APL 745: Deep Learning in Mechanics

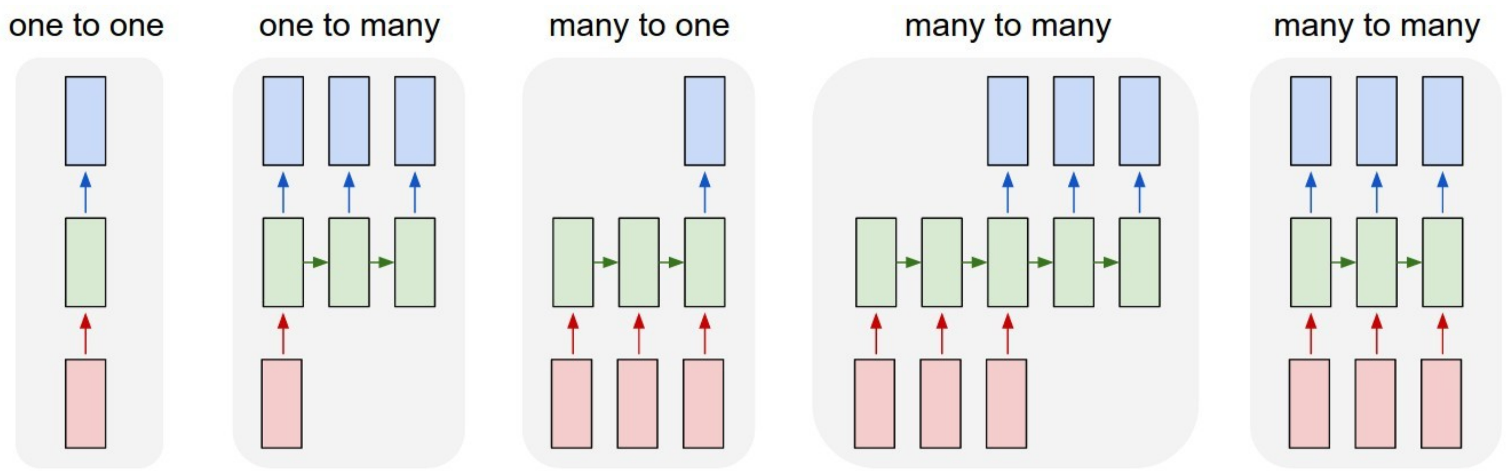
Topics:

- Recurrent Neural Networks (RNNs)
  - (Truncated) BackProp Through Time (BPTT)
  - LSTMs

Sitikantha Roy

IIT Delhi

New Topic: RNNs



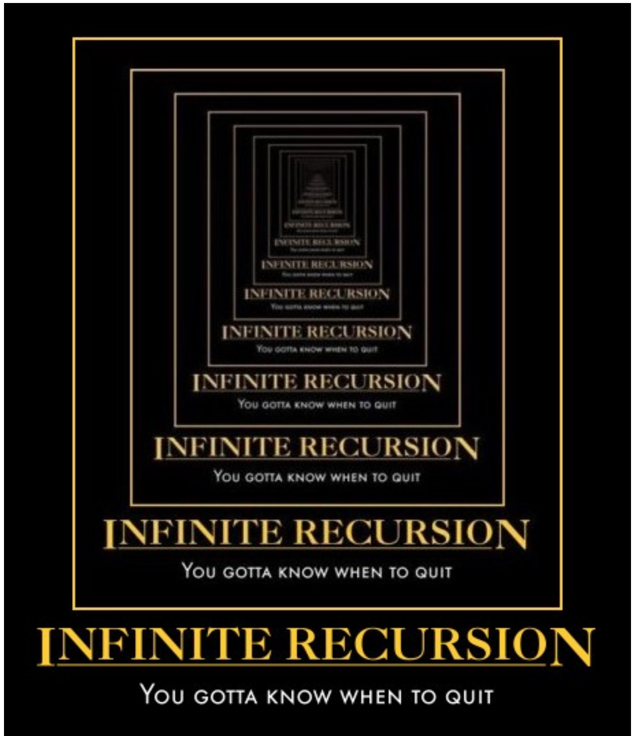


Image Credit: Andrej Karpathy

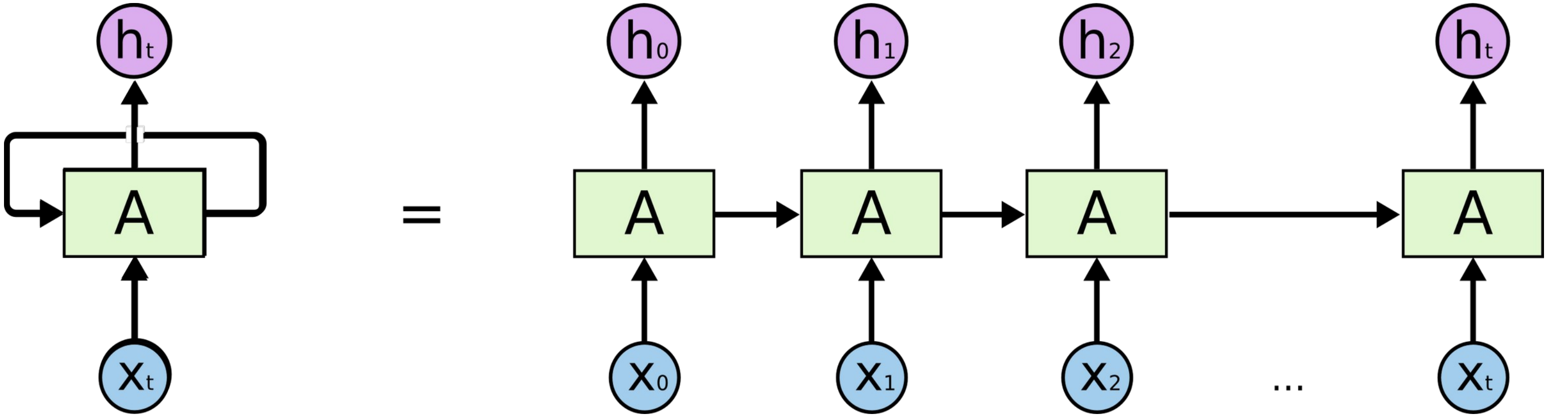
New terminologies

- 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)

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. Recurrent Neural Networks have loops.



This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on.

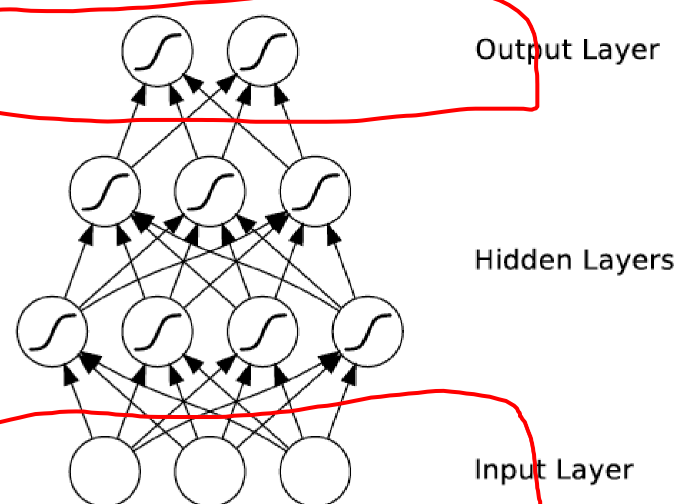
What’s wrong with MLPs?

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

- Problem 2: Pure feed-forward processing
  - No “memory”, no feedback

- Each input to network, was independent of the previous or future inputs

- Computations, output and decisions for two successive Images (CNN) are completely Independent of each other.



$y \in \mathbb{R}^{c}$

$\vec{x} \in \mathbb{R}^{d_i}$

Image Credit: Alex Graves, book

Why model sequences?

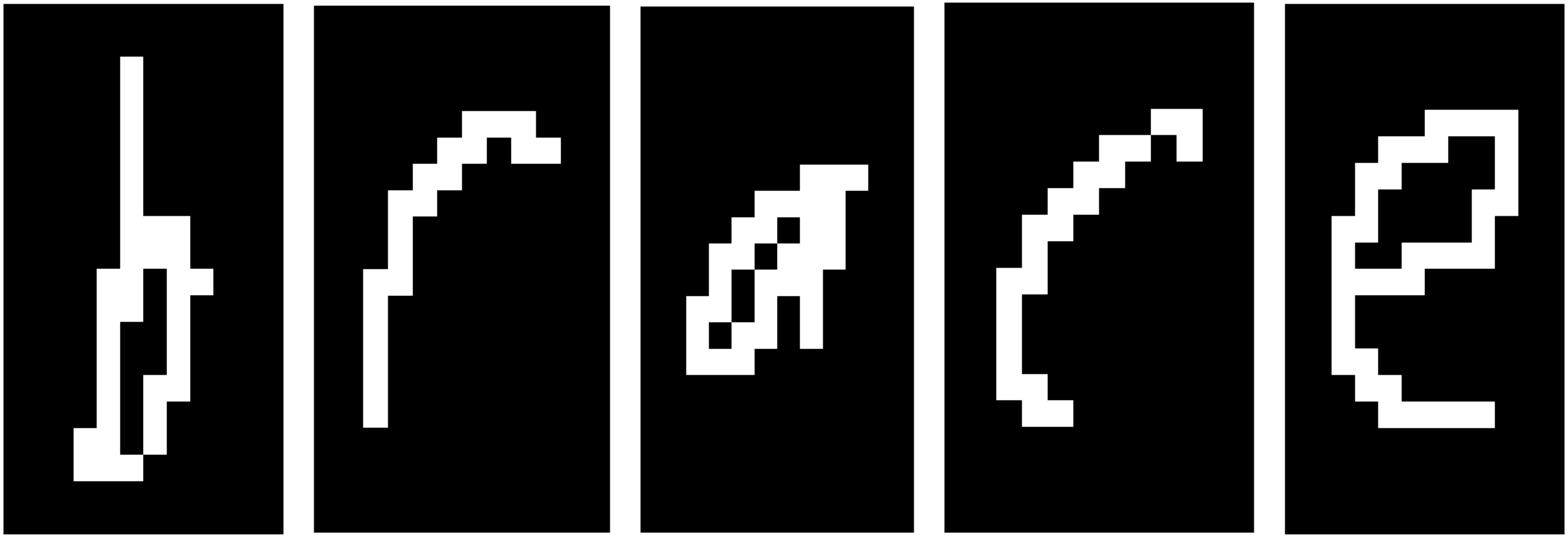


Figure Credit: Carlos Guestrin

Why model sequences?

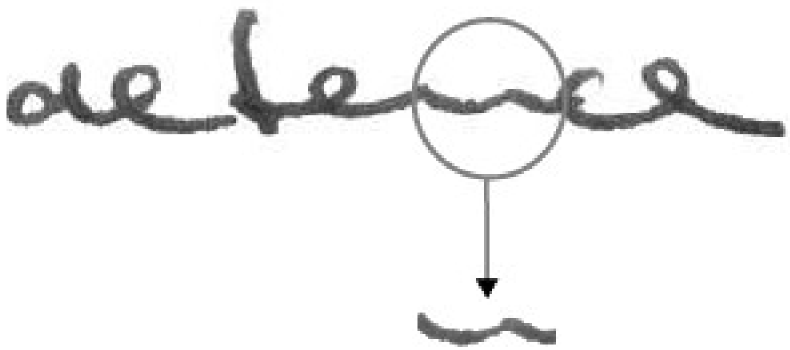
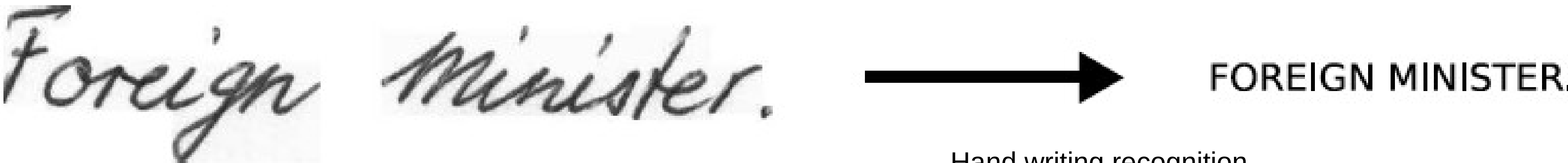
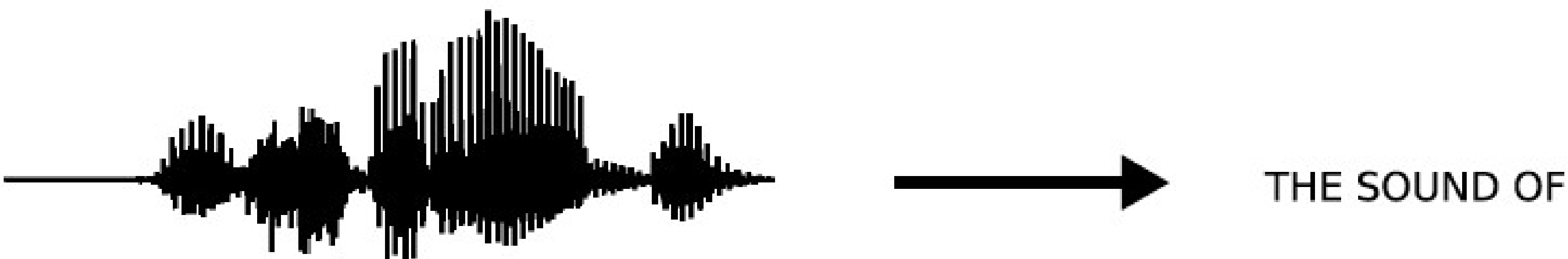


Image Credit: Alex Graves

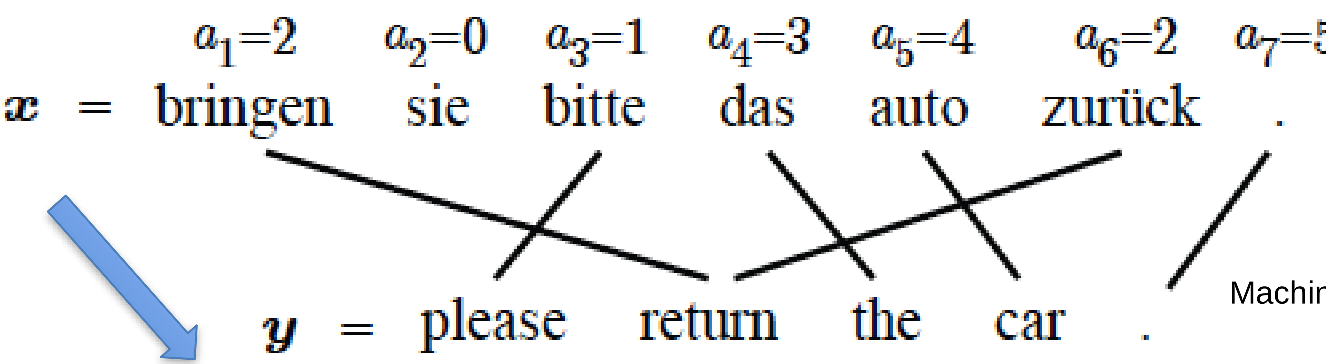
Sequences are everywhere…



Hand writing recognition



Voice recognition, speech to text conversion. Both speech and text are sequence

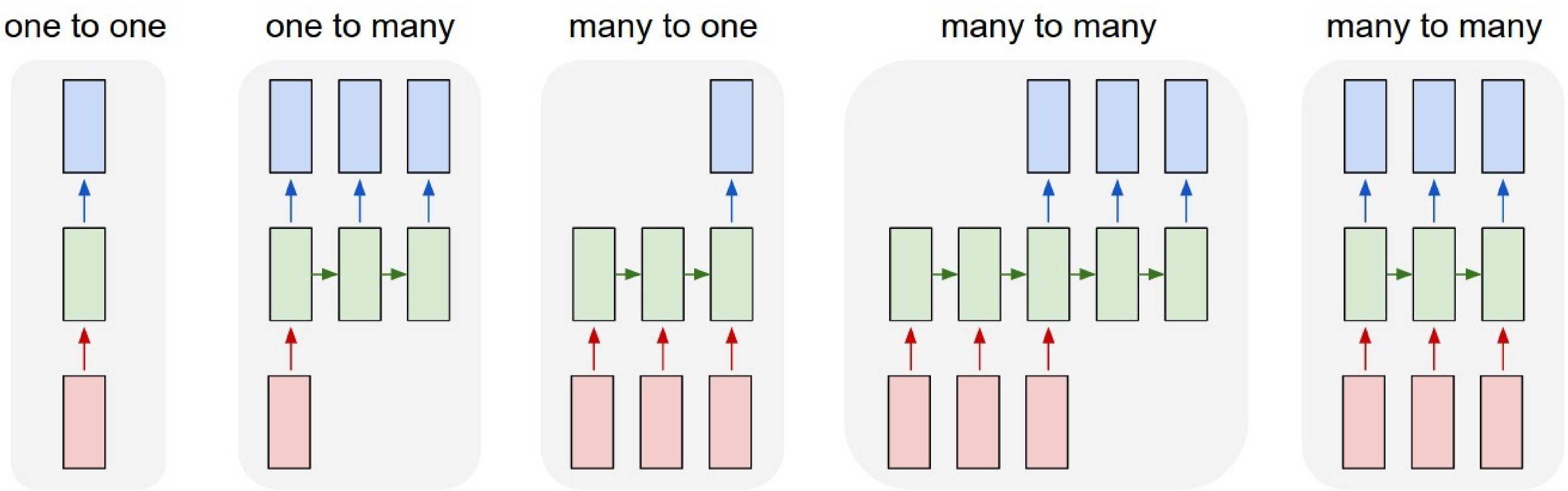


Machine translation

Image Credit: Alex Graves and Kevin Gimpel

Sequences in Input or Output?

- It's a spectrum…



Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

Input: No sequence

Output: Sequence

Example: Im2Caption

Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

Input: Sequence

Output: Sequence

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

Image Credit: Andrej Karpathy

2 Key Ideas

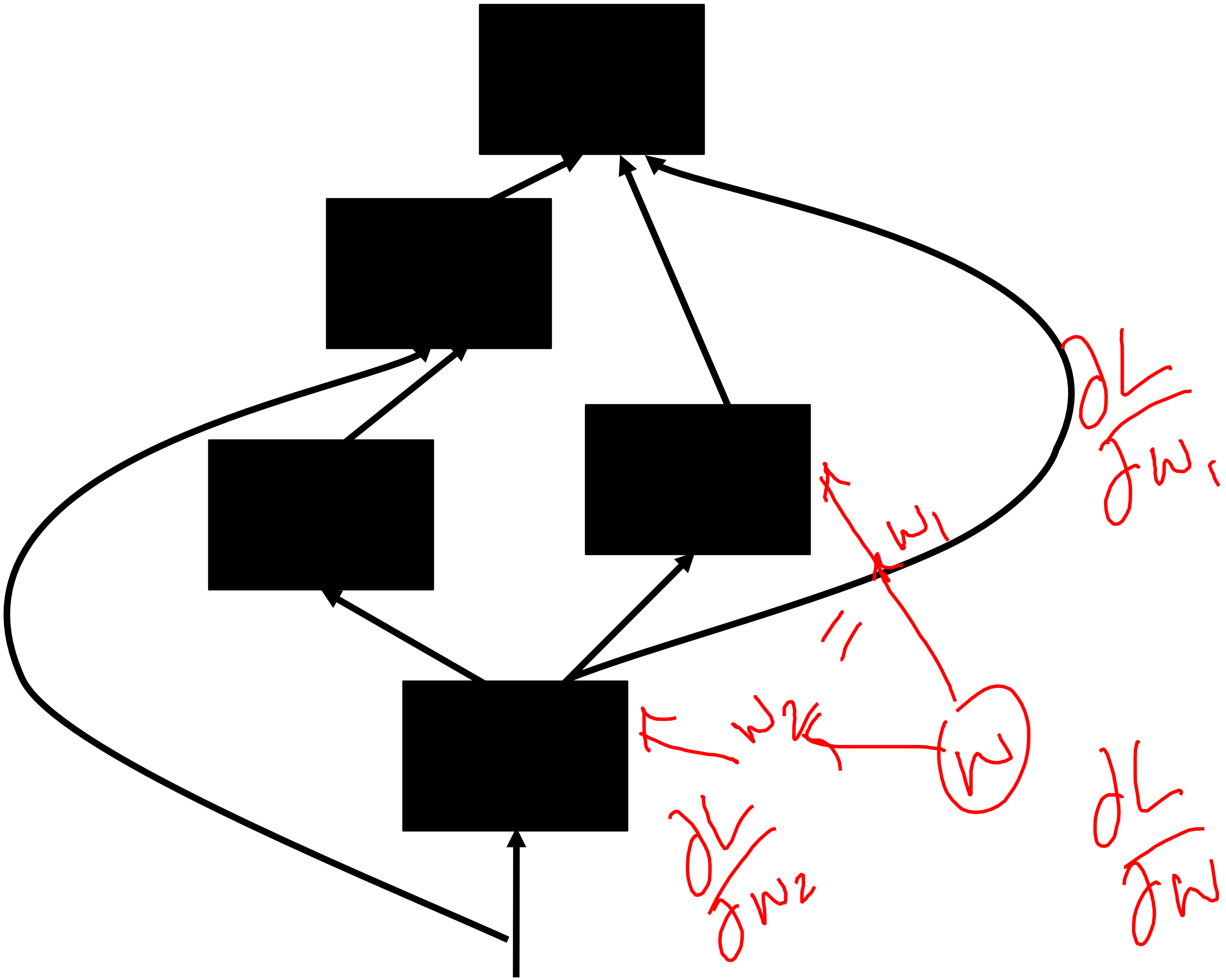
- Parameter Sharing
  - in computation graphs = adding gradients

- “Unrolling”
  - in computation graphs with parameter sharing

The above two accommodates the following …

- Account for dependency between inputs
- Account for variable number of inputs
- Make sure the function executed at each time step is the same. Why?

