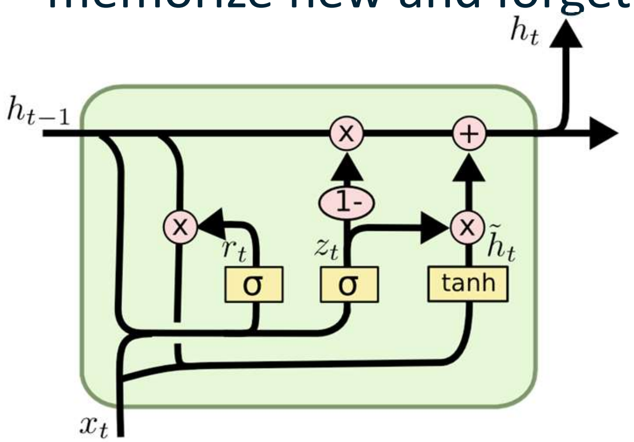


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants: Gated Recurrent Units

- Changes:
  - No explicit memory; memory = hidden output
  - Z = memorize new and forget old



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

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

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

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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Other RNN Variants

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

MUT1:

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

MUT2:

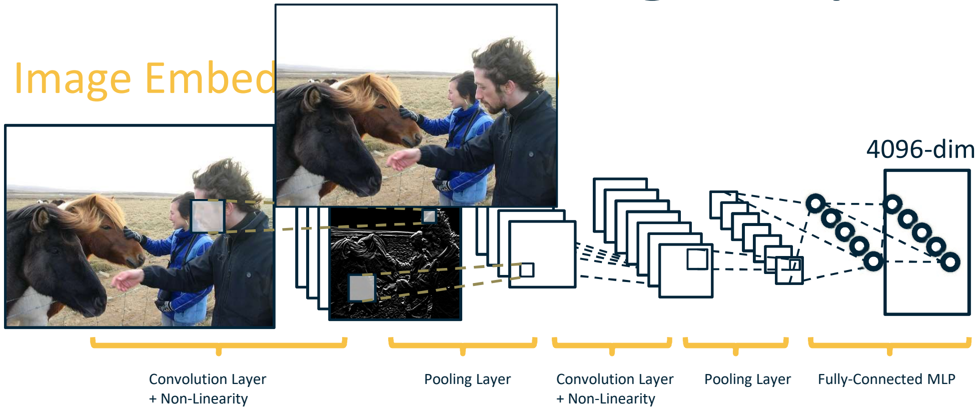
$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ \quad h_t \odot (1 - z) \end{aligned}$$

MUT3:

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

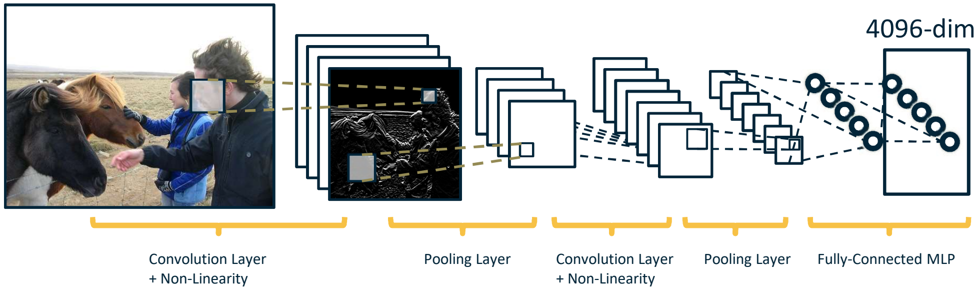
Neural Image Captioning

Image Embed

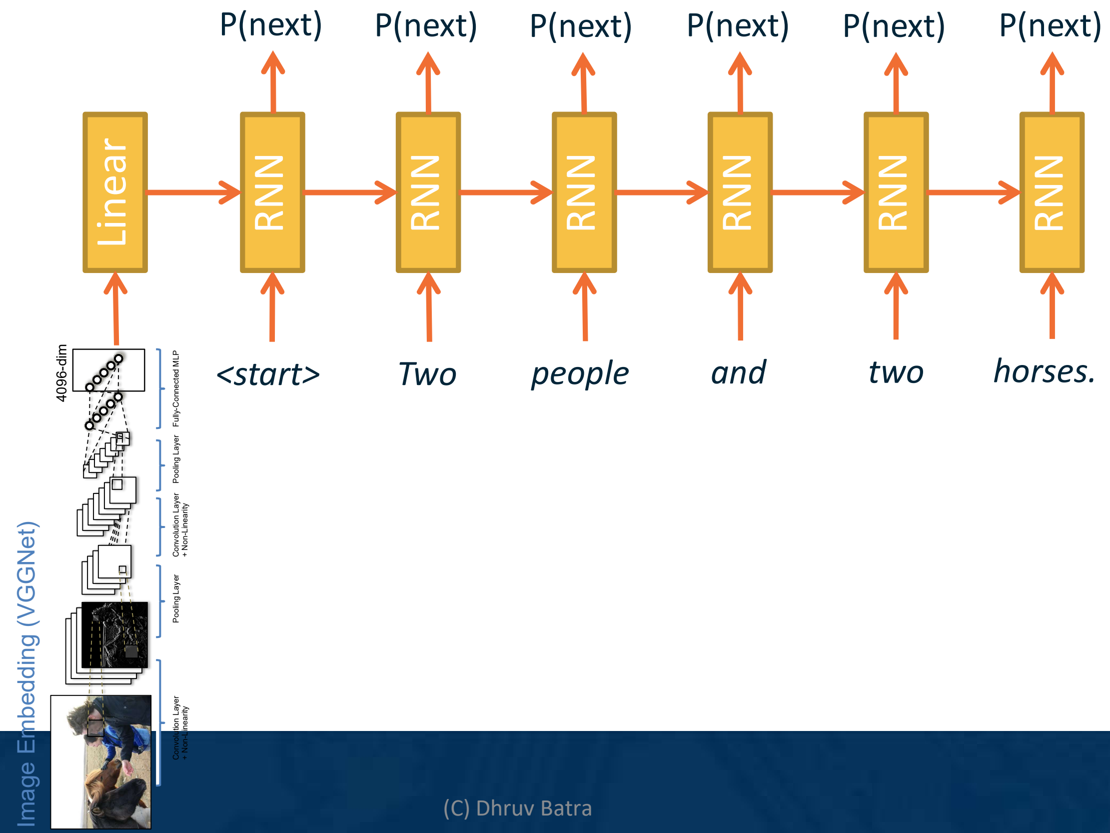


Neural Image Captioning

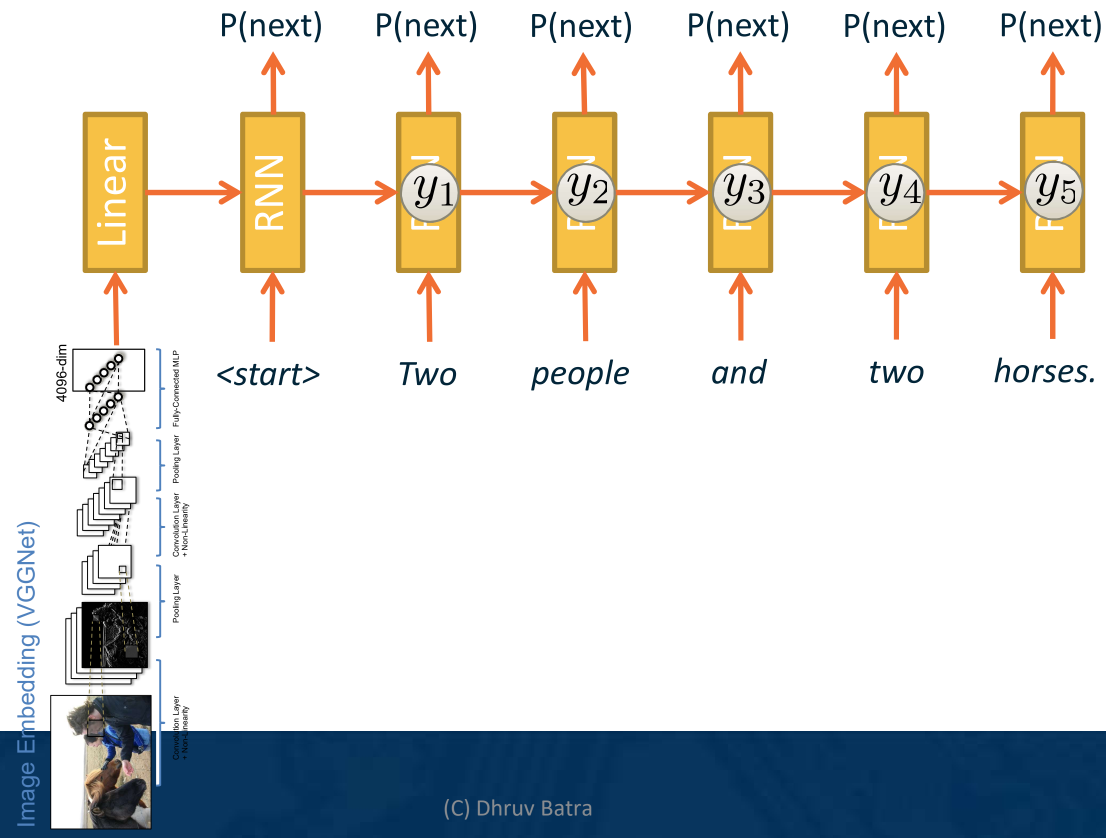
Image Embedding (VGGNet)