Computational Graph



How do we model sequences?

- No input

$$s_t = f_\theta(s_{t-1})$$

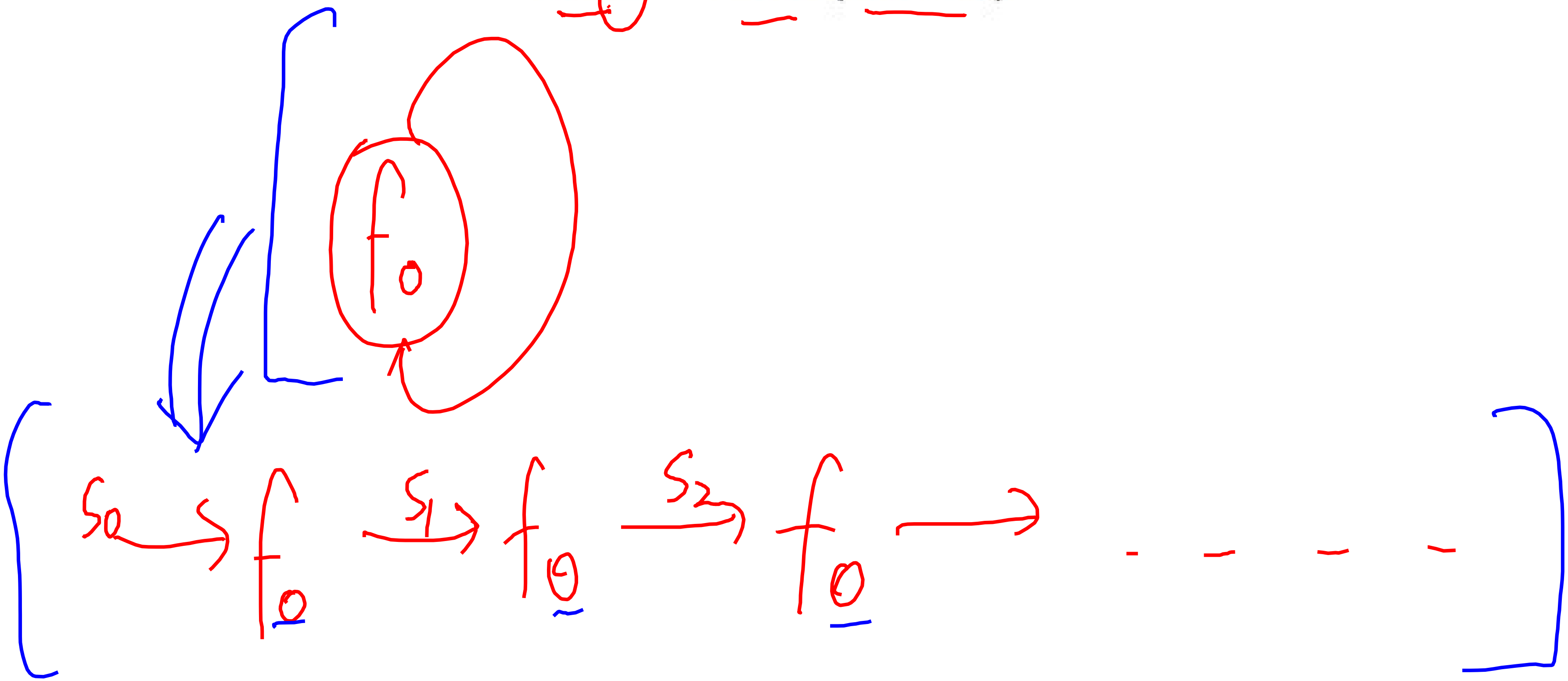


Image Credit: Bengio, Goodfellow, Courville

How do we model sequences?

- With inputs

$$s_t = f_\theta(s_{t-1}, x_t)$$

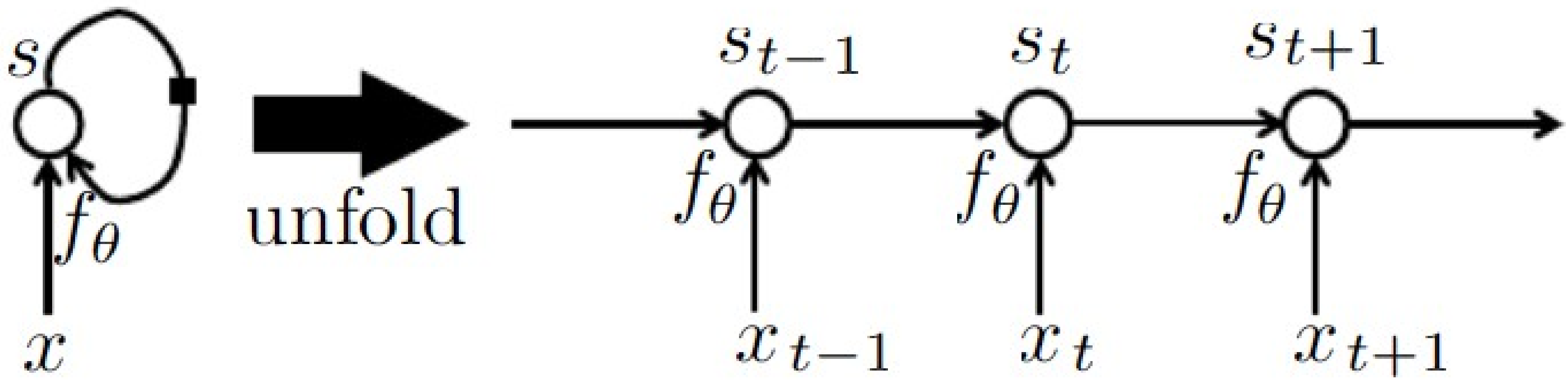


Image Credit: Bengio, Goodfellow, Courville

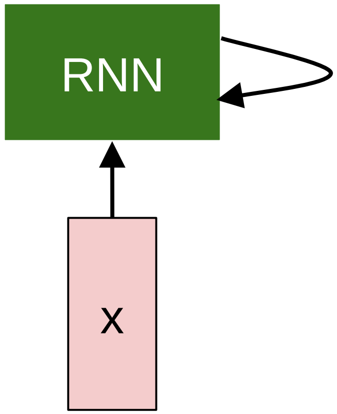
2 Key Ideas

- Parameter Sharing
  - in computation graphs = adding gradients

- “Unrolling”
  - in computation graphs with parameter sharing

- Parameter sharing + Unrolling
  - Allows modeling arbitrary sequence lengths!
  - Keeps numbers of parameters in check

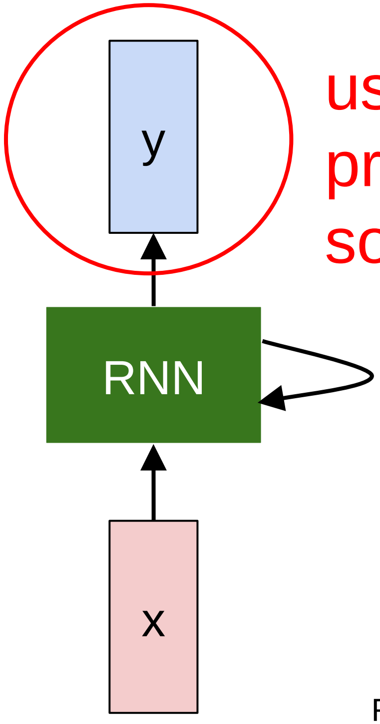
Recurrence in Recurrent Neural Network



Key idea: RNNs have an internal state that is updated as the sequence is processed. Self looping..

Internal sate is some kind of memory, which will keep track of what has happened before in the data in the past. Retaining the information of the past.

Recurrent Neural Network



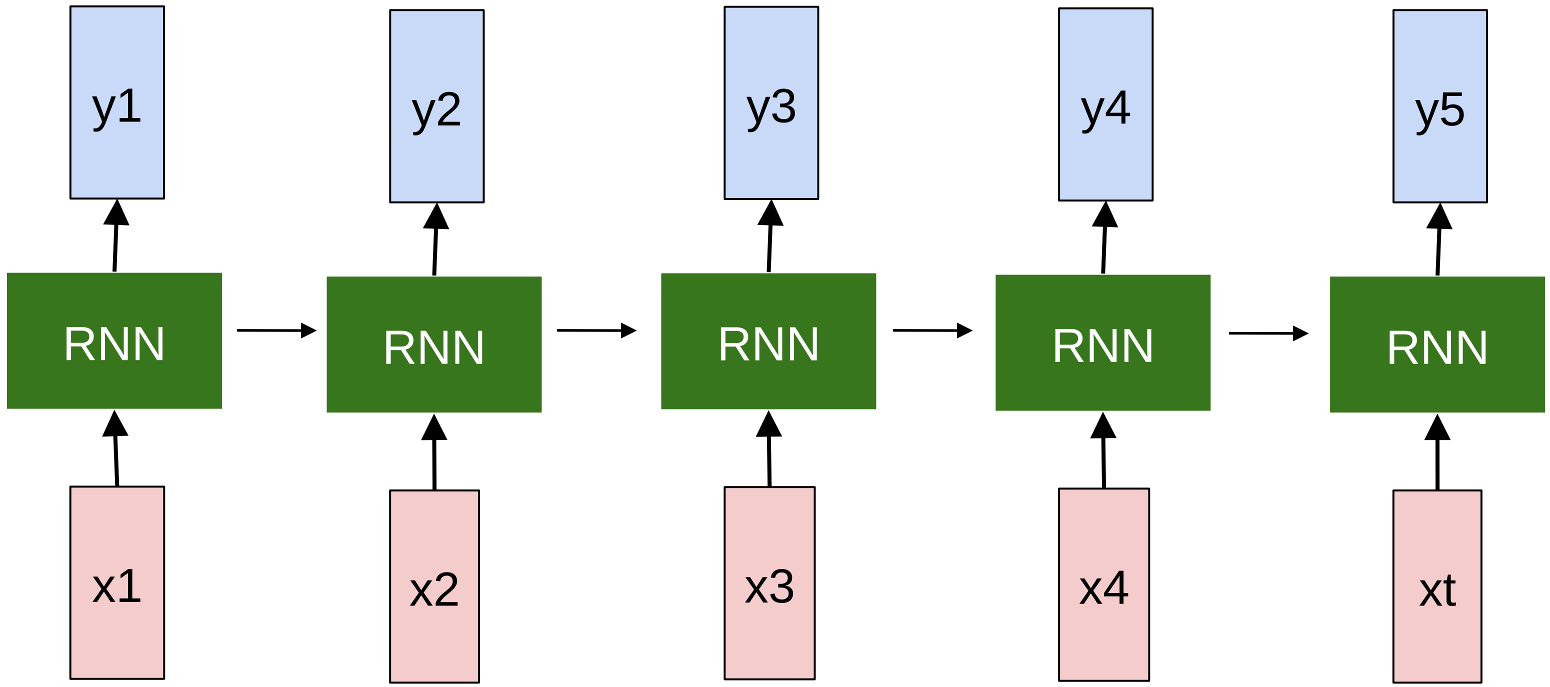
usually want to predict a vector at some time steps

RNN Block: Can have 1 or many hidden layers

Could have skip connections

Or any other architecture

Unfolded RNN



Given a problem we need to decided what is “t” going to be, how much we want to unroll to derive inference.

Do we want look at past 20 frames, 100 frames or 200 frames to make a decision about a video. That will determine

Number in the sequence. This is pre-decided before the training…

Recurrent Neural Network

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

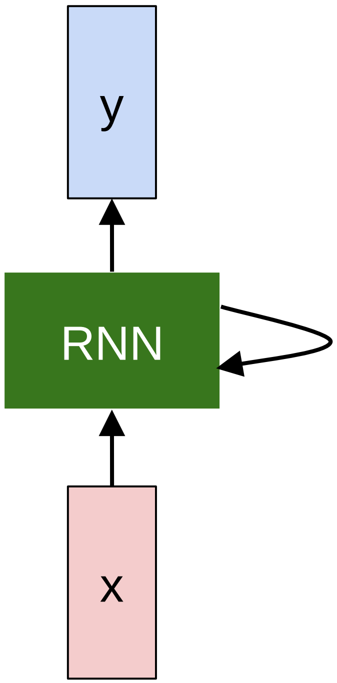
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

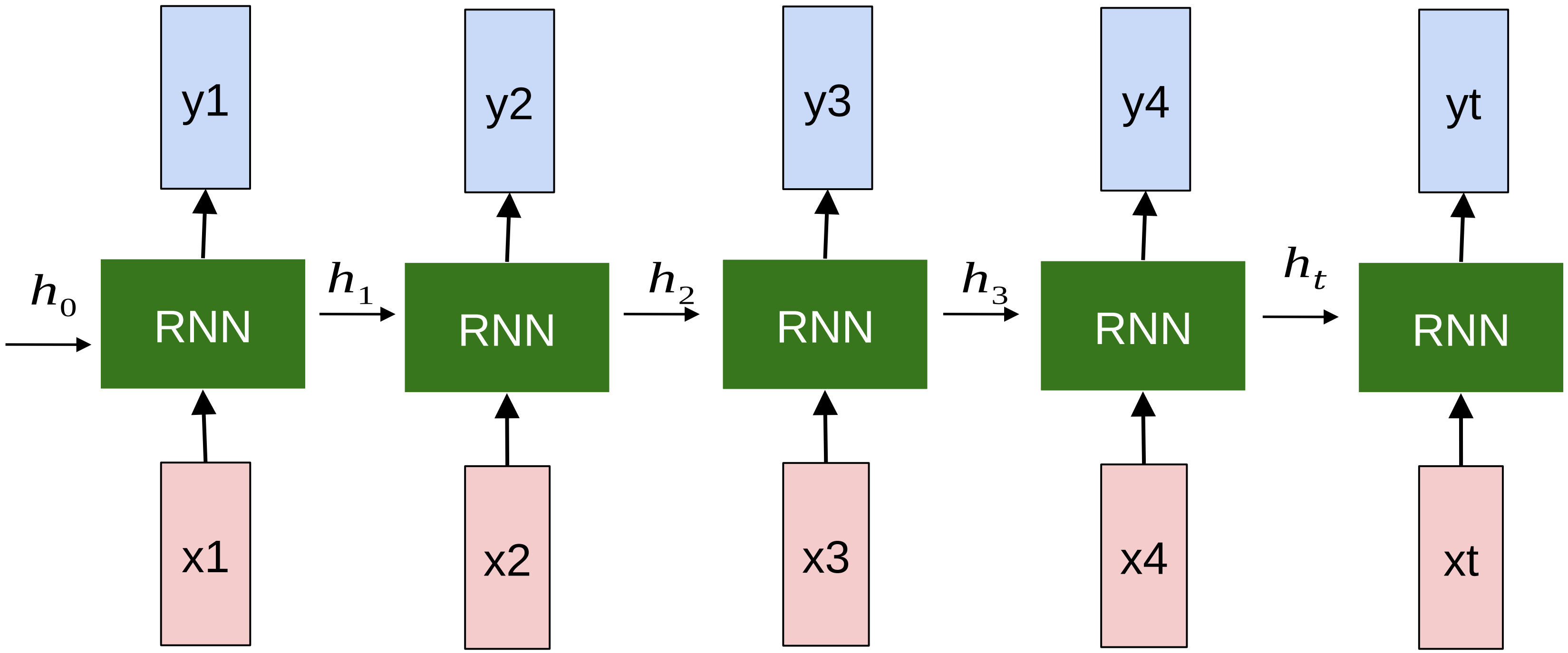
old state

input vector at some time step



W=U, V the sets of parameters in the model

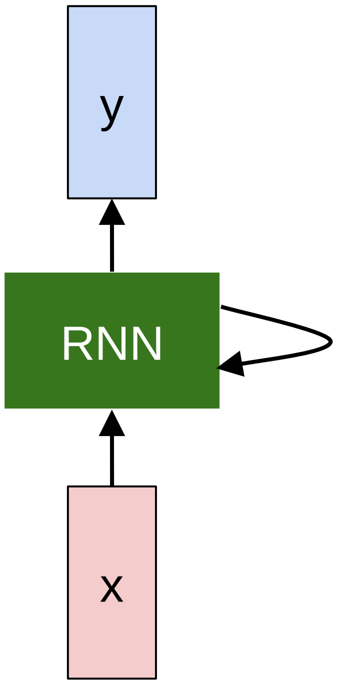
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n


y1
y2
y3
y4
yt
h_0
RNN
h_1
RNN
h_2
RNN
h_3
RNN
h_t
RNN
x1
x2
x3
x4
xt

Recurrent Neural Network

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

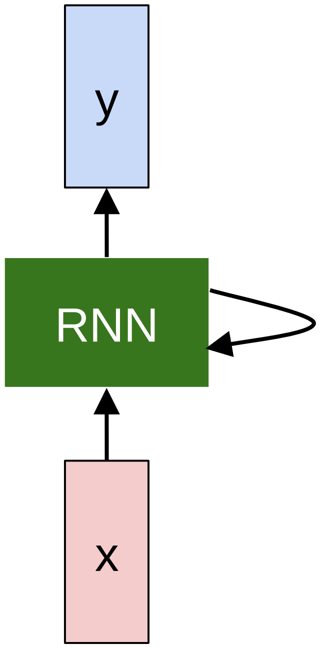
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

The state consists of a single *“hidden”* vector **h**:



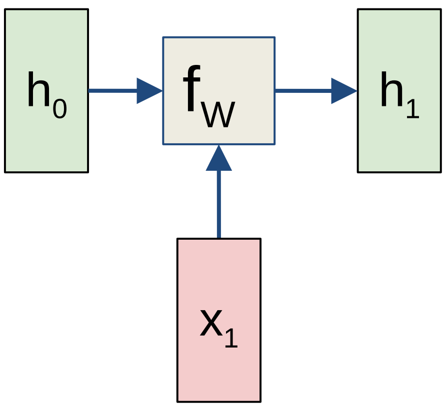
$$y_t = W_{hy} h_t + b_y$$

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

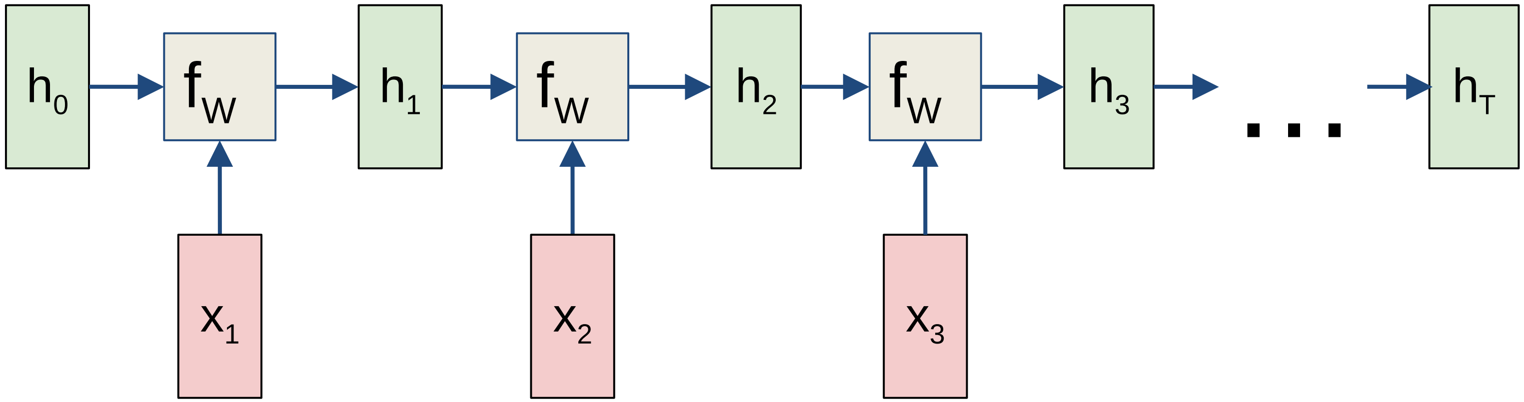
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

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

RNN: Computational Graph

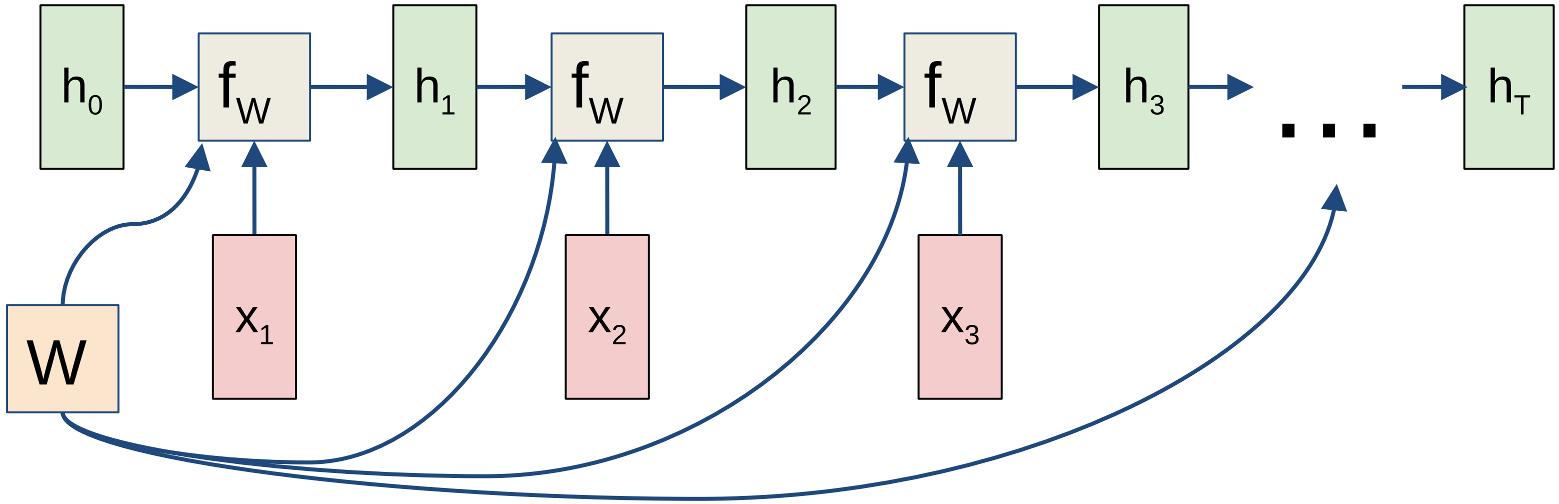


RNN: Computational Graph

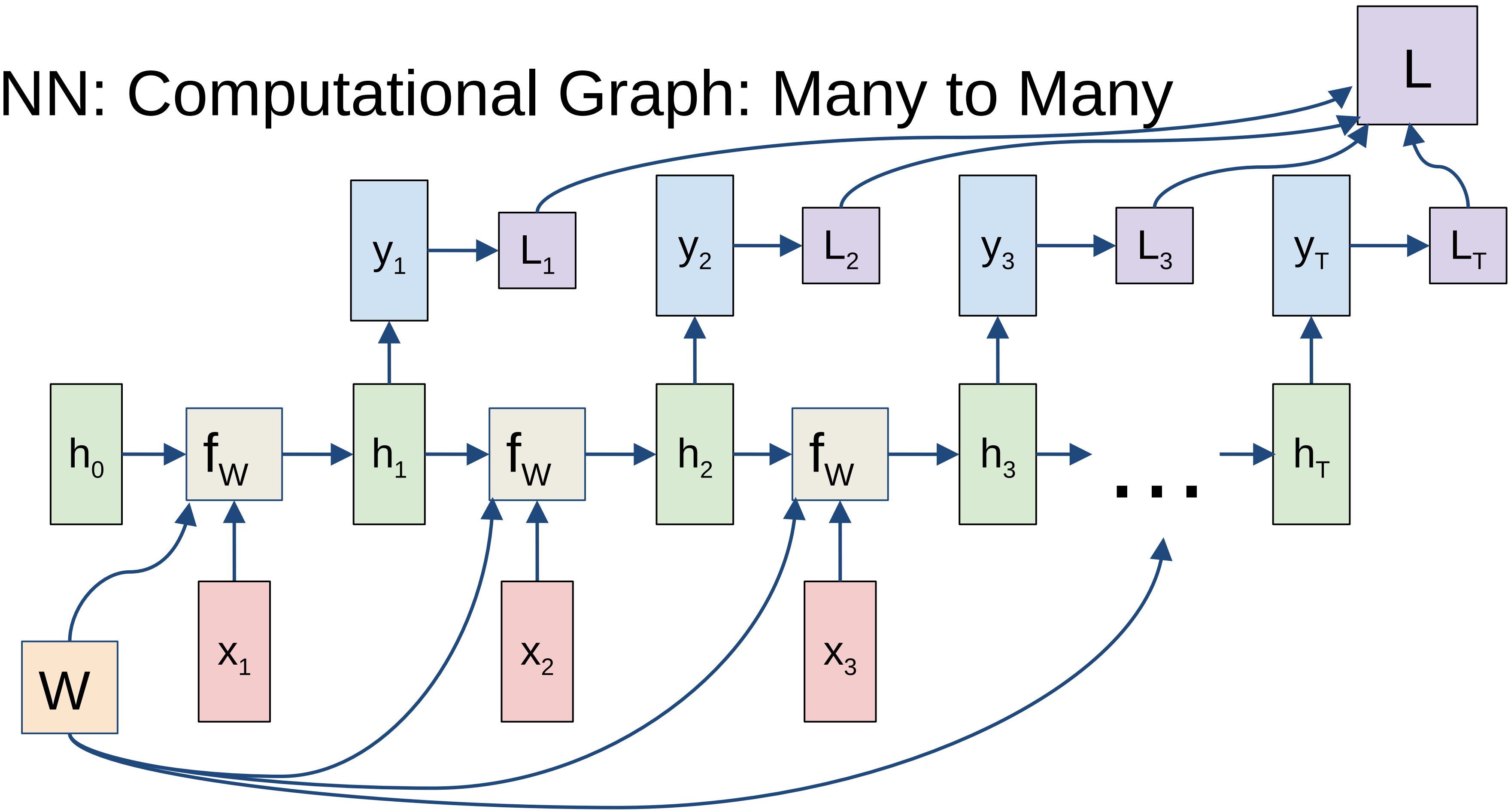


RNN: Computational Graph

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

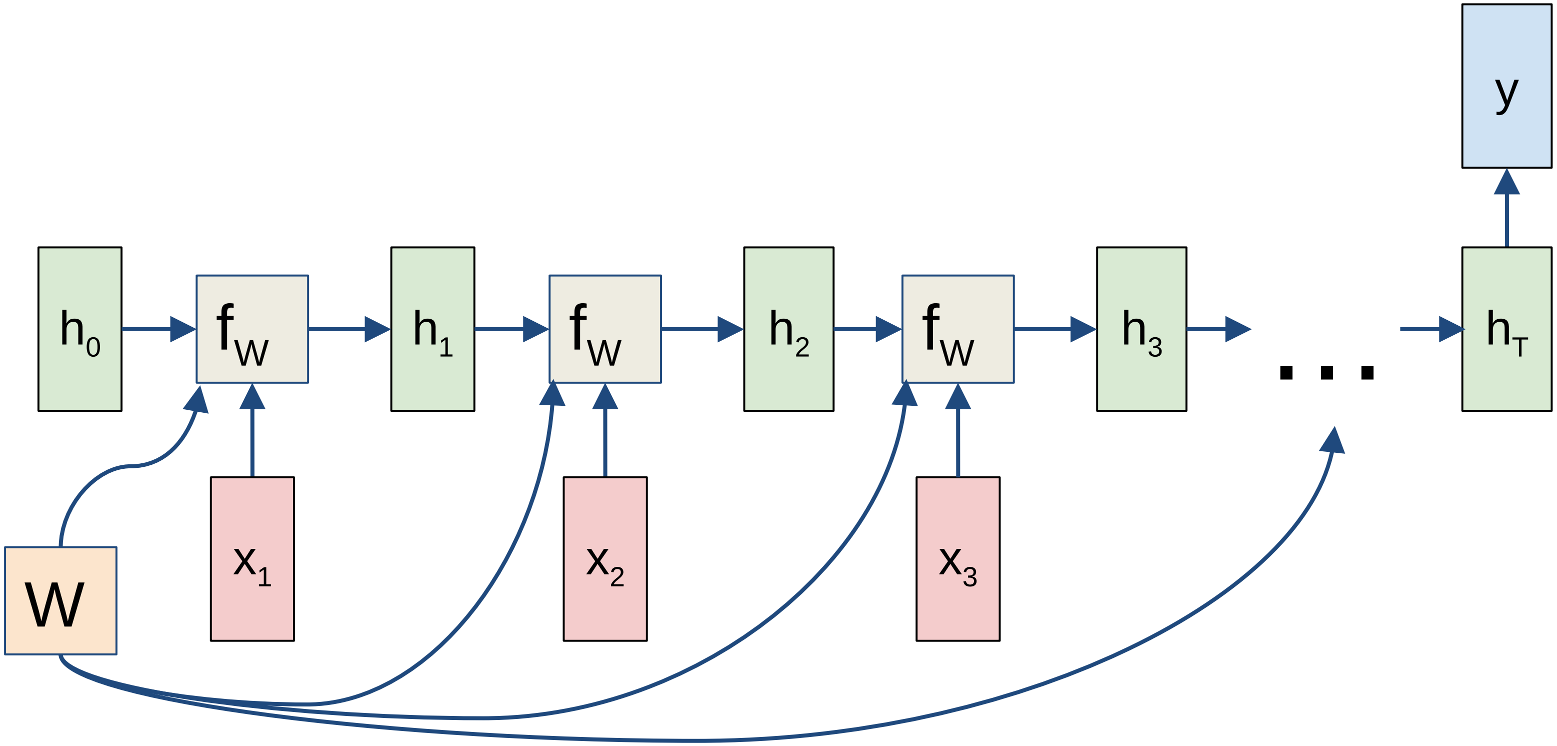


RNN: Computational Graph: Many to Many

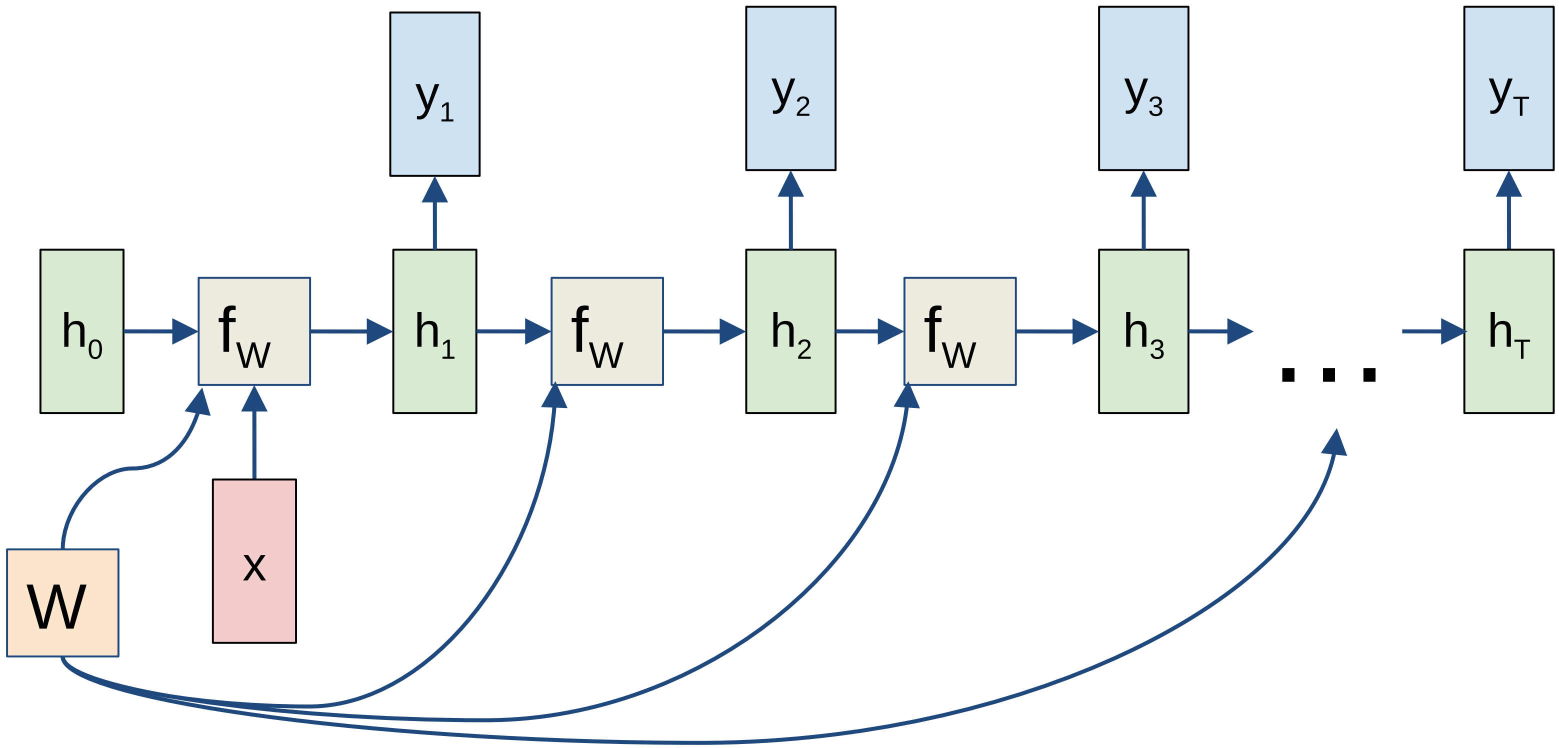


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

RNN: Computational Graph: Many to One



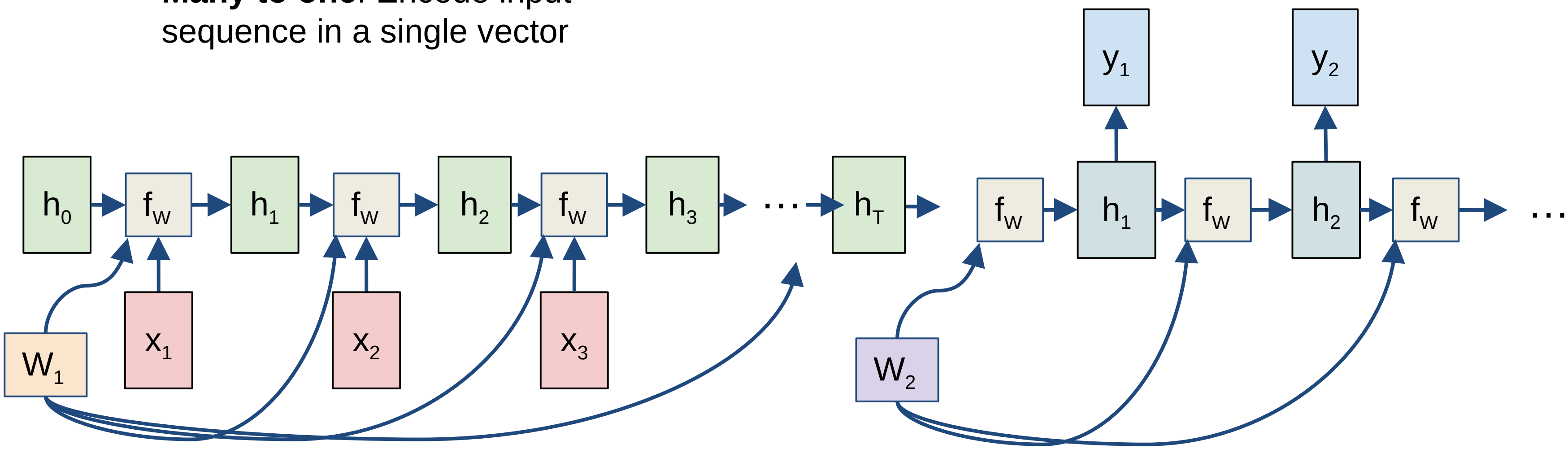
RNN: Computational Graph: One to Many



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

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



Computational Graph: Basics

Plan for Today

- Recurrent Neural Networks (RNNs)
  - Example Problem: (Character-level) Language modeling
  - Learning: (Truncated) BackProp Through Time (BPTT)
  - Visualizing RNNs
  - Example: Image Captioning
  - Inference: Beam Search
  - Multilayer RNNs
  - Problems with gradients in “vanilla” RNNs
  - LSTMs (and other RNN variants)

Language Modeling

- Given a dataset, build an accurate model: $P(y_1, y_2, \ldots y_T)$

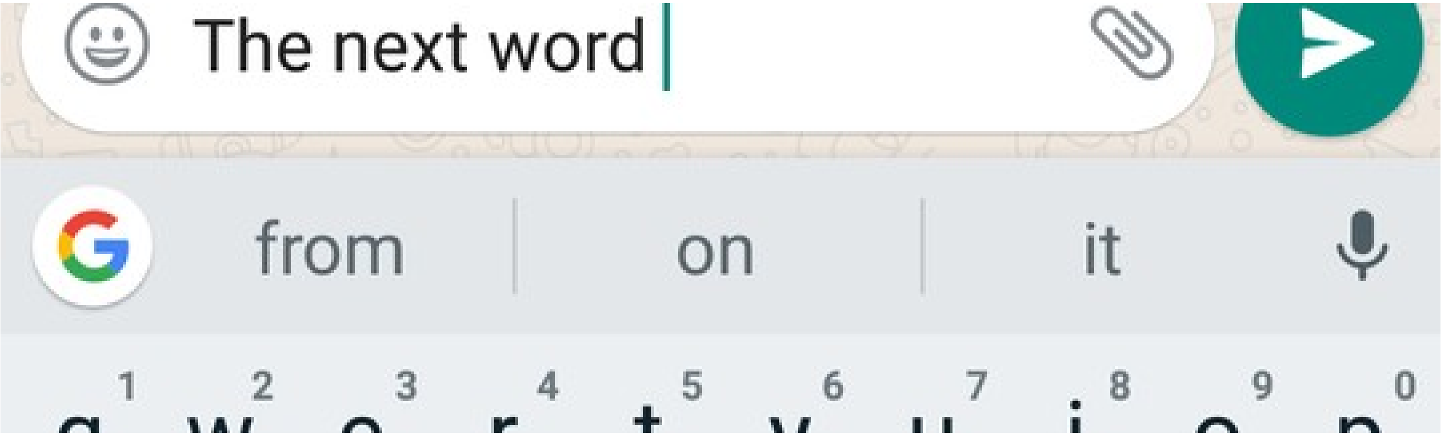
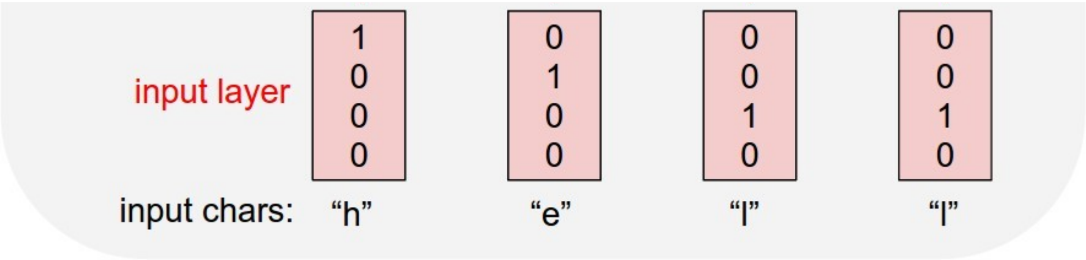


Image Credit: https://ofir.io/Neural-Language-Modeling-From-Scratch/

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

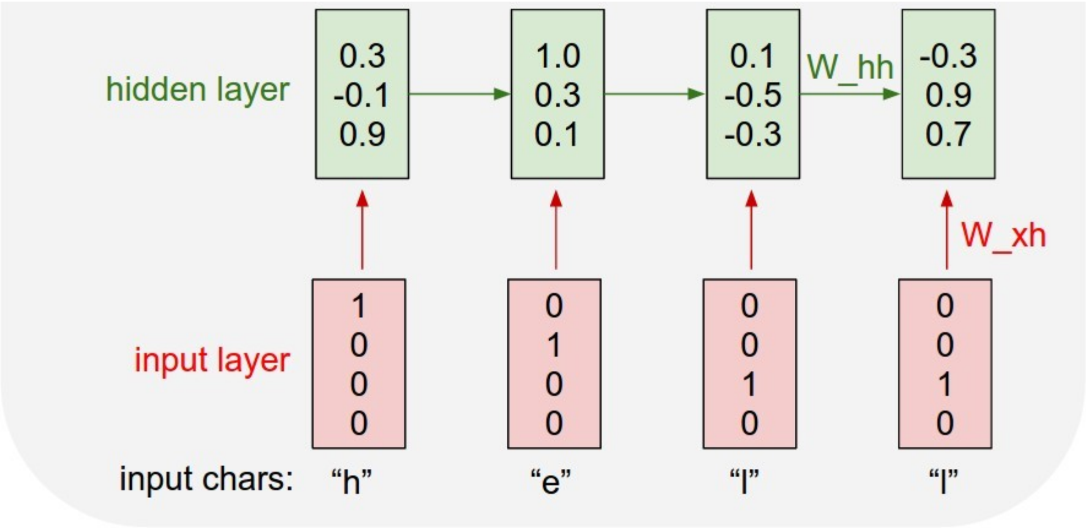


Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

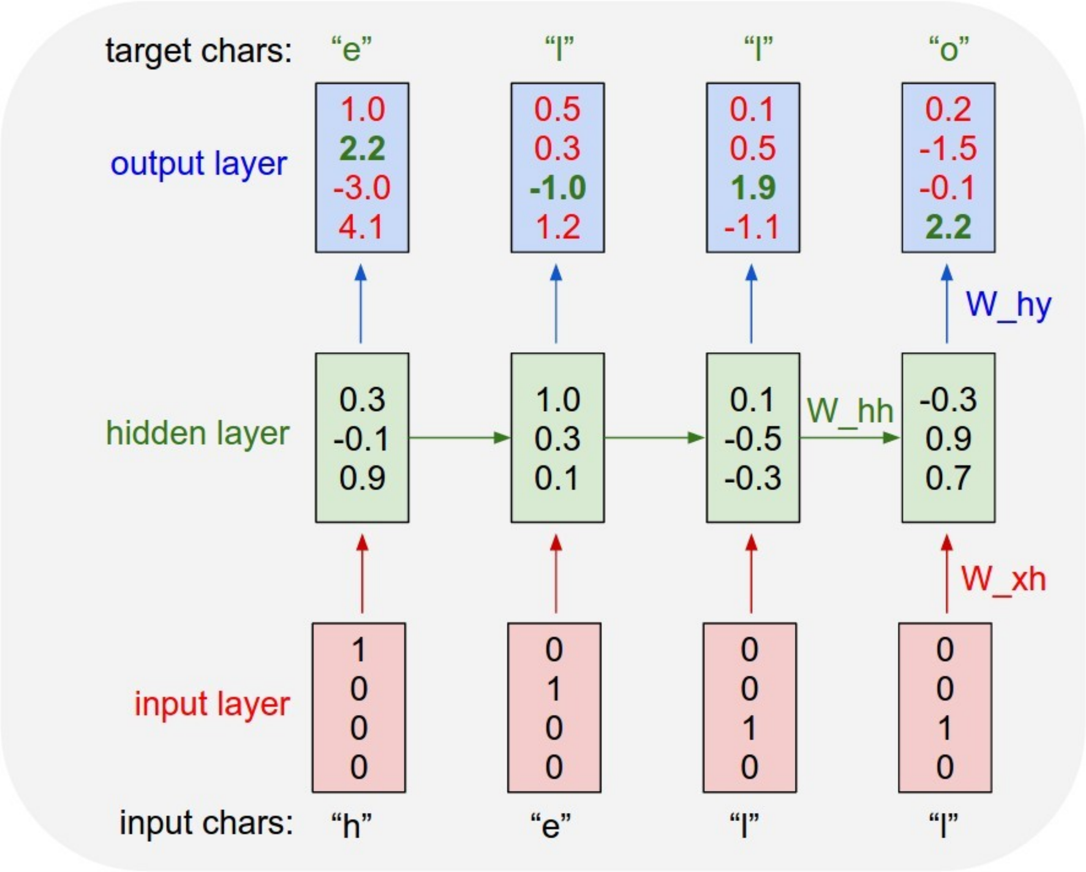
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

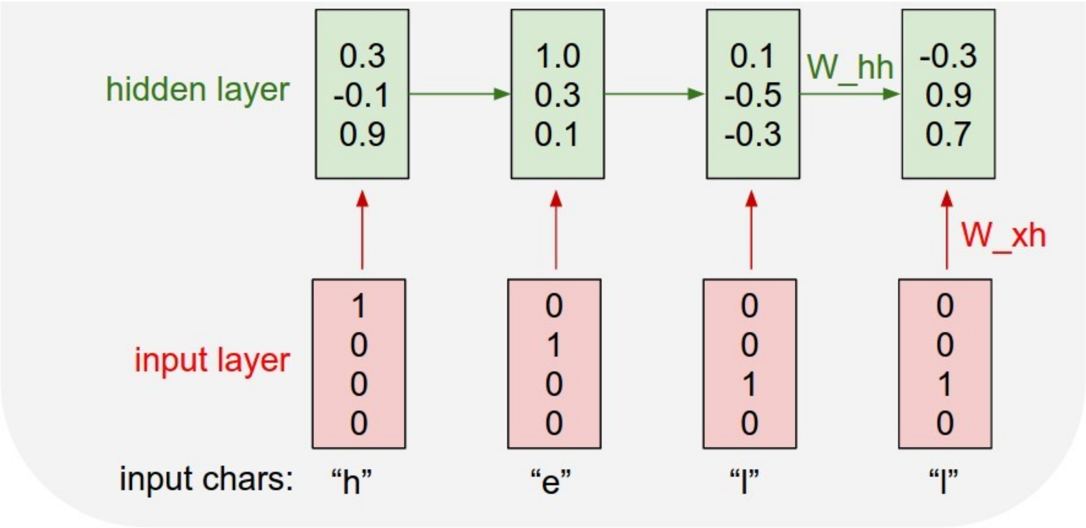


Example: Character-level Language Model

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

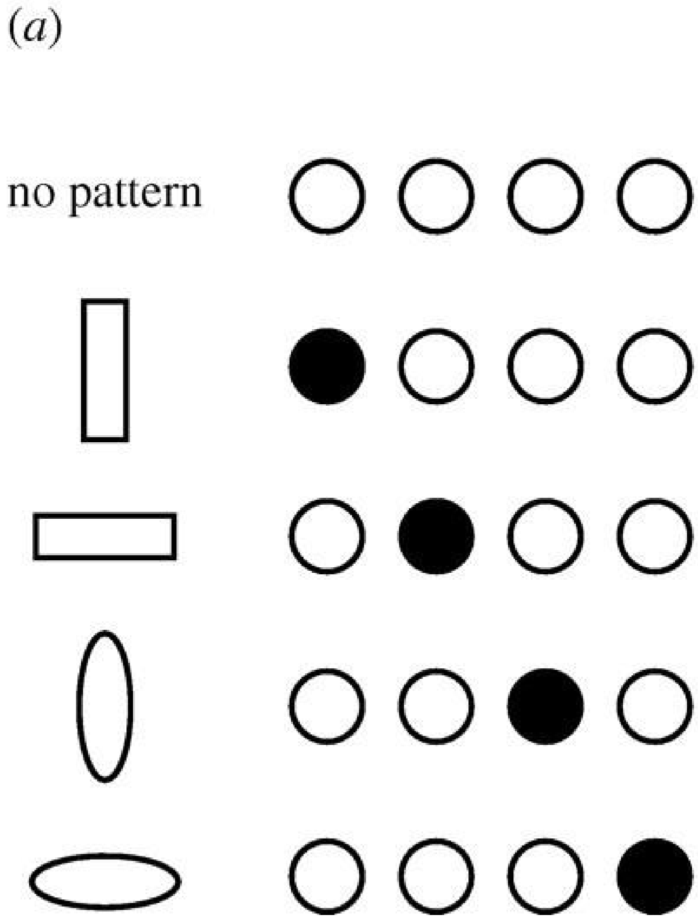
Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**



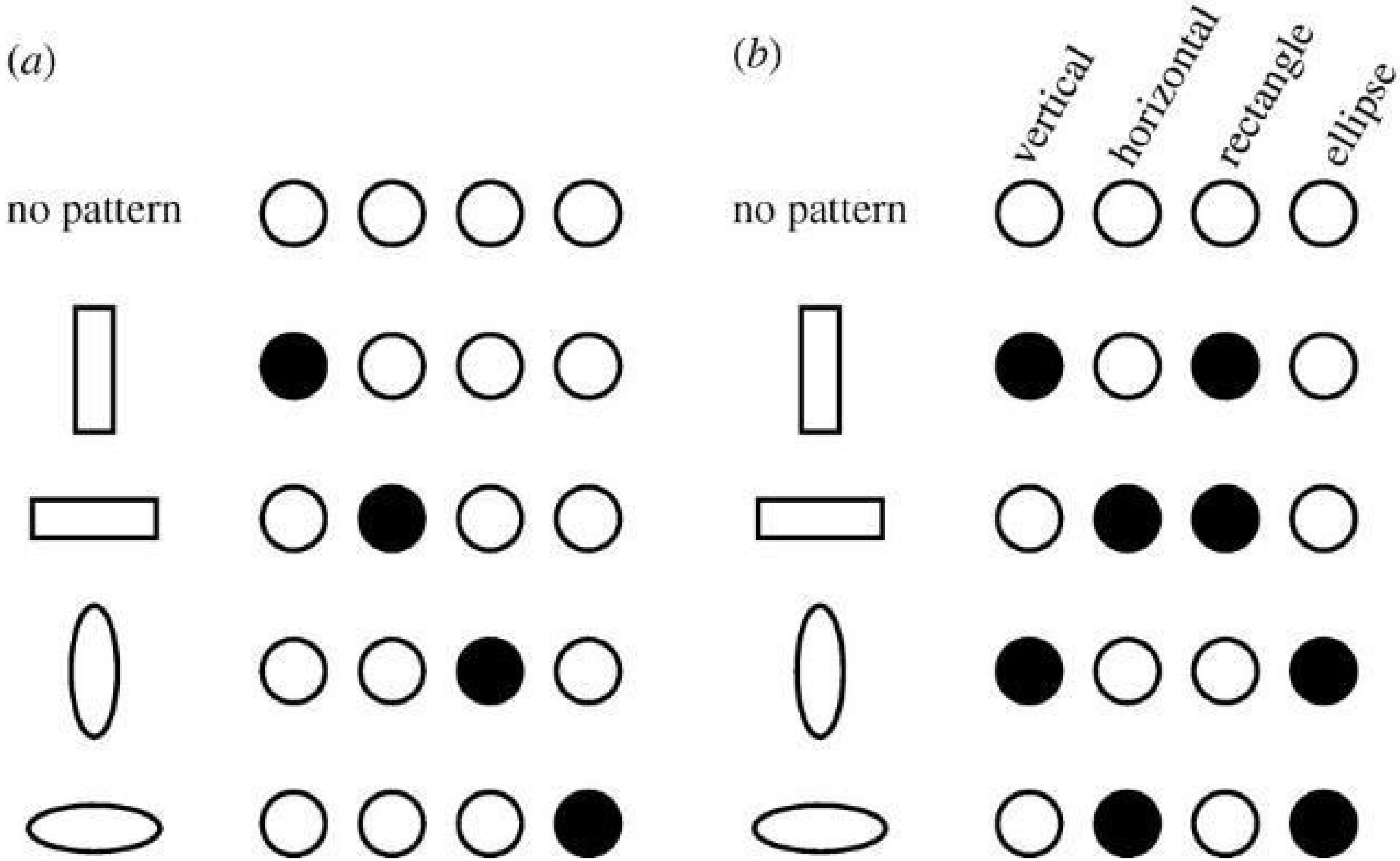
Distributed Representations Toy Example

- Local vs Distributed



Distributed Representations Toy Example

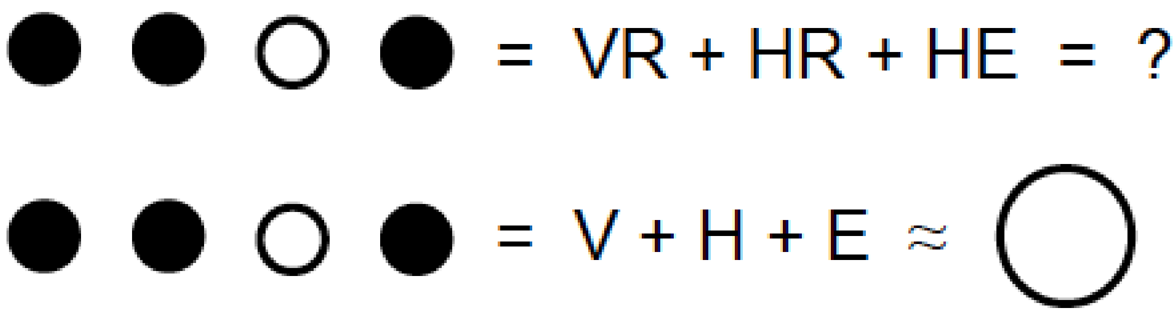
- Can we interpret each dimension?



Power of distributed representations!

Local

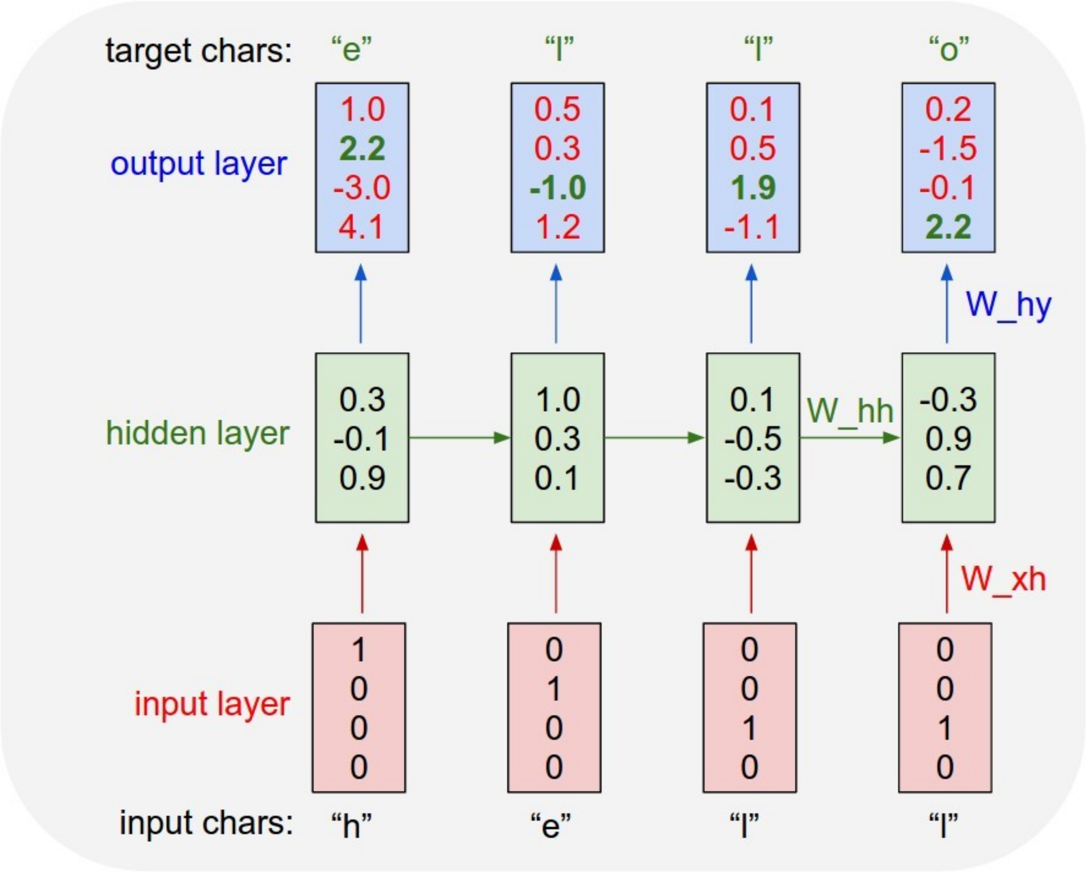
Distributed



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

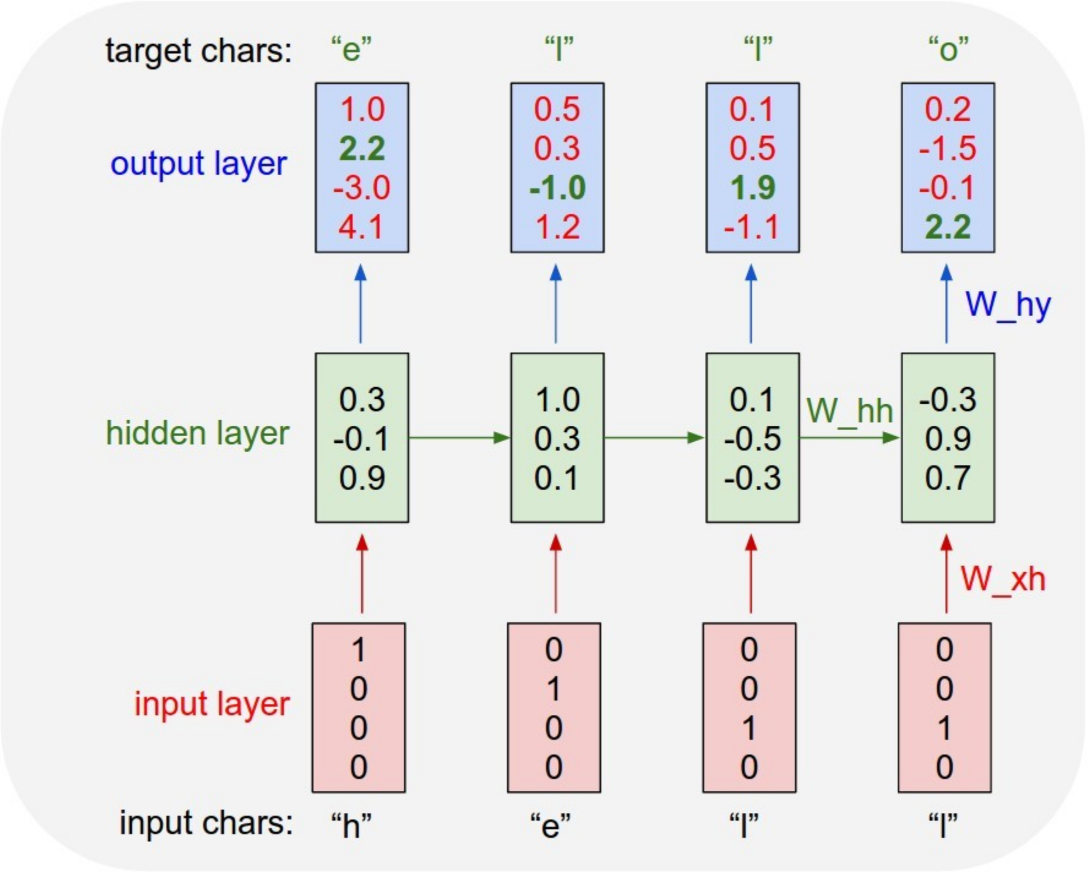


Training Time: MLE / “Teacher Forcing”

**Example:
Character-level
Language Model**

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

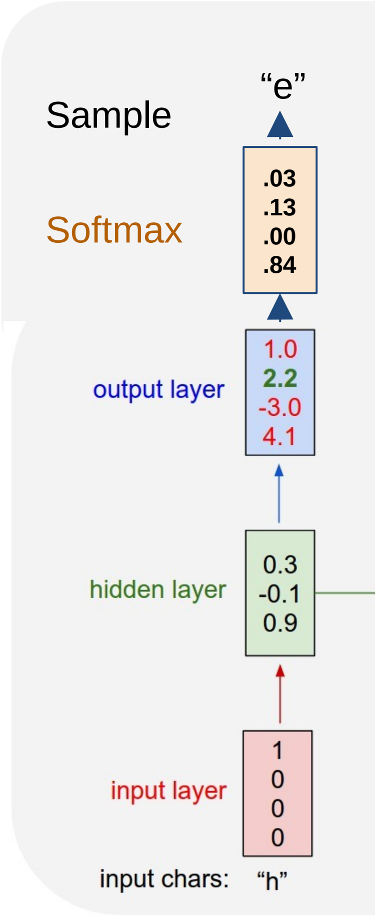


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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



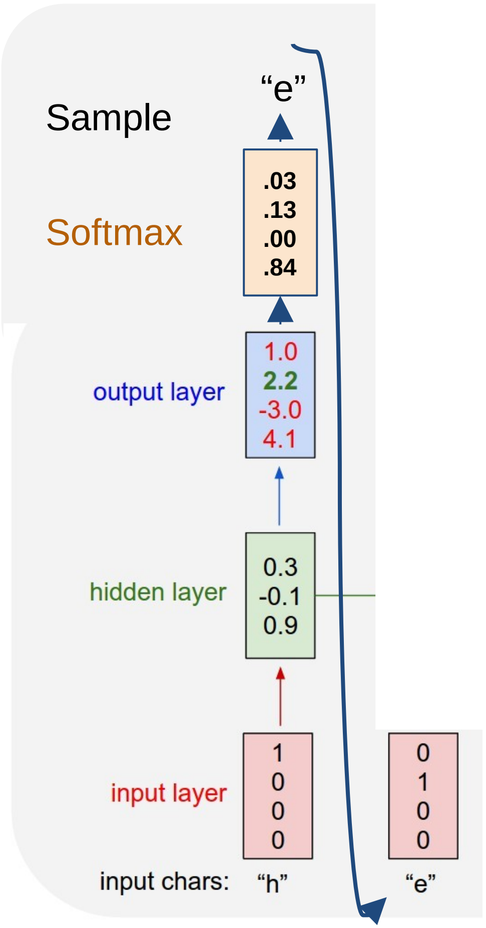
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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



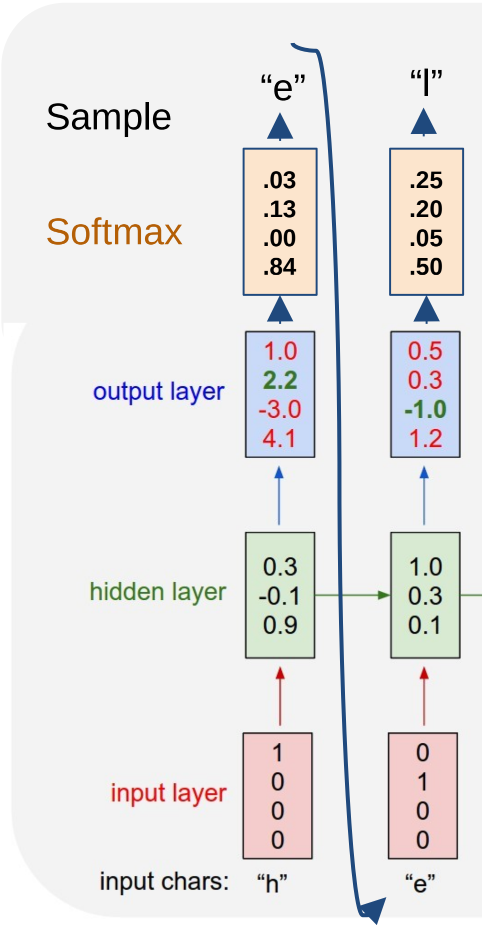
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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



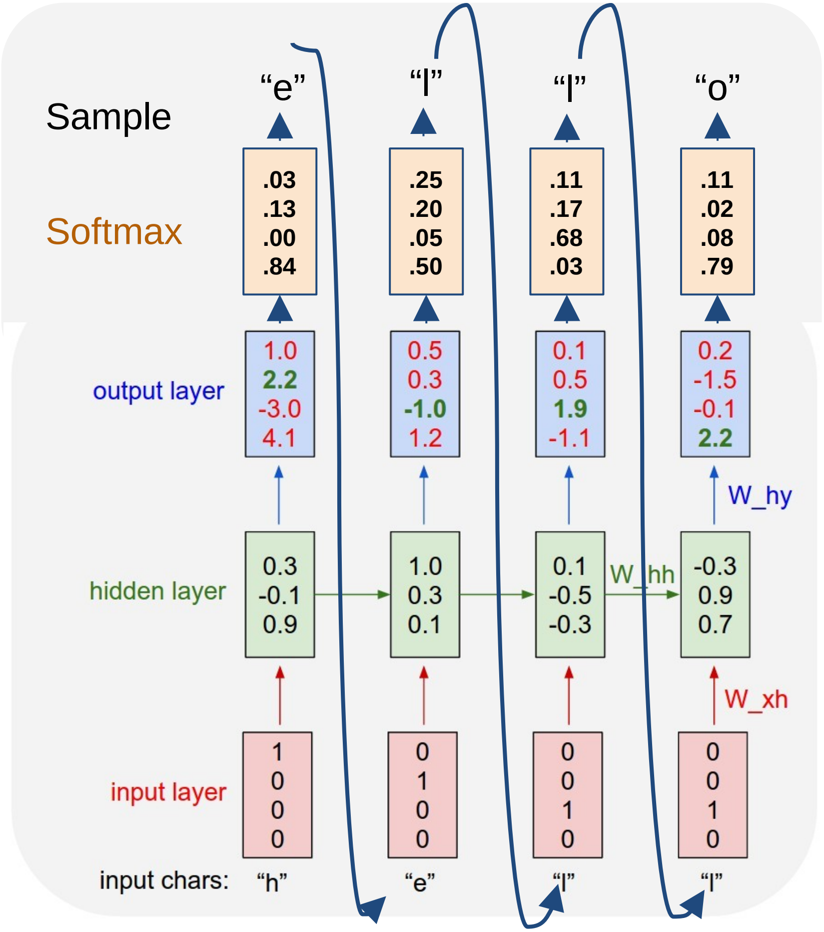
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

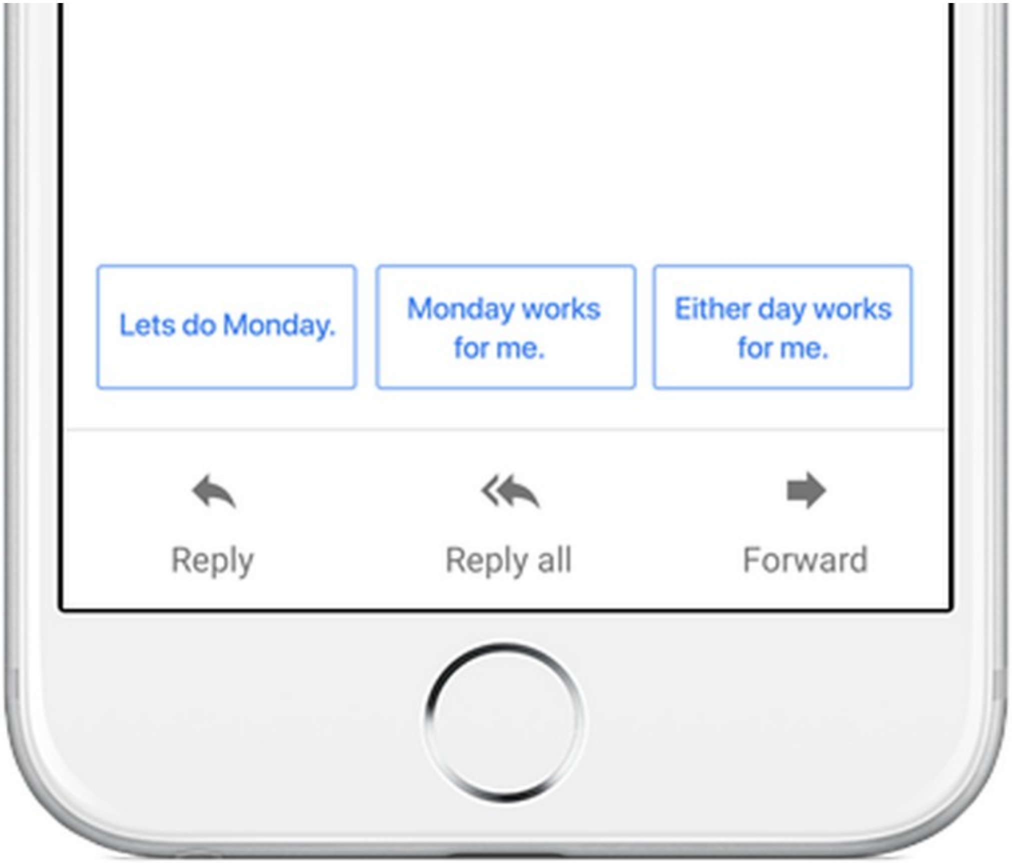
Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

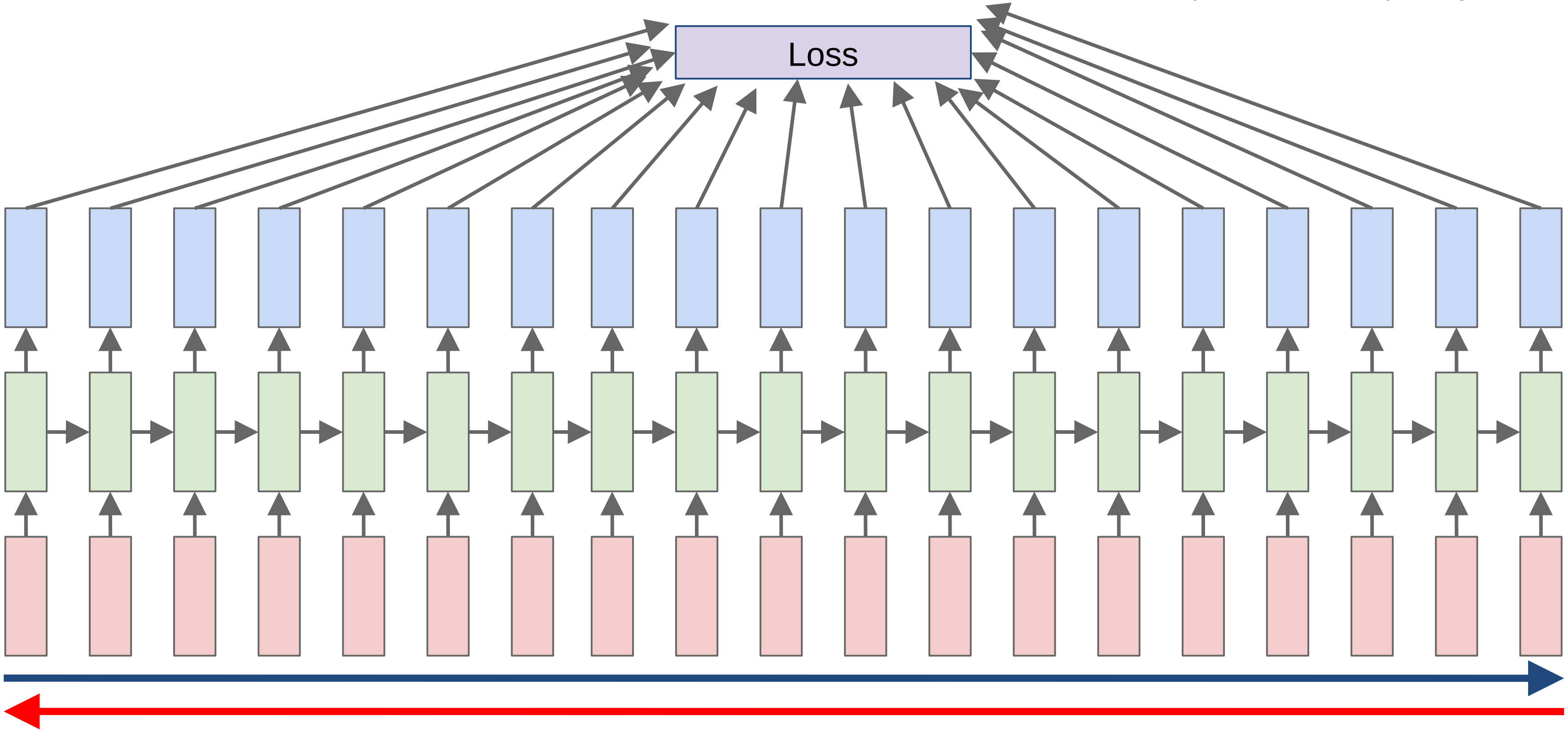
At test-time sample characters one at a time, feed back to model



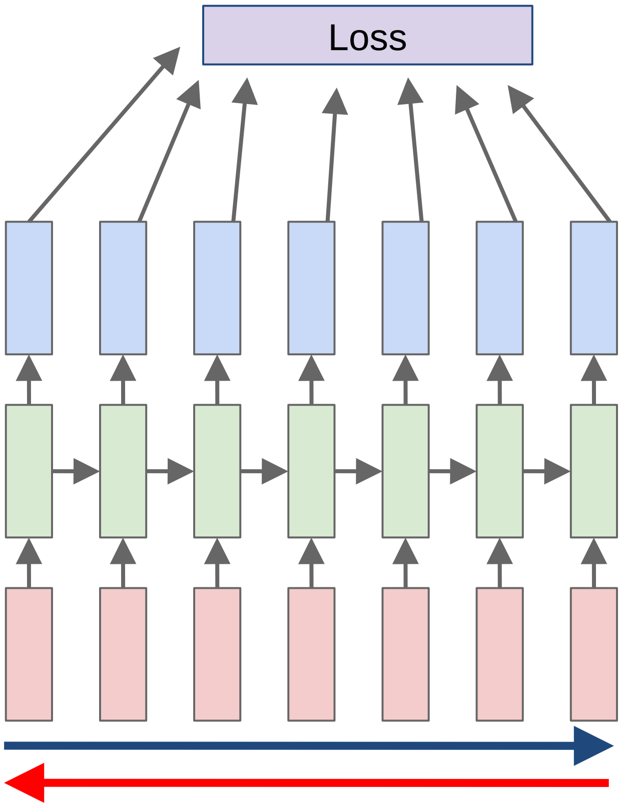

Lets do Monday.
Monday works for me.
Either day works for me.
Reply
Reply all
Forward

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

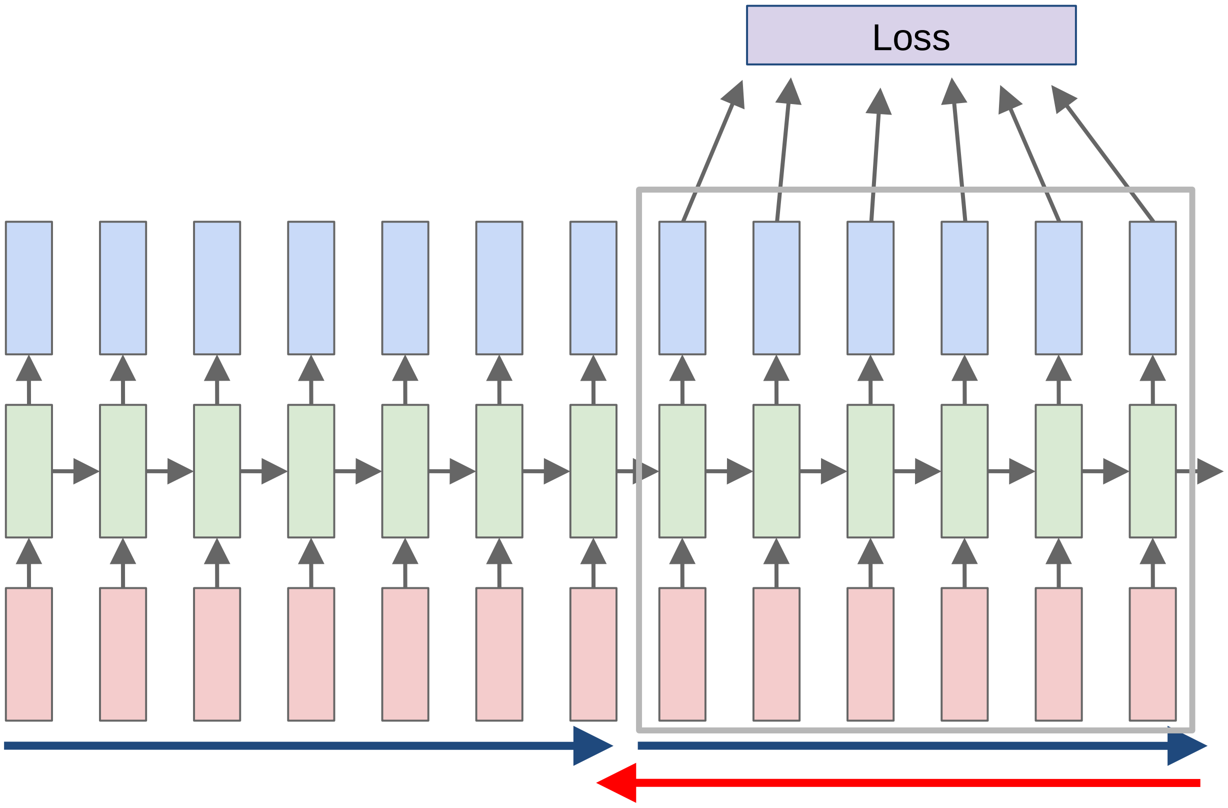


Truncated Backpropagation through time



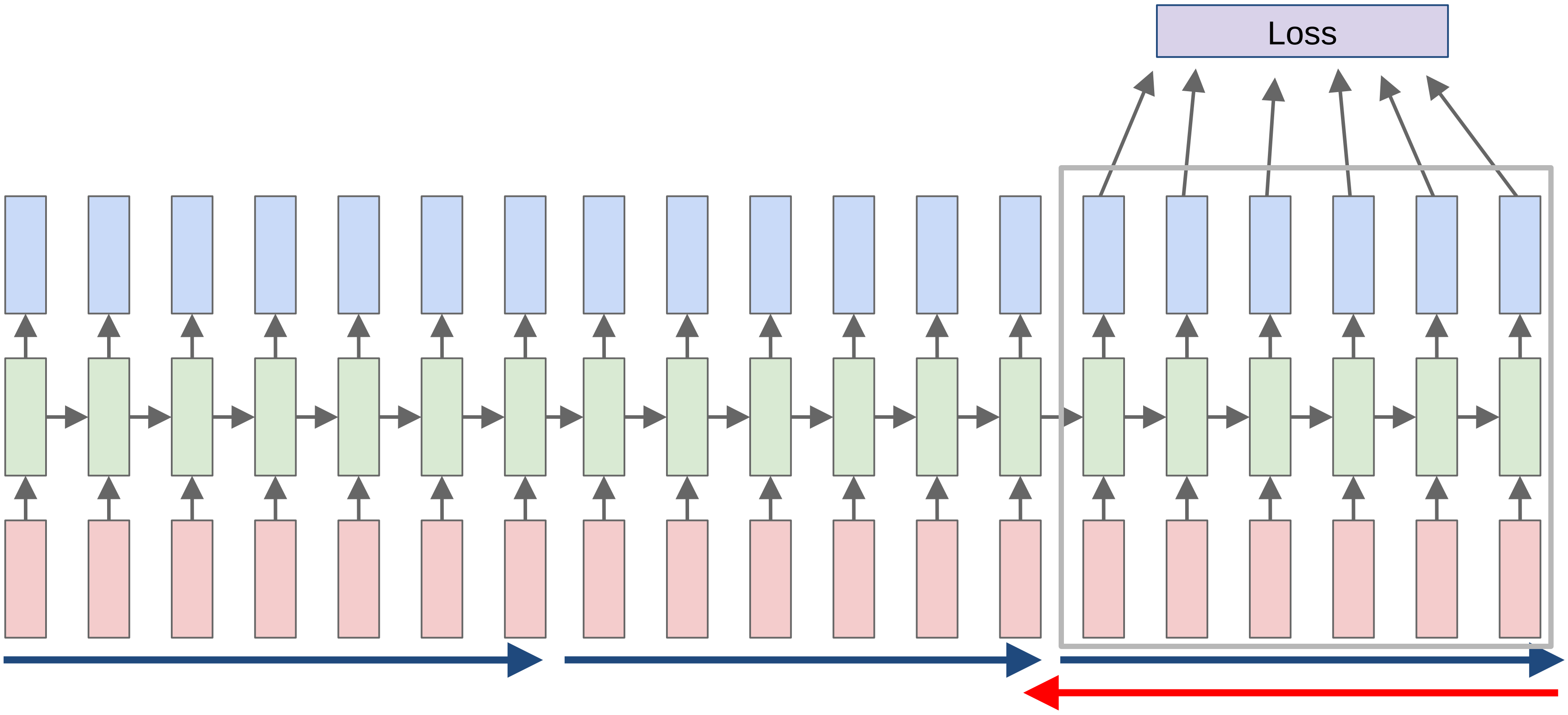
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



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

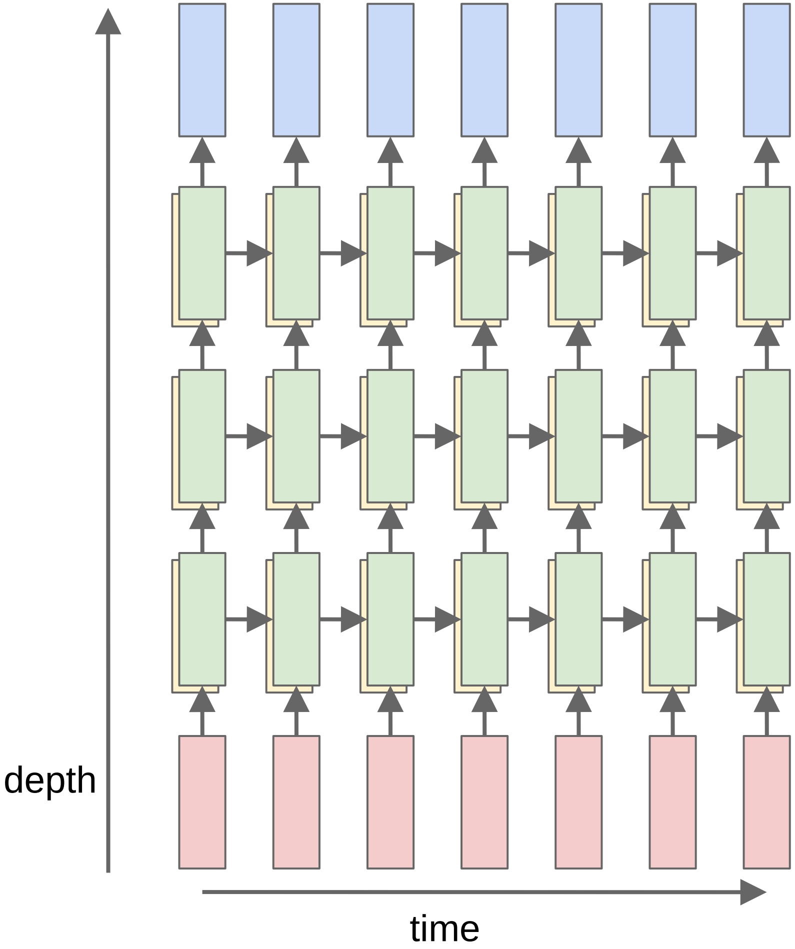
Plan for Today

- Recurrent Neural Networks (RNNs)
  - Example Problem: (Character-level) Language modeling
  - Learning: (Truncated) BackProp Through Time (BPTT)
  - Visualizing RNNs
  - Example: Image Captioning
  - Inference: Beam Search
  - Multilayer RNNs
  - Problems with gradients in “vanilla” RNNs
  - LSTMs (and other RNN variants)

Multilayer RNNs

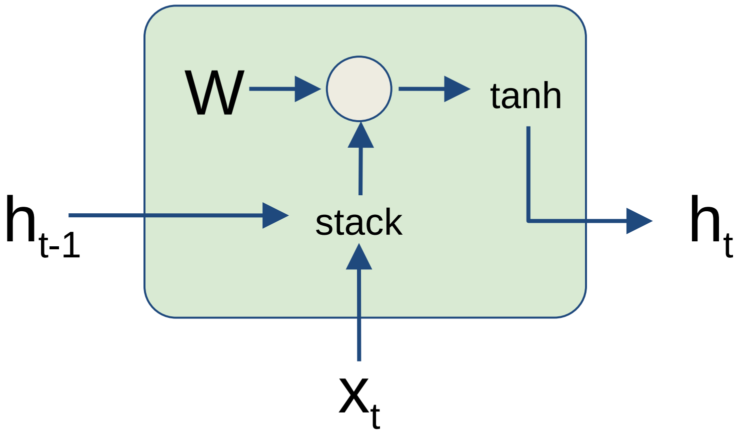
$$\boxed{h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}}$$

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



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

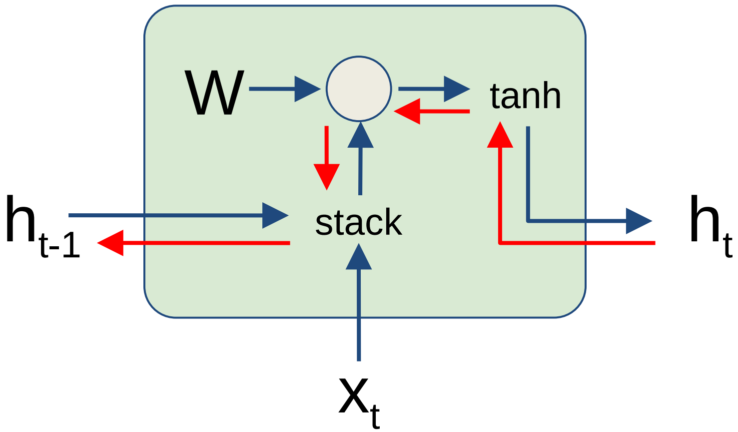


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

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

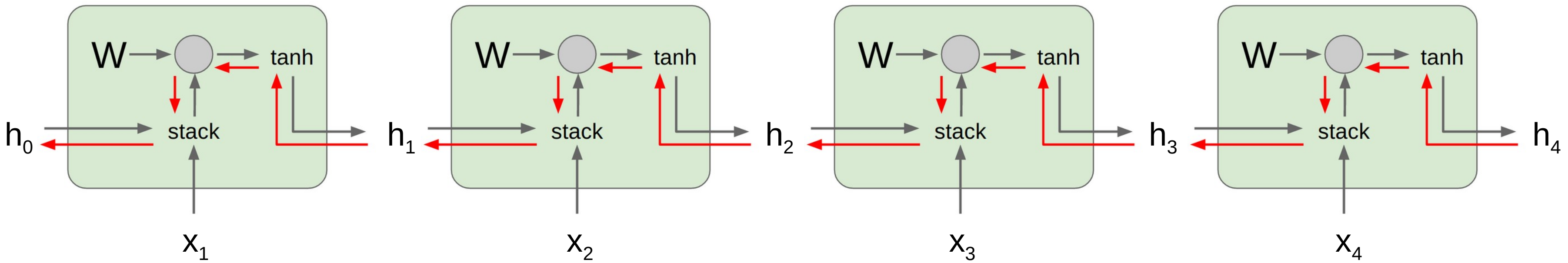
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh})



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

Vanilla RNN Gradient Flow

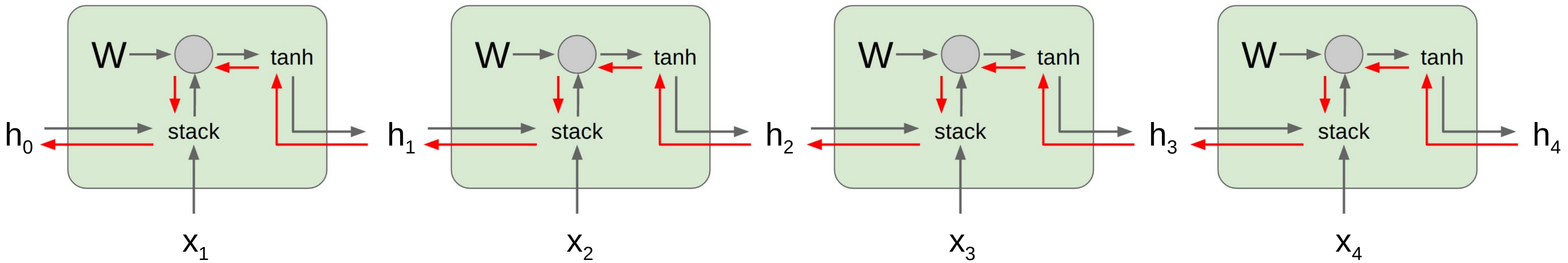
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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



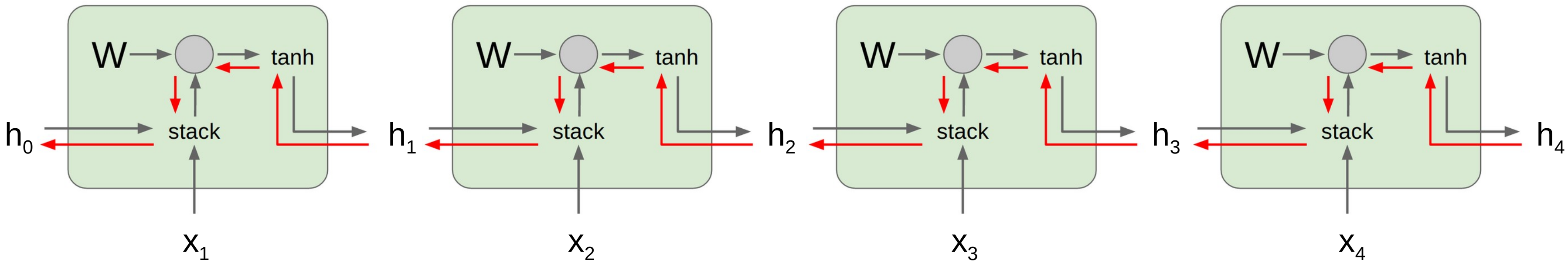
Computing gradient of h_0 involves many factors of W (and repeated tanh)

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



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

Largest singular value > 1:
Exploding gradients

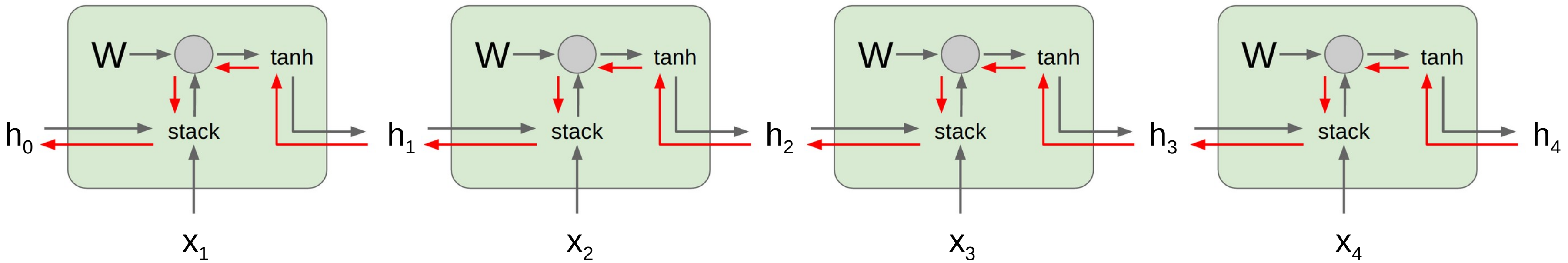
Largest singular value < 1:
Vanishing gradients

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

How to tackle vanishing gradients

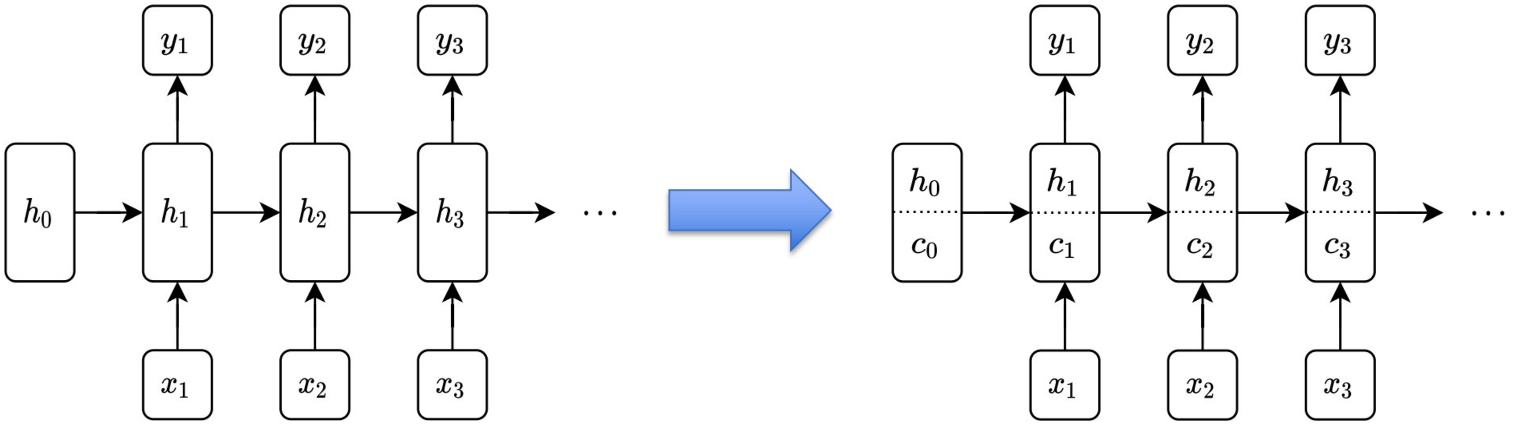
1. Use ReLU, instead of sigmoid and tanh. ReLU does not vanish the gradient, it keeps the gradient positive.
2. Regularization.
3. Better initialization of weights.
4. Use only short time sequences (Truncated BacProp)

How to overcome limitations of RNN, by changing the architecture?

Long short term memory RNNs

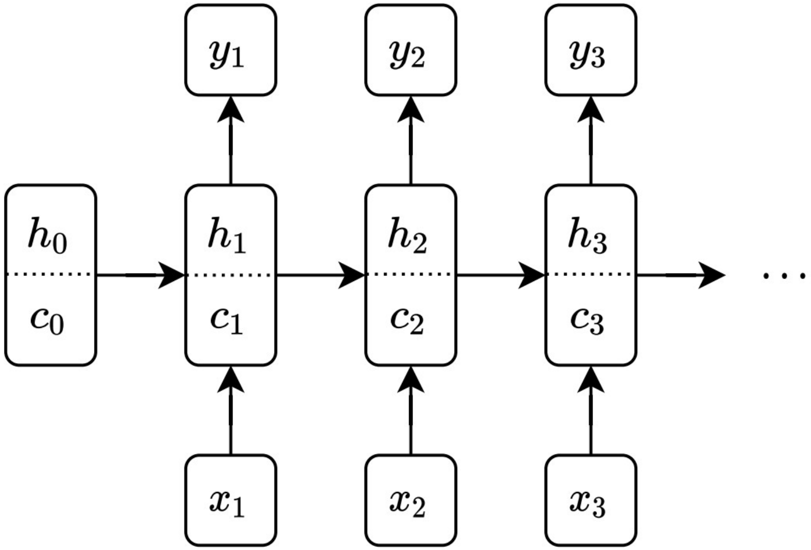
Long short term memory (LSTM) cells are a particular form of hidden unit update that avoids (some of) the problems of vanilla LSTMs

Step 1: Divide the hidden unit into two components, called (confusingly) the *hidden state* and the *cell state*



Long short term memory RNNs

Step 2: Use a very specific formula to update the hidden state and cell state (throwing in some other names, like “forget gate”, “input gate”, “output gate” for good measure)



$$\begin{bmatrix} i_t \\ f_t \\ g_t \\ o_t \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \tanh \\ \text{sigmoid} \end{pmatrix} (W_{hh} h_{t-1} + \mathrm{W}_{hx} x_t + b_h)$$

$$c_t = c_{t-1} \circ f_t + i_t \circ g_t$$

$$h_t = \tanh(c_t) \circ o_t$$

?????

Some famous LSTMs

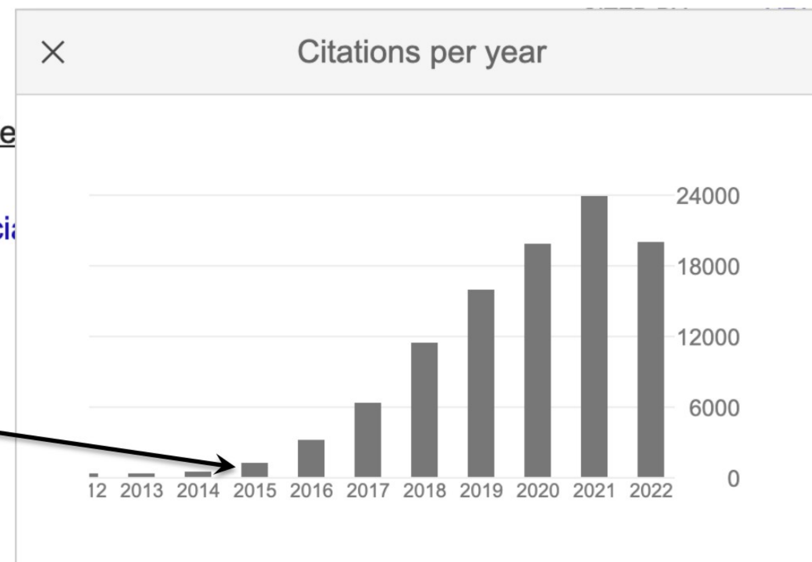
A notably famous blog post in the history of LSTMs:
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sepp Hochreiter

Institute for Machine Learning, Johannes Ke

Verified email at ml.jku.at - Homepage

Machine Learning   Deep Learning   Artifici



Andrej's blog post

Long Short Term Memory (LSTM)

Vanilla RNN

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

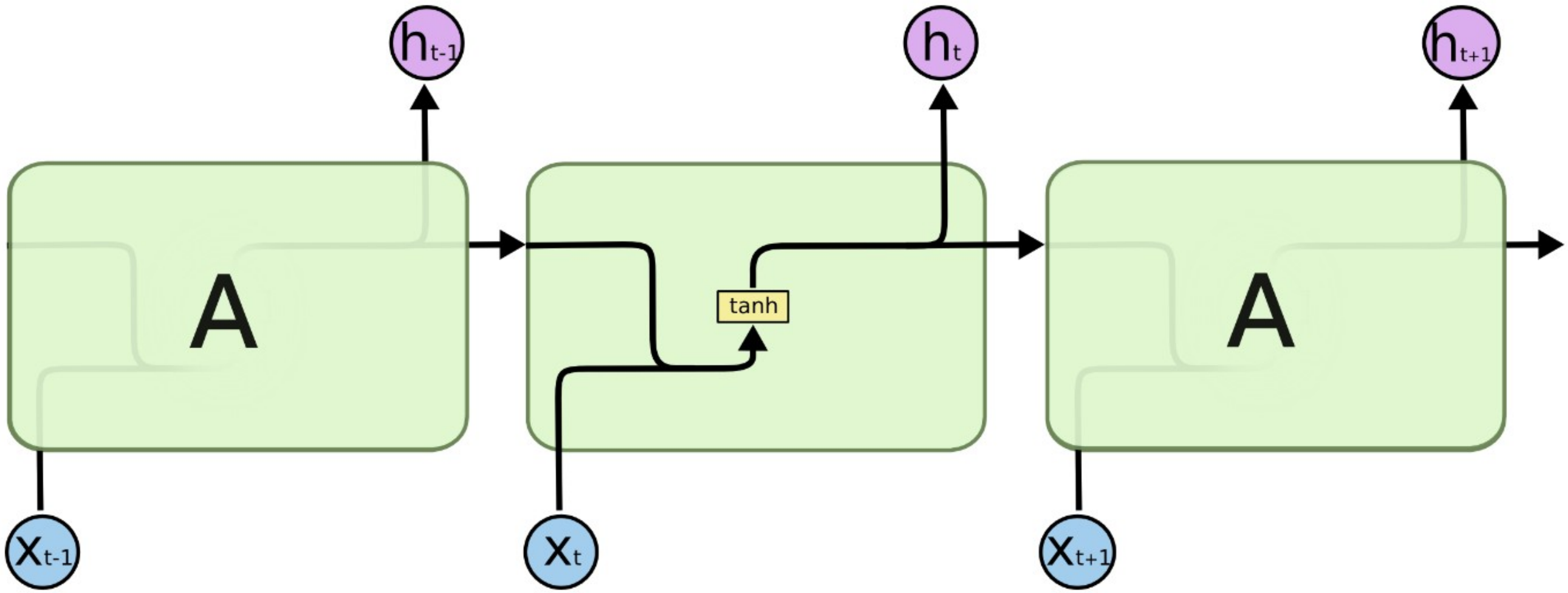
LSTM

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

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

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

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



The repeating module in a standard RNN contains a single layer.

Understanding LSTM Networks -- colah's blog

Why do LSTMs work?

There have been a seemingly infinite number of papers / blog posts about "understanding how LSTMs work" (I find most of them rather unhelpful)

$$\begin{bmatrix} i_t \\ f_t \\ g_t \\ o_t \end{bmatrix} = \begin{pmatrix} \mathrm{sigmoid} \\ \mathrm{sigmoid} \\ \tanh \\ \mathrm{sigmoid} \end{pmatrix} (\mathrm{W}_{hh} h_{t-1} + \mathrm{W}_{hx} x_t + b_h)$$

$$\boxed{c_t = c_{t-1} \circ f_t + i_t \circ g_t}$$

$$h_t = \tanh(c_t) \circ o_t$$

The key is this line here:

- We form c_t by scaling down c_{t-1} (remember, f_t is in $[0,1]^n$), then adding a term to it
- Importantly, "saturating" sigmoid activation for f_t at 1 would just pass through c_{t-1} untouched
- $\Longrightarrow$ For a wide(r) range of weights, LSTMs don't suffer vanishing gradients

Meet LSTMs

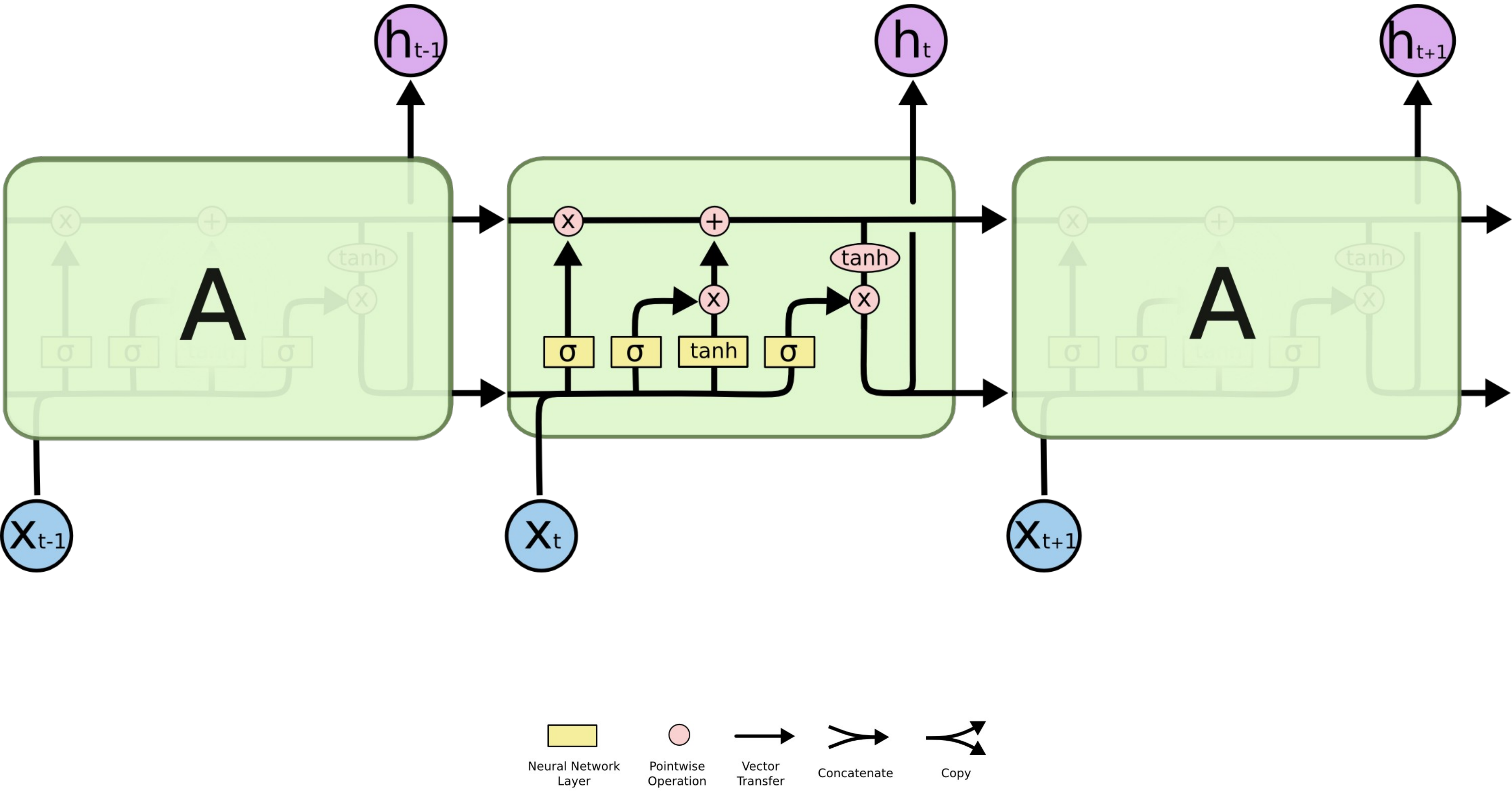


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Memory

- Cell State / Memory

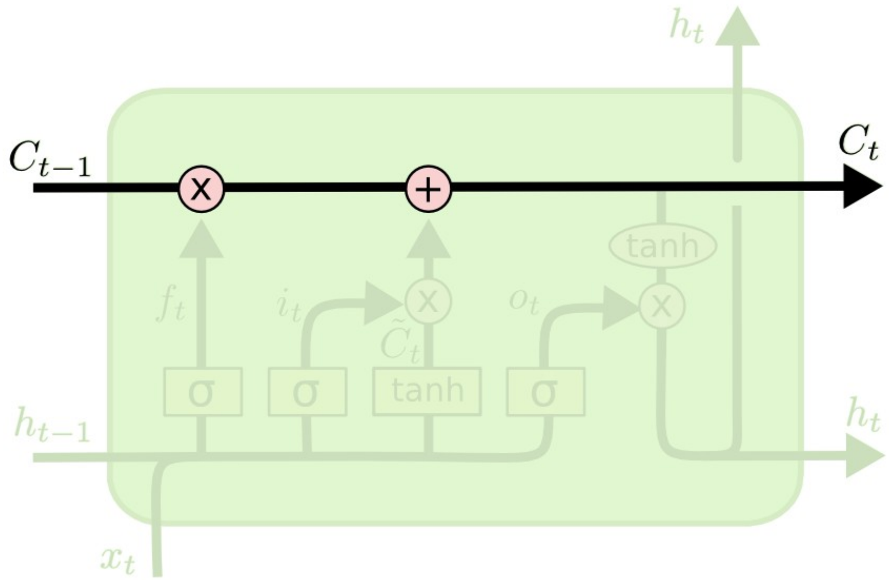
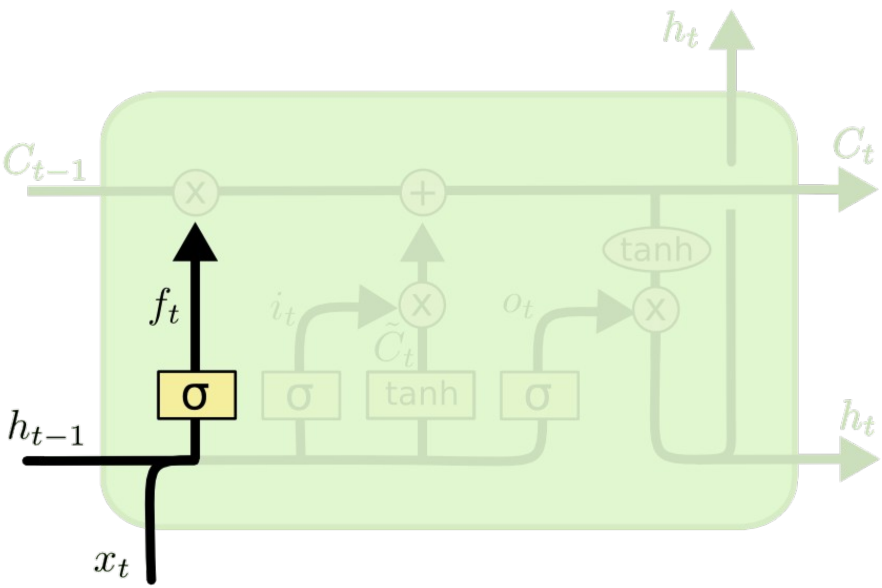


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Forget Gate

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

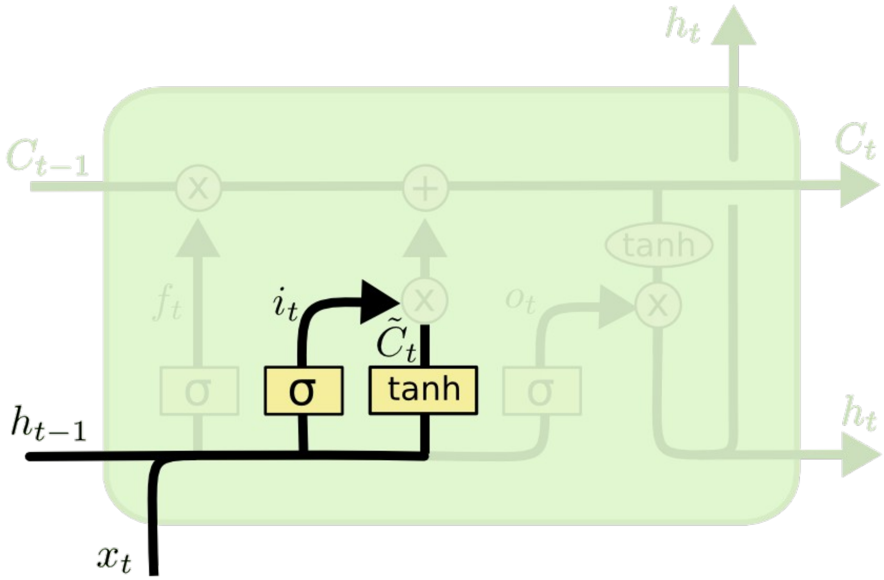


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

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?



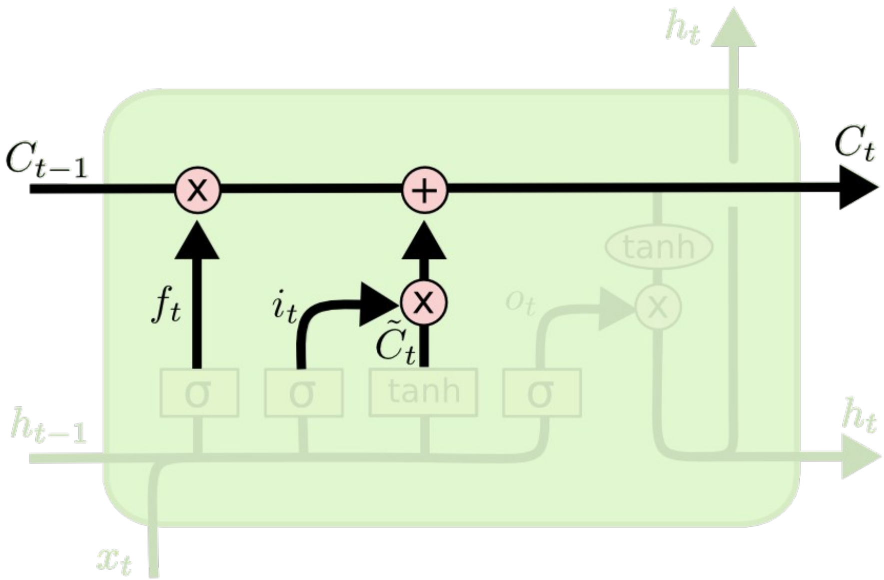
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

LSTMs Intuition: Memory Update

- Forget that + memorize this

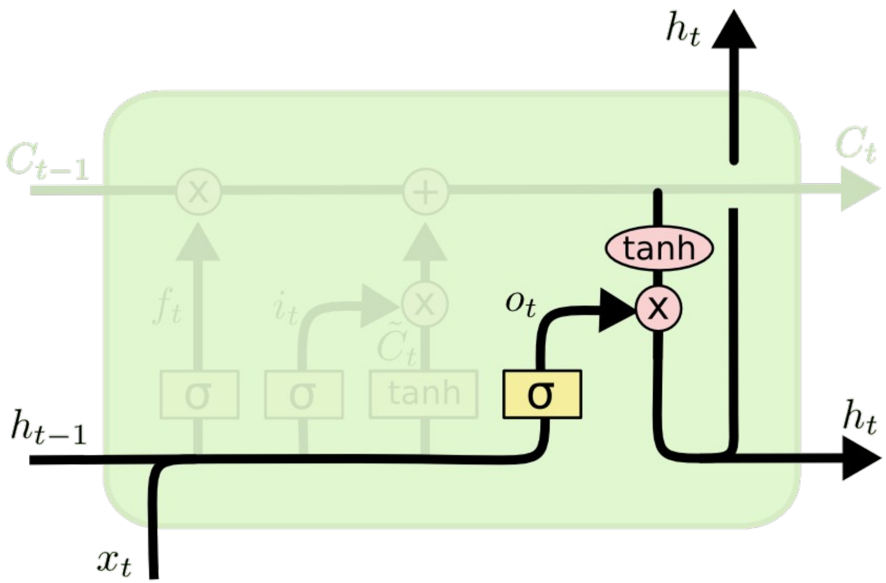


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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?



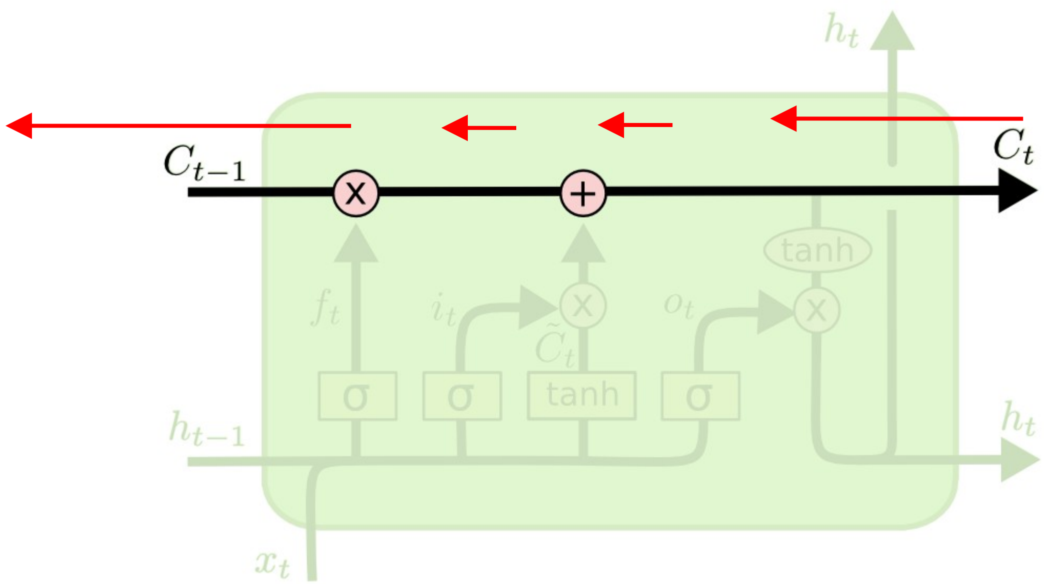
$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

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

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

LSTMs Intuition: Additive Updates

- Gradient Highway…



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

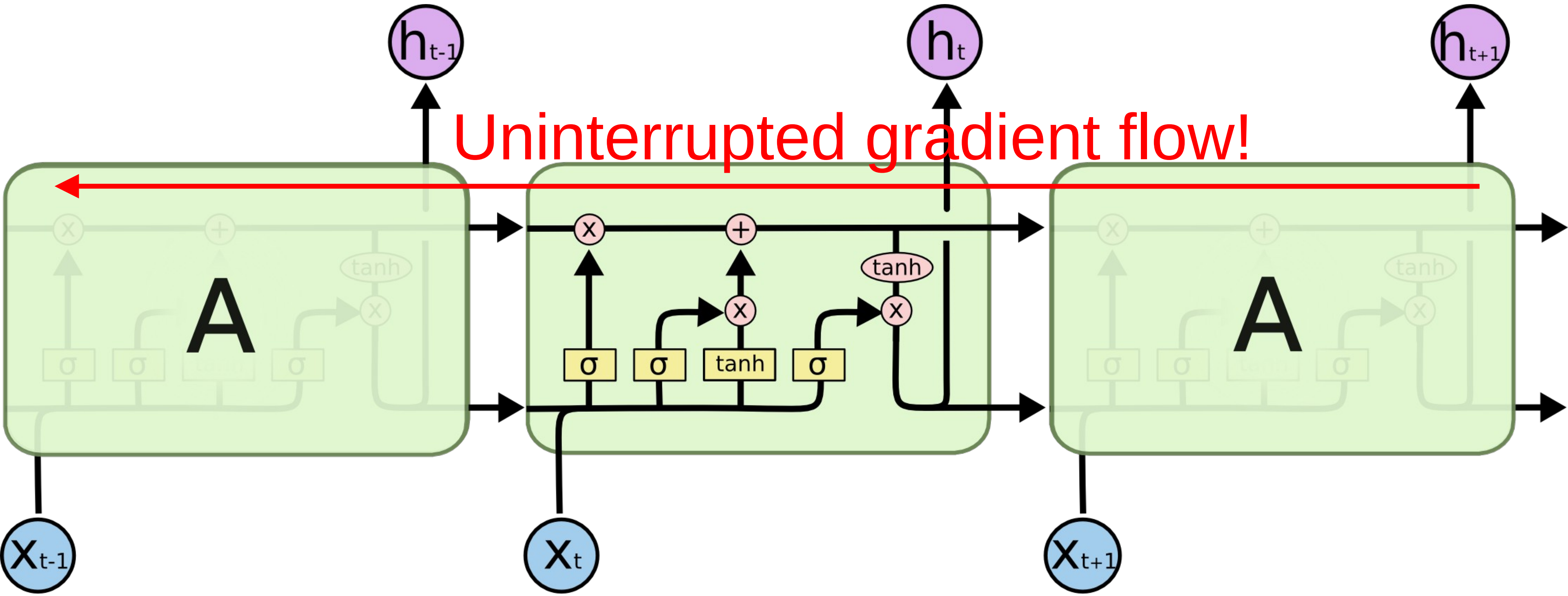


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs

- A pretty sophisticated cell

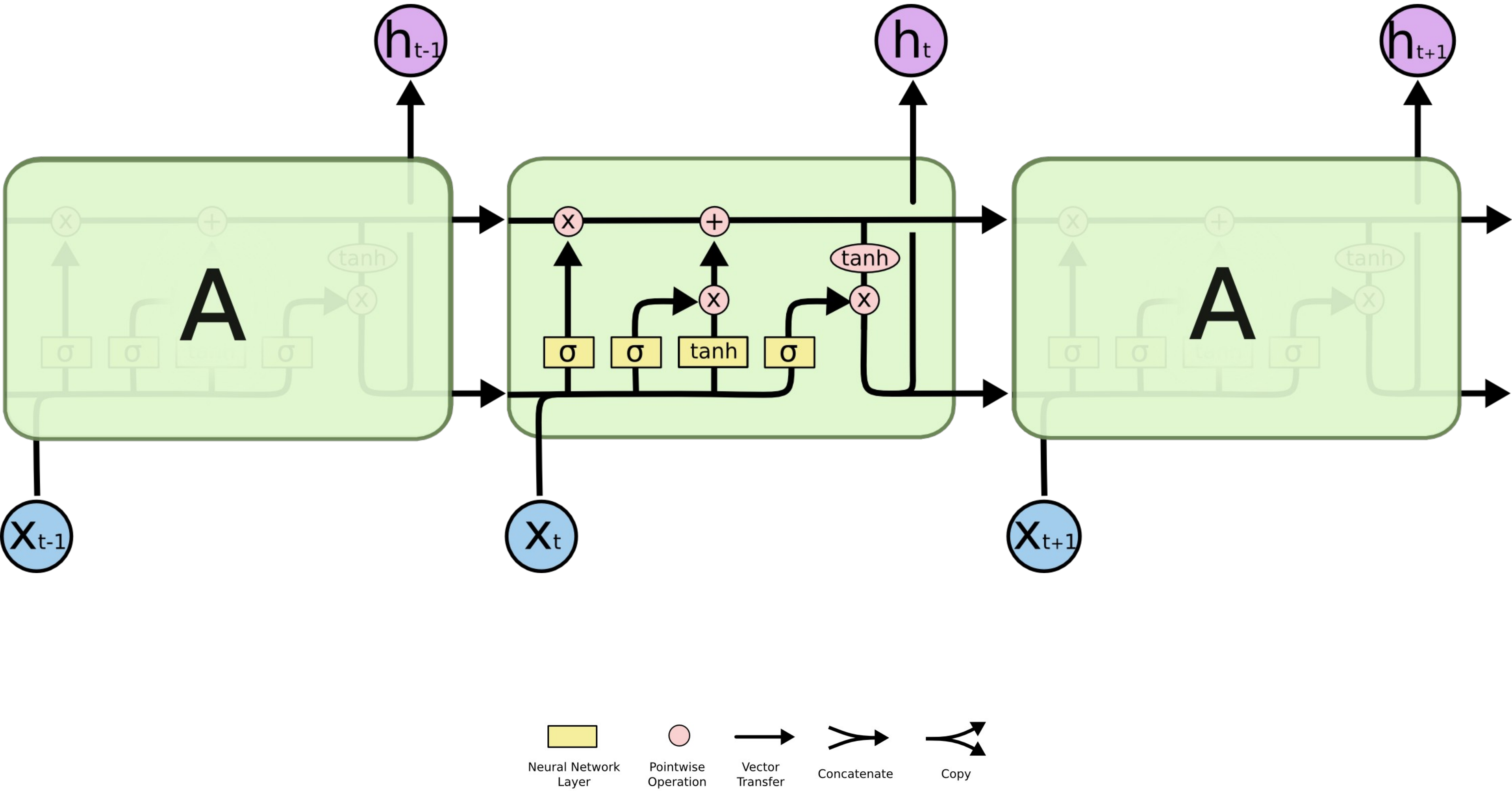


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$f_t = \boxed{\sigma}\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \boxed{\sigma}\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$o_t = \boxed{\sigma}\left(W_o \, [h_{t-1}, x_t] + b_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

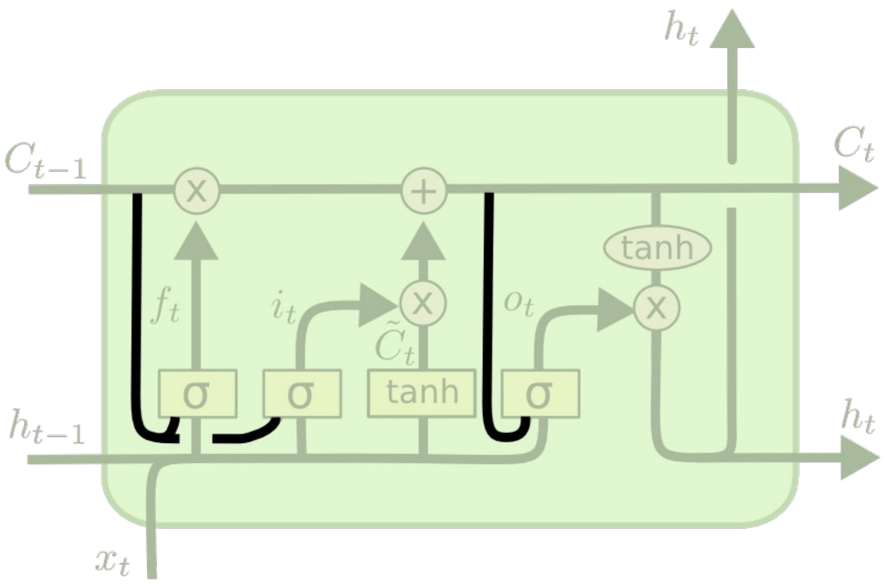
Hidden state: read ("output") some content from the cell

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

- What can you tell about cell state(C_t), if forget gate is set to 1 and input gate set to 0?
  - Information of that cell is preserved indefinitely
- What happens if you fix input gate to all 1s, forget gate to all 0s, output gate to all 1s?
  - Almost standard RNN;

LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory



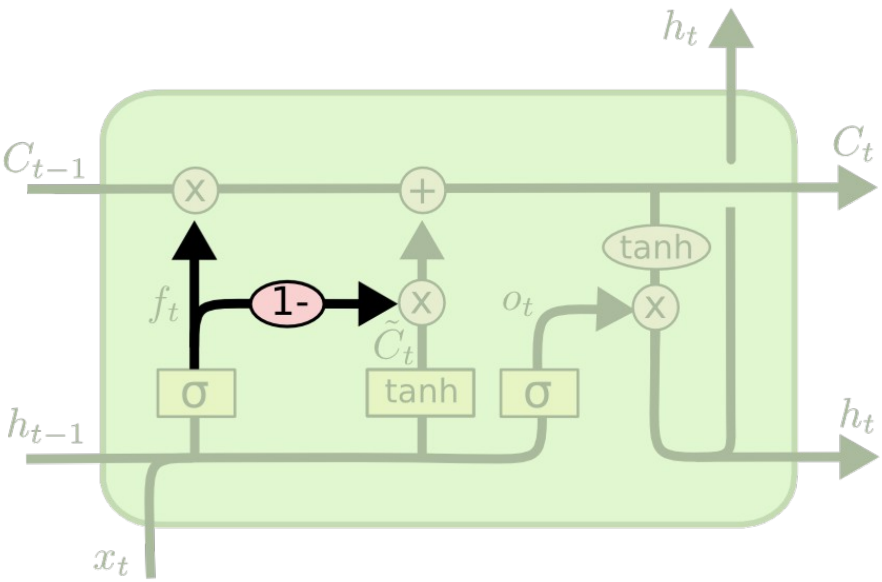
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$

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

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

LSTM Variants #2: Coupled Gates

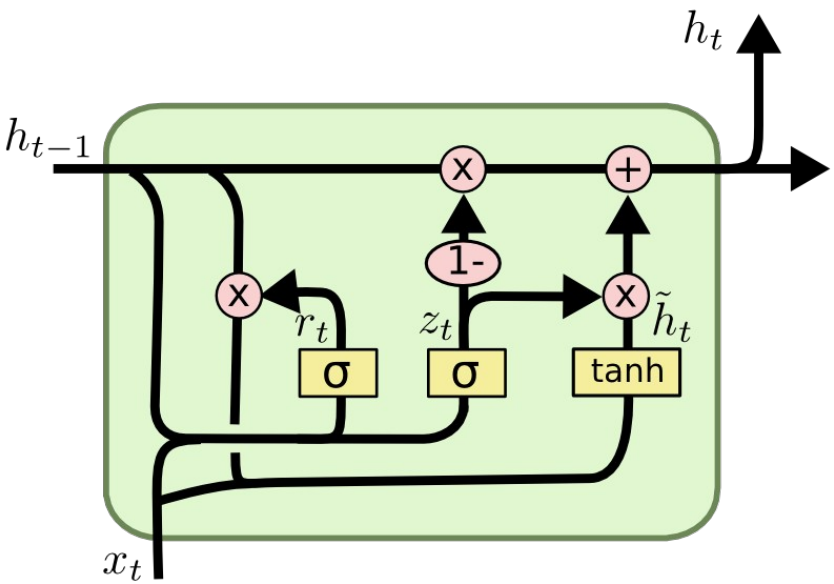
- Only memorize new if forgetting old



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM Variants #3: 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:

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

MUT2:

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

MUT3:

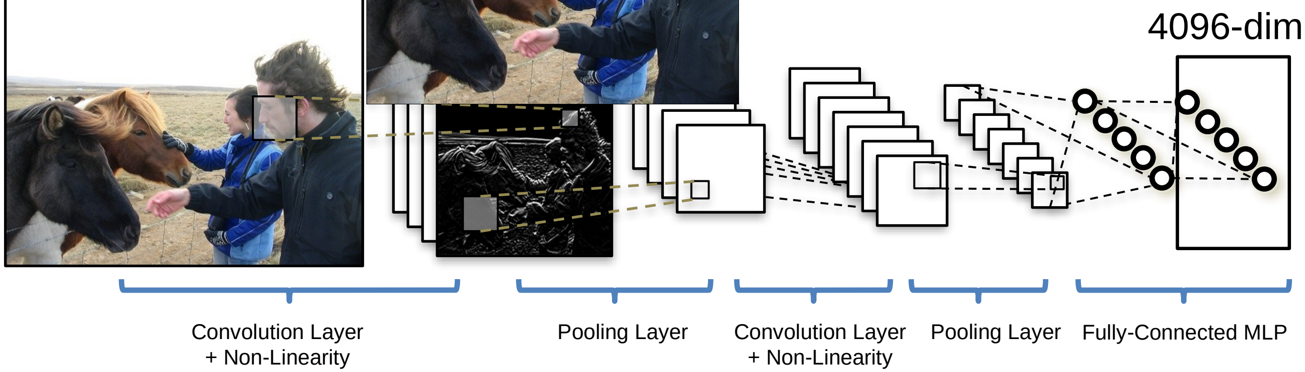
$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ \ h_t \odot (1 - z)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

Plan for Today

- Recurrent Neural Networks (RNNs)
  - Example Problem: (Character-level) Language modeling
  - Learning: (Truncated) BackProp Through Time (BPTT)
  - Visualizing RNNs
  - Example: Image Captioning
  - Inference: Beam Search
  - Multilayer RNNs
  - Problems with gradients in “vanilla” RNNs
  - LSTMs (and other RNN variants)

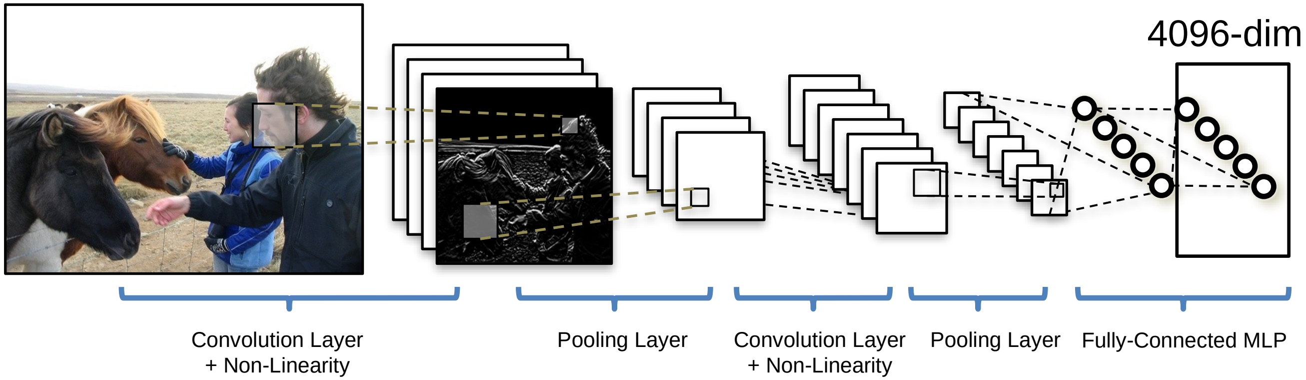
Neural Image Captioning

Image Embe Net)

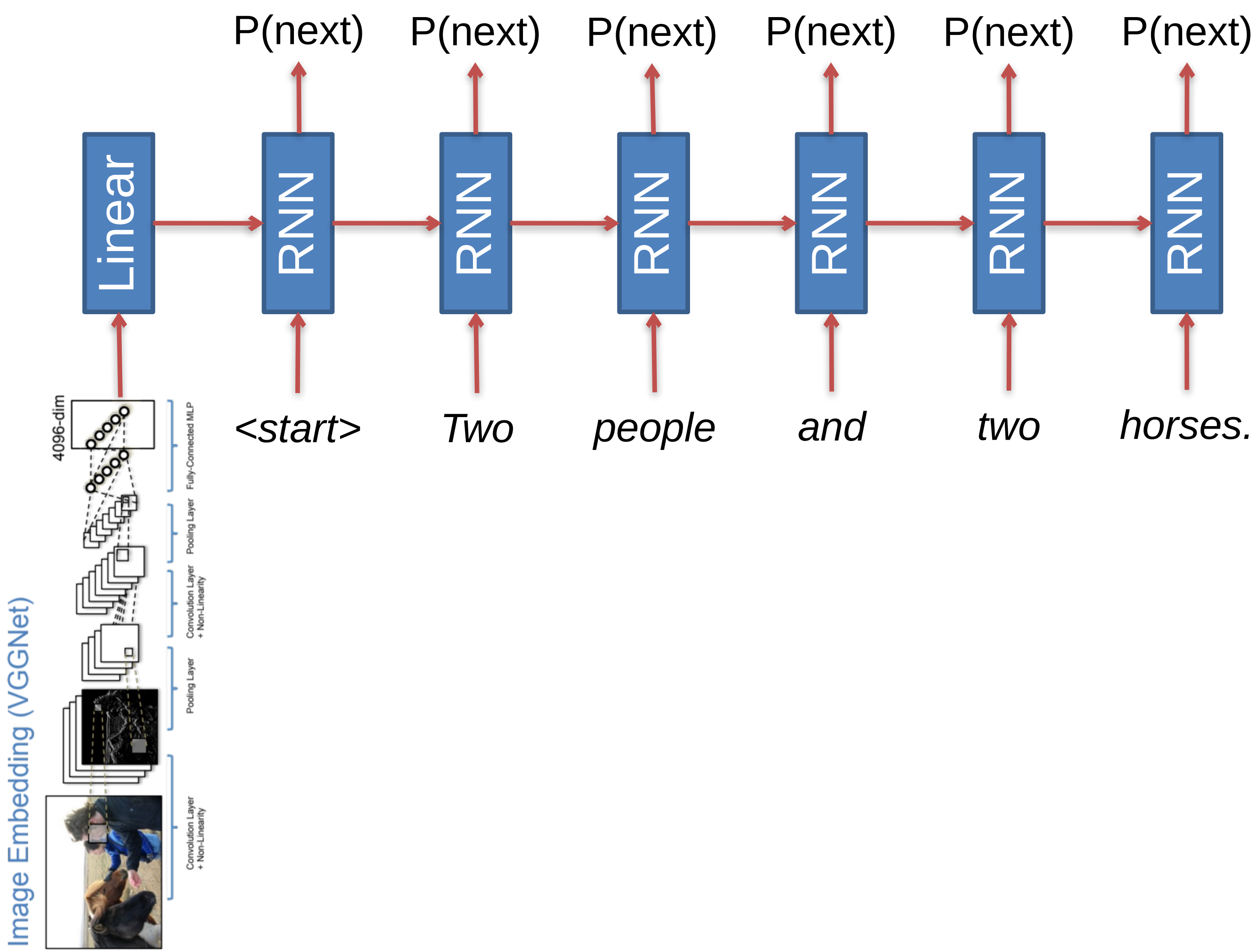


Neural Image Captioning

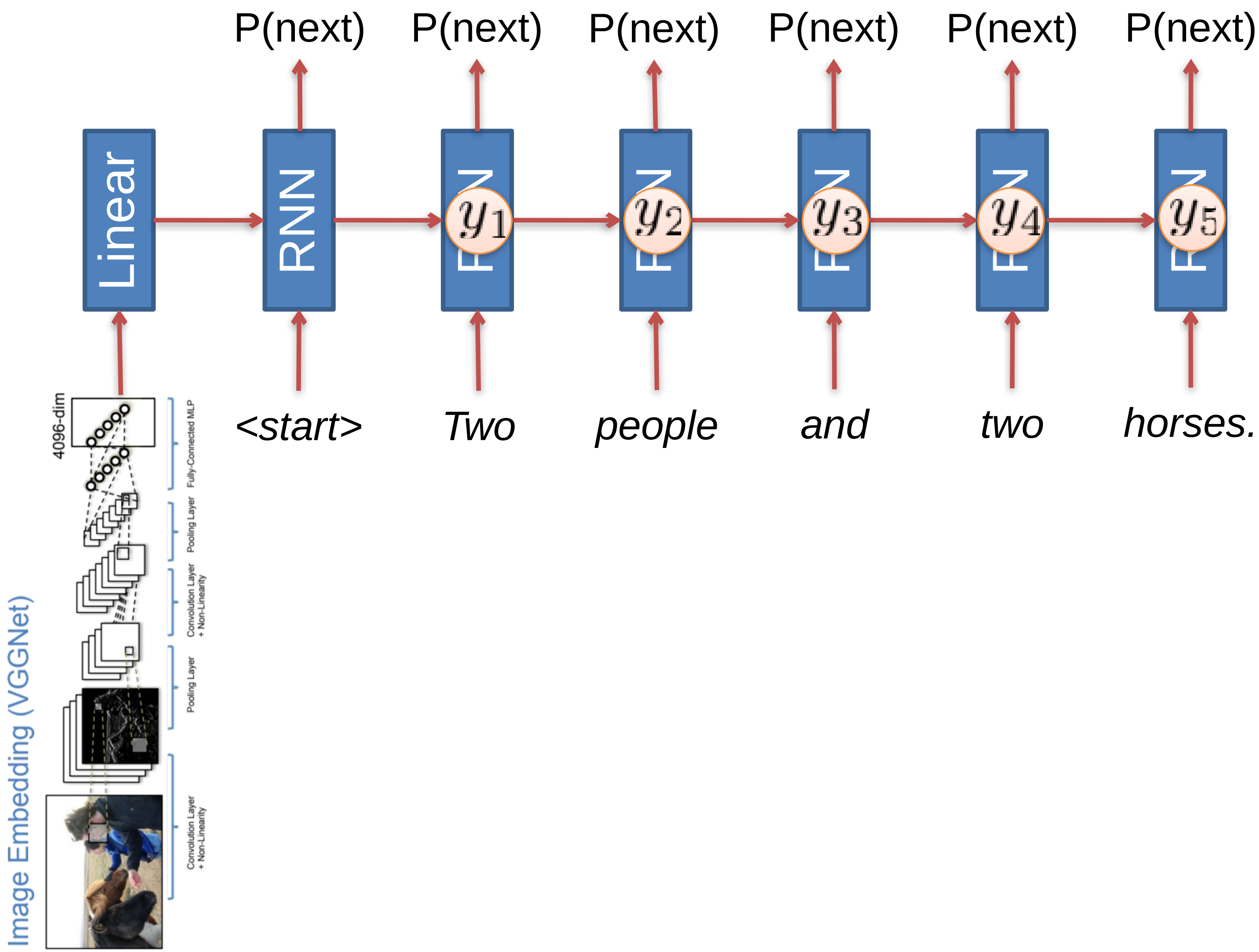
Image Embedding (VGGNet)



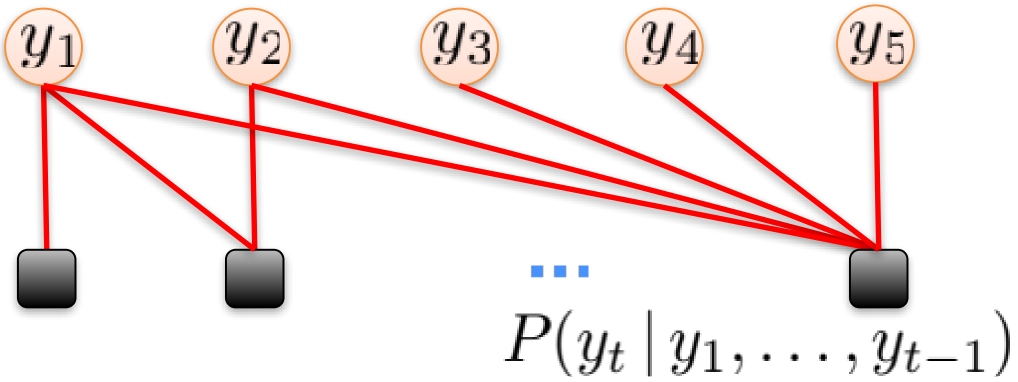
Neural Image Captioning



Neural Image Captioning



Sequence Model Factor Graph



Beam Search Demo

- http://dbs.cloudcv.org/captioning&mode=interactive

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

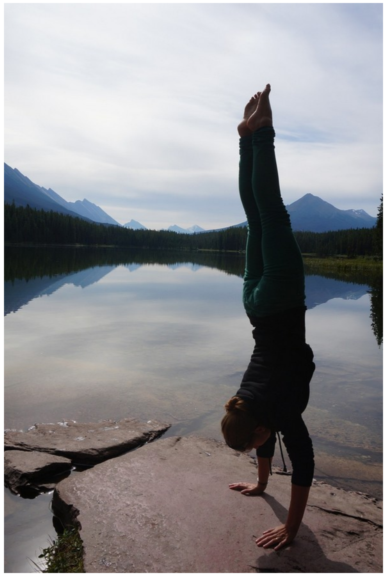
A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain:
fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.