Neural Image Captioning



Neural Image Captioning



One-hot representations

- Simple way how to encode discrete concepts, such as words

Example:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday  = [1 0 0 0 0]
Tuesday = [0 1 0 0 0]
is      = [0 0 1 0 0]
a       = [0 0 0 1 0]
today   = [0 0 0 0 1]
```

Also known as 1-of-N (where in our case, N would be the size of the vocabulary)

An aside: Representing words

You can get a lot of value by representing a word by means of its neighbors

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

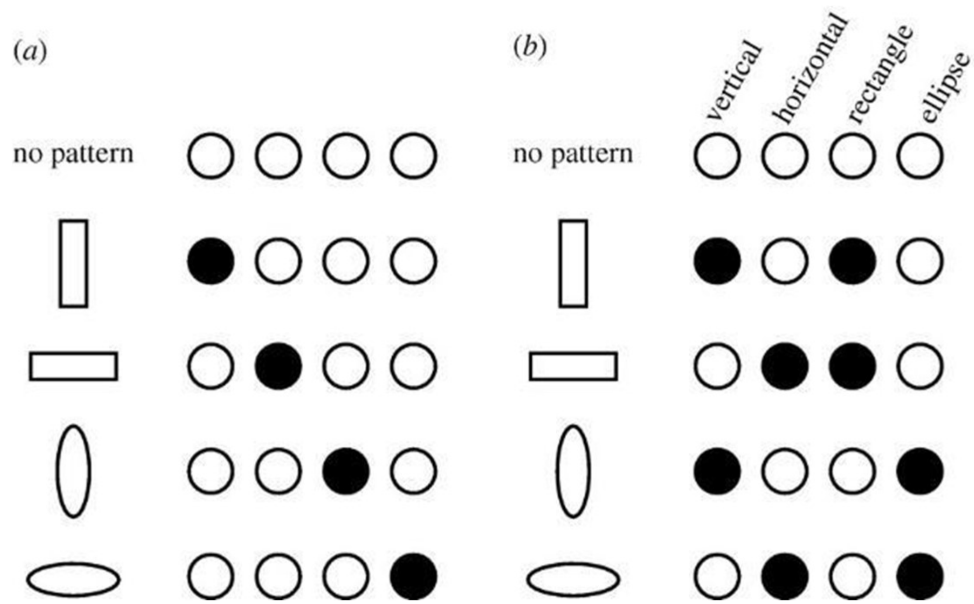
government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

You can vary whether you use local or large context to get a more syntactic or semantic clustering

Distributed Representations Toy Example

- Can we interpret each dimension?



Power of distributed representations!

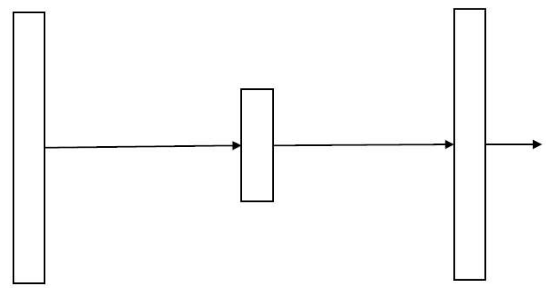
Local ● ● ○ ● = VR + HR + HE = ?

Distributed ● ● ○ ● = V + H + E ≈ ○

Slide Credit: Moontae Lee

Vector representations

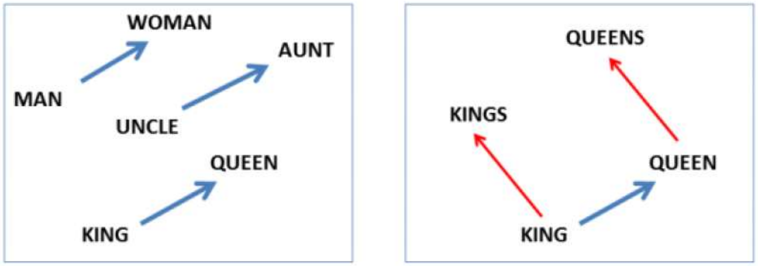
- Instead of a sparse one-hot vector, represent words as a **dense** vector



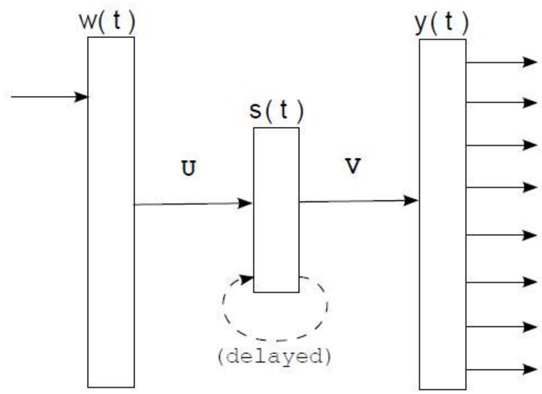
- Bigram neural language model
- Previous word is used to predict the current word by going through hidden layer (classifier with as many outputs as there are words in the vocabulary)

Linguistic regularities in word vectors

- Recently, it was shown that word vectors capture many linguistic properties (gender, tense, plurality, even semantic concepts like "capital city of")
- We can do nearest neighbor search around result of vector operation "King – man + woman" and obtain "Queen" (*Linguistic regularities in continuous space word representations* (Mikolov et al, 2013))



Word representations using RNNs



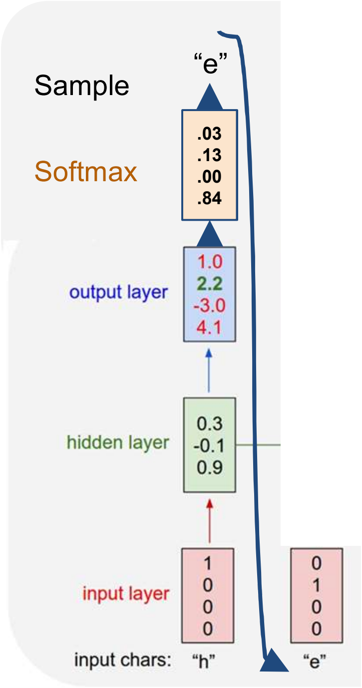
- Input layer $\mathbf{w}$ and output layer $\mathbf{y}$ have the same dimensionality as the vocabulary
- Hidden layer $\mathbf{s}$ is orders of magnitude smaller
- $\mathbf{U}$ is the matrix of weights between input and hidden layer, $\mathbf{V}$ is the matrix of weights between hidden and output layer

Georgia Tech

Potential Input Representations

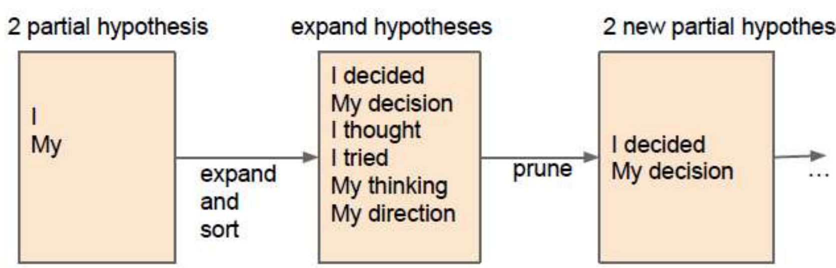
- One hot encoding -> FC layer

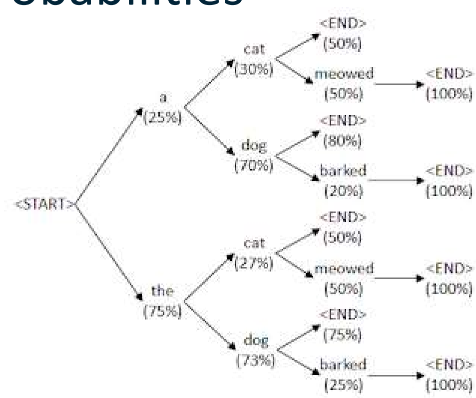
- Parameters/embeddings indexed in a table
  - Can be initialized randomly
  - Or can be initialized with pre-trained word embeddings



Beam Search

- Proceed from left to right
- Maintain N partial captions
- Expand each caption with possible next words
- Discard all but the top N new partial translations
  - Maintain score for each, e.g. product of probabilities



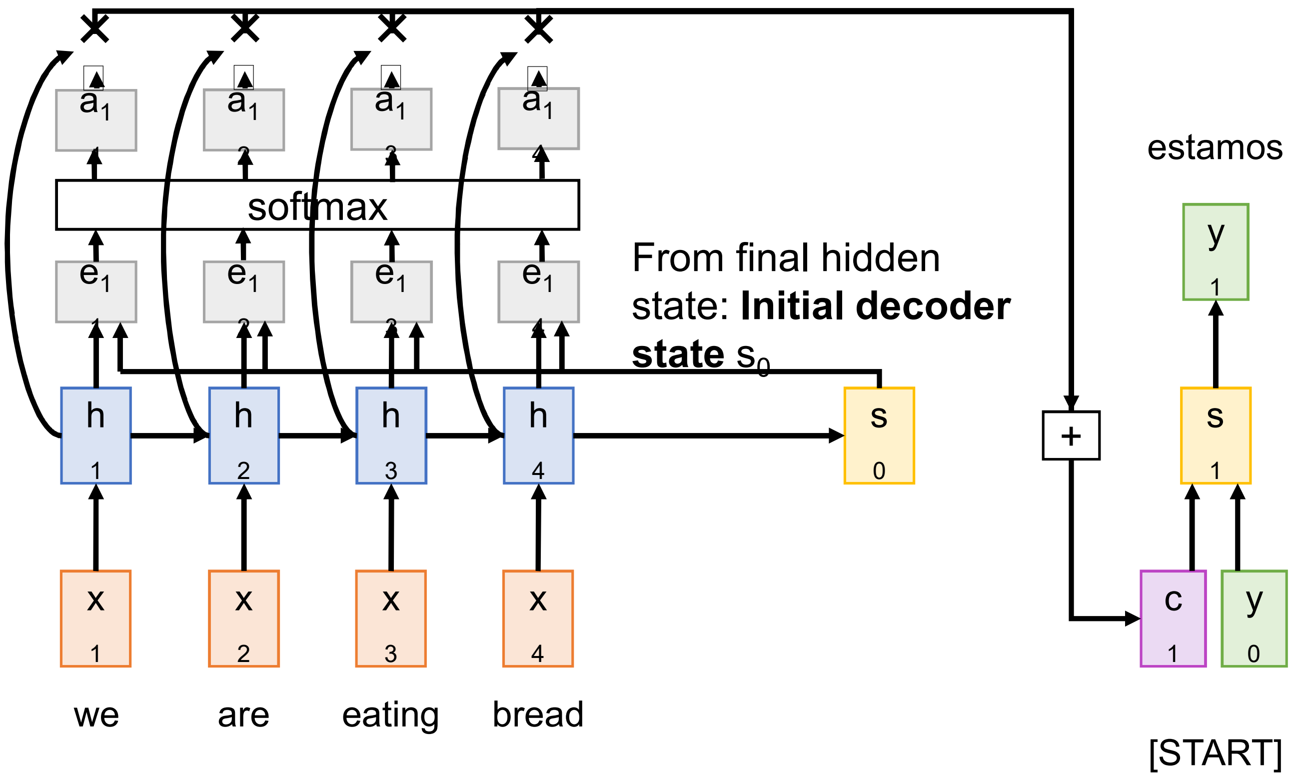


https://geekyisawesome.blogspot.com/2016/10/using-beam-search-to-generate-most.html

Summary

- RNNs leverage *internal state* information to propagate information across sequence
  - Same shared function/parameters
- LSTMs improve gradient flow across the computation graph with *gating*
- **Next time**: Attention mechanisms and transformers to explicitly access and propagate information

Machine Translation with RNNs and Attention



Compute alignment scores

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize to get attention weights

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Set context vector c to a linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

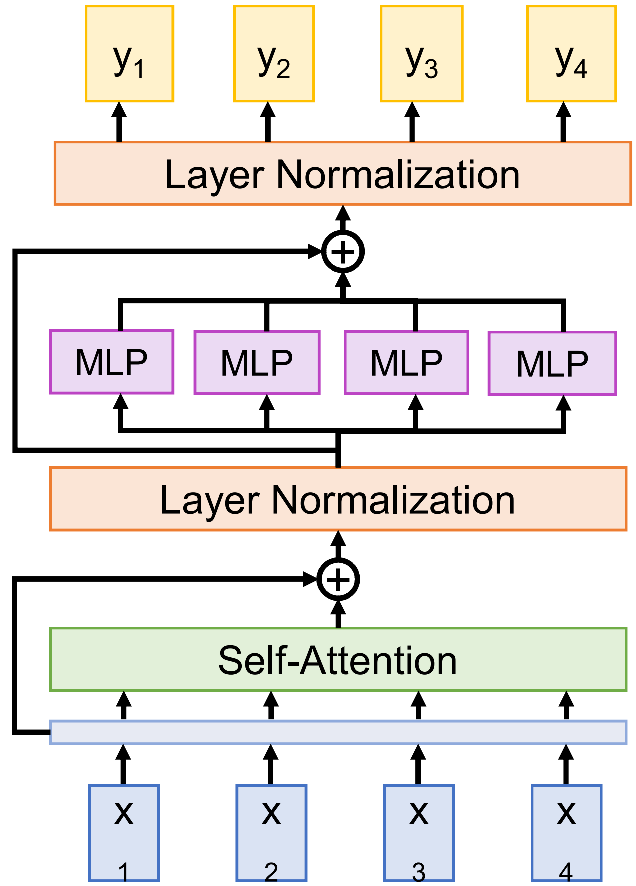
The Transformer

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson