ROB 498/599: Deep Learning for Robot Perception (DeepRob)

Lecture 16: Sequences (RNNs, Seq2Seq)

03/17/2025

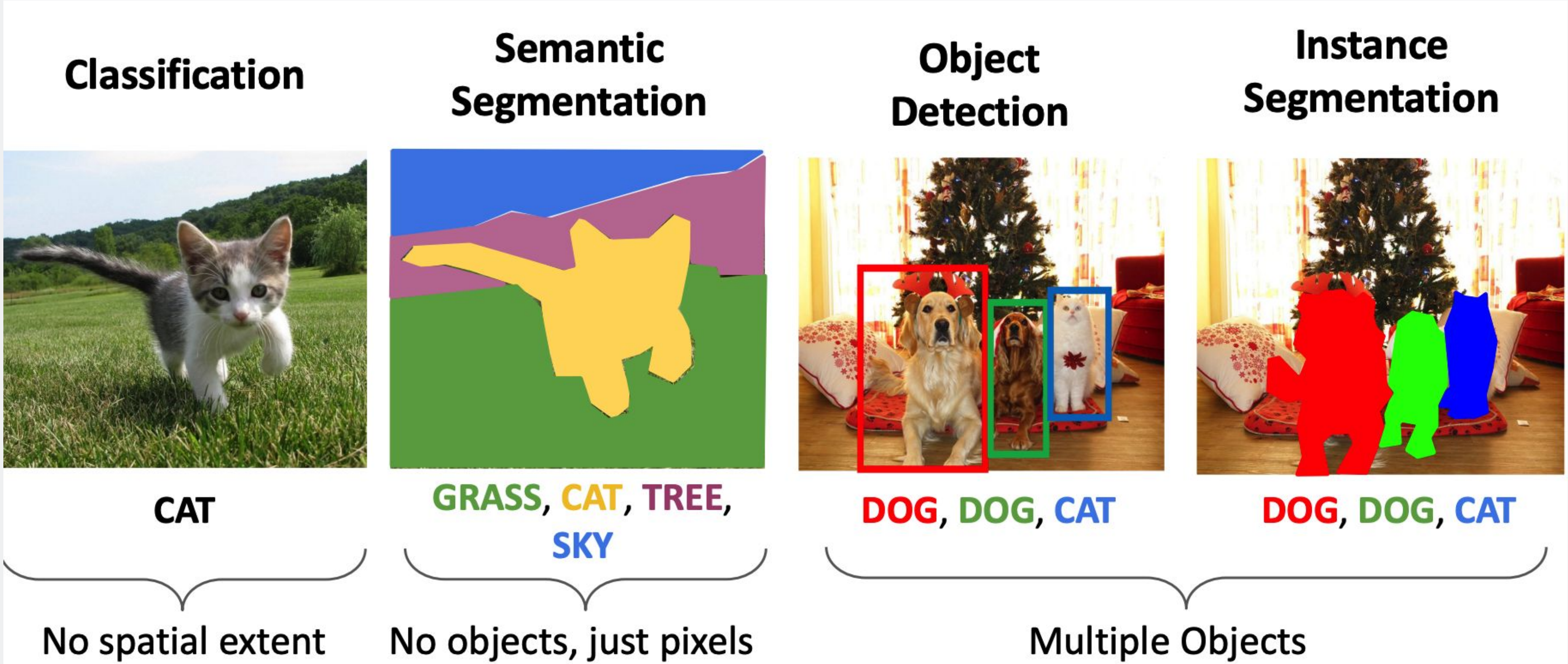
https://deeprob.org/w25/

ROBOTICS

Today

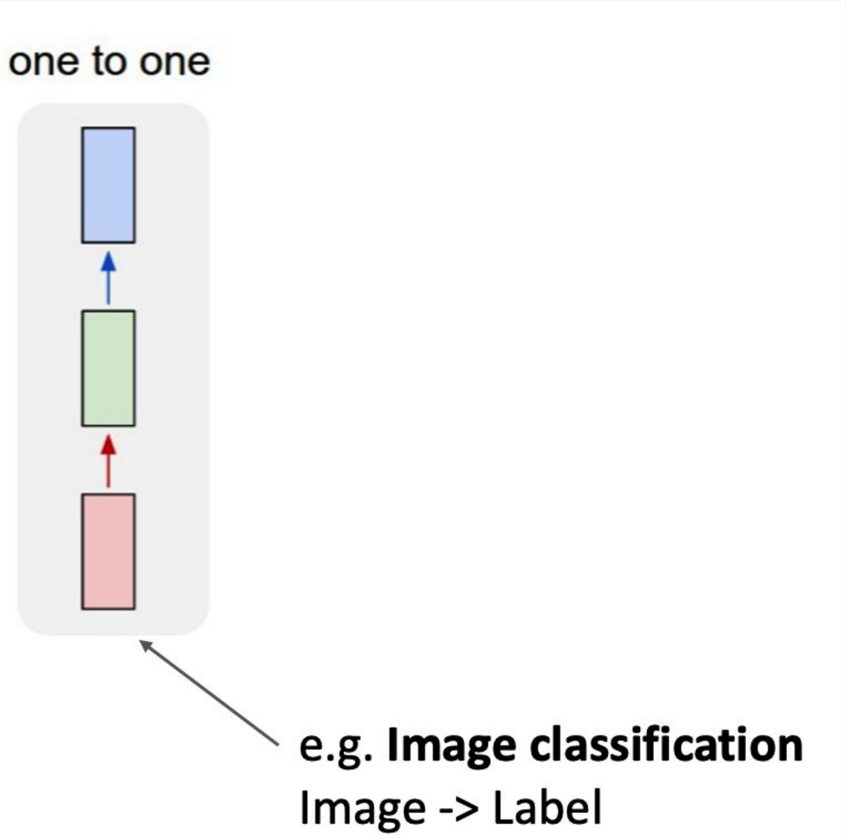
- Feedback and Recap (5min)
- Processing Sequences (40min)
  - RNN
  - LSTM
  - Seq2Seq
- Midterm Review (~20min)
- Summary and Takeaways (5min)

So far...

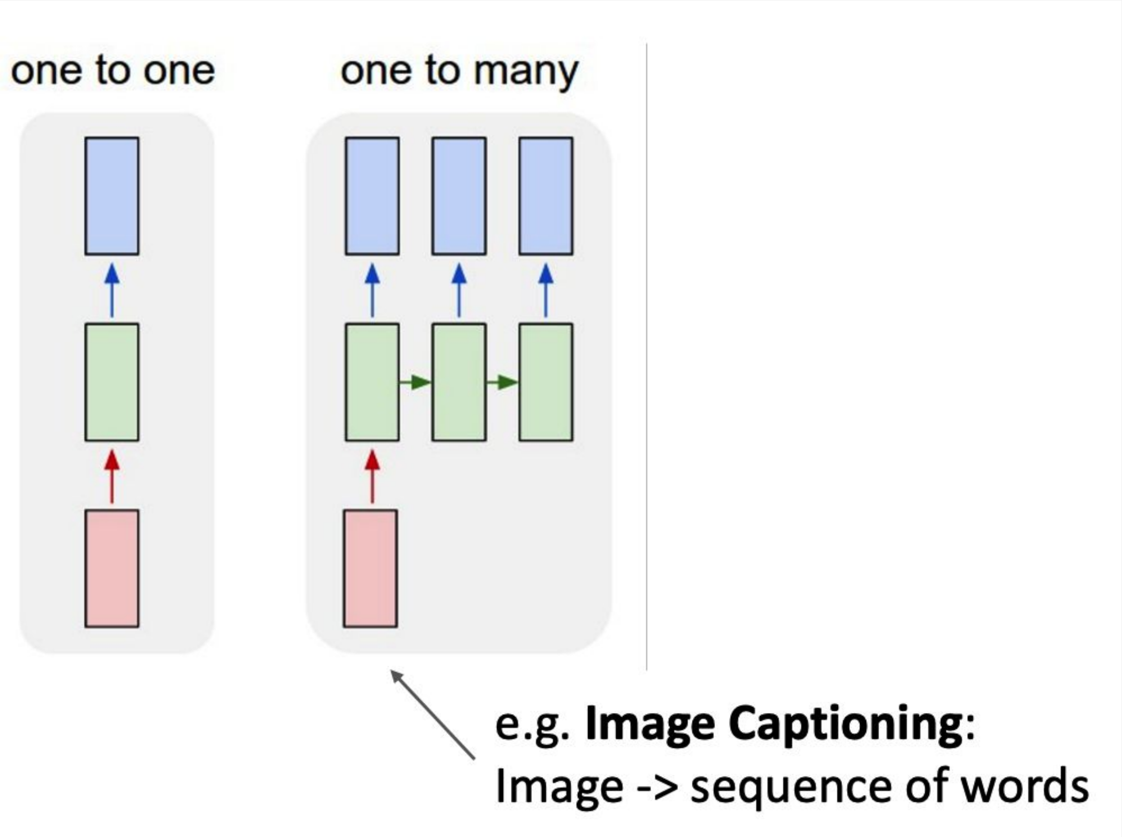


Classification
Semantic Segmentation
Object Detection
Instance Segmentation
CAT
GRASS, CAT, TREE, SKY
DOG, DOG, CAT
DOG, DOG, CAT
No spatial extent
No objects, just pixels
Multiple Objects

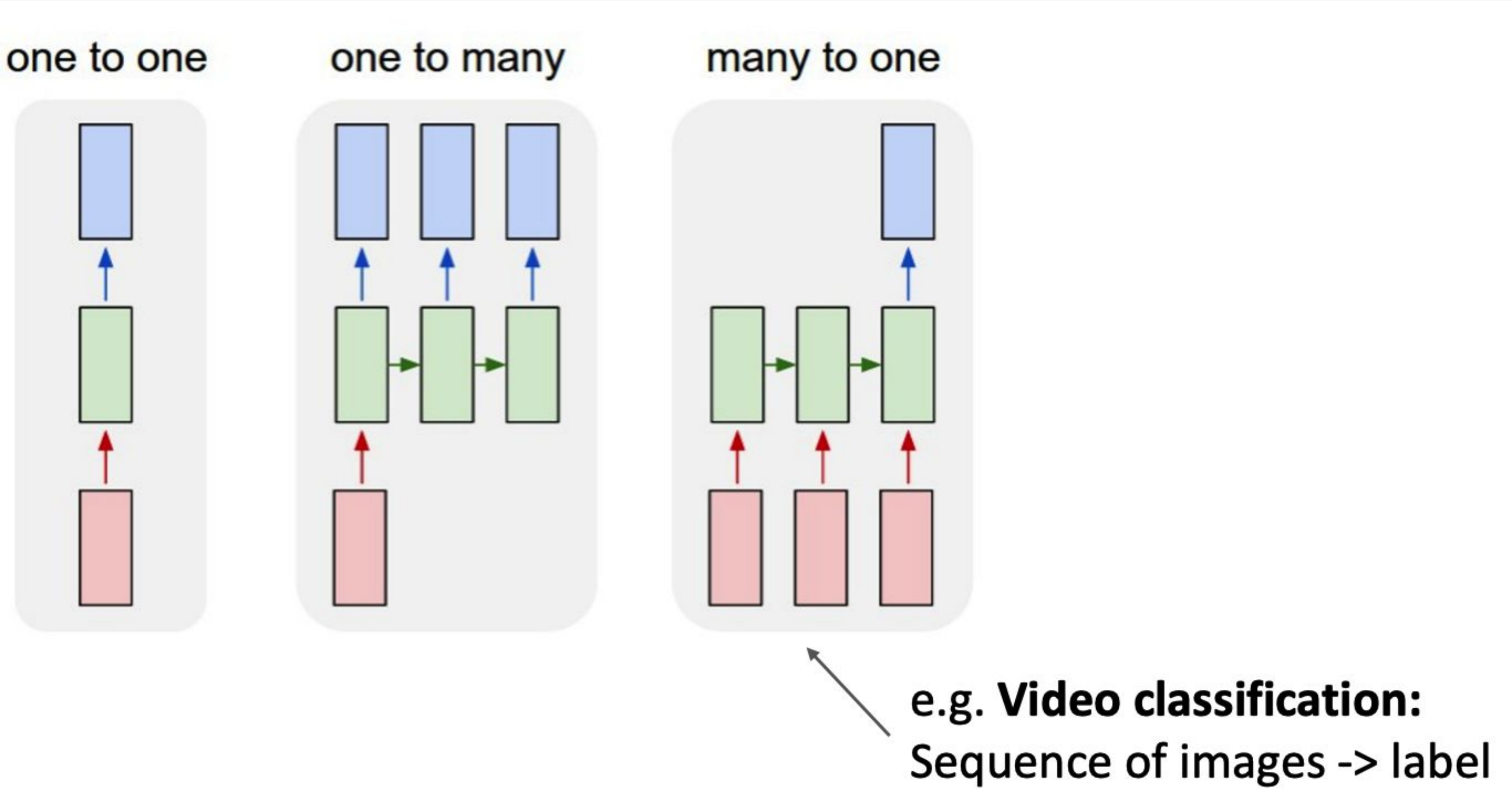
Recurrent Neural Networks



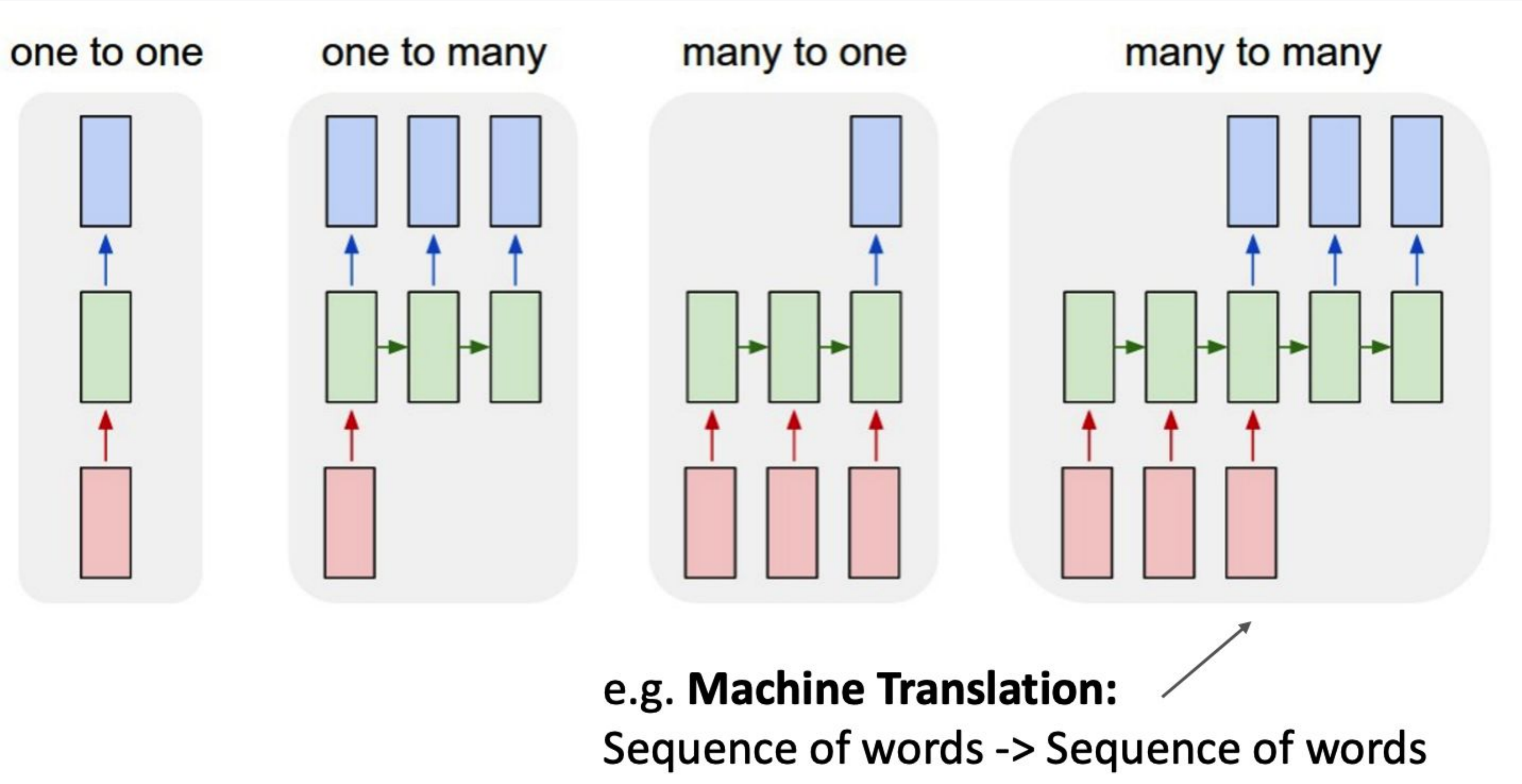
Recurrent Neural Networks



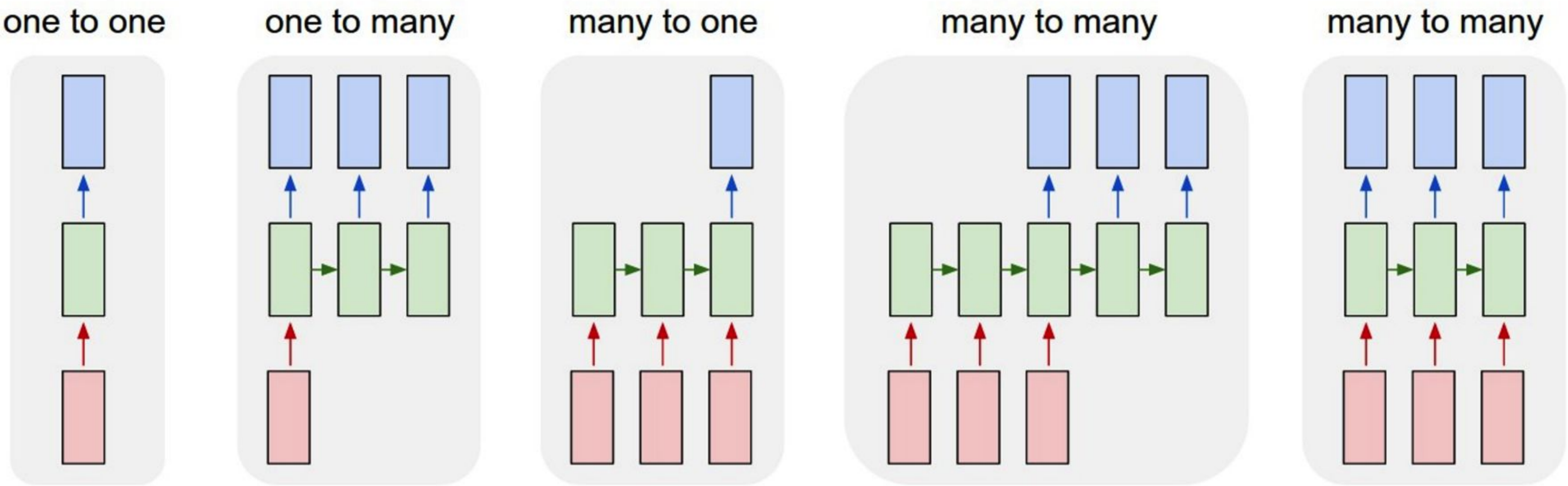
Recurrent Neural Networks



Recurrent Neural Networks

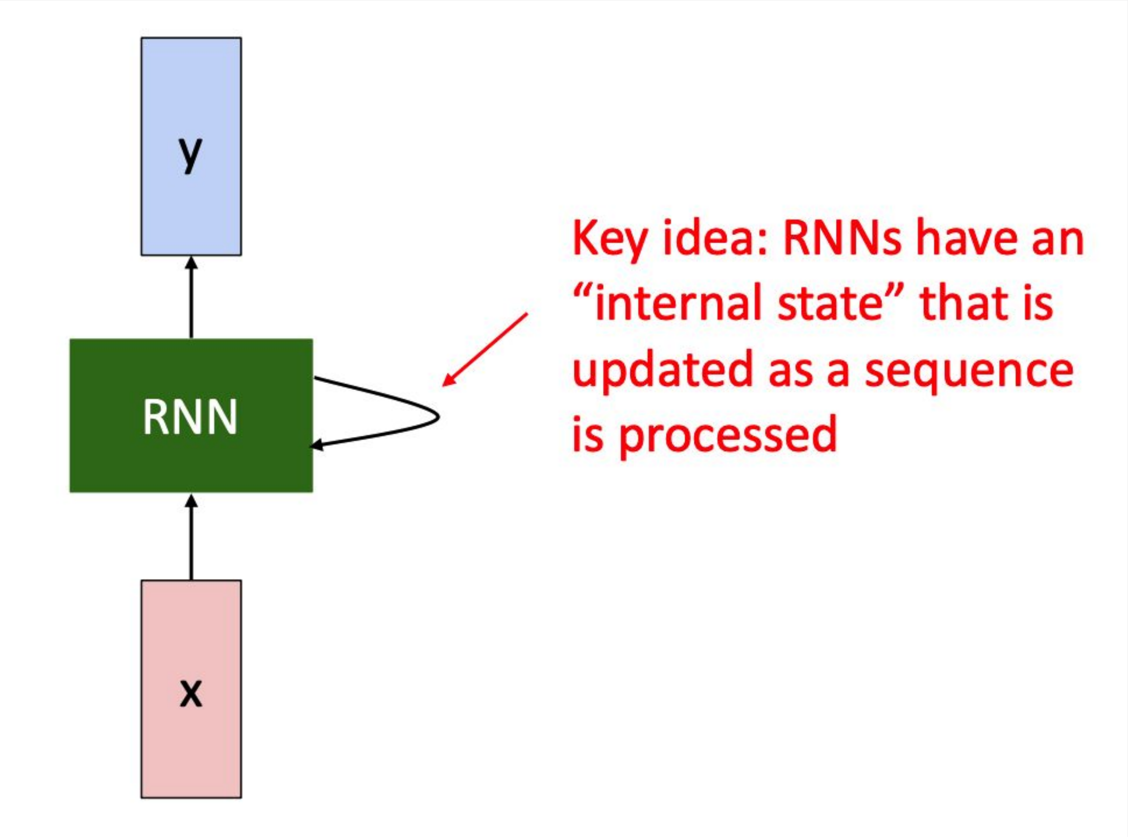


Recurrent Neural Networks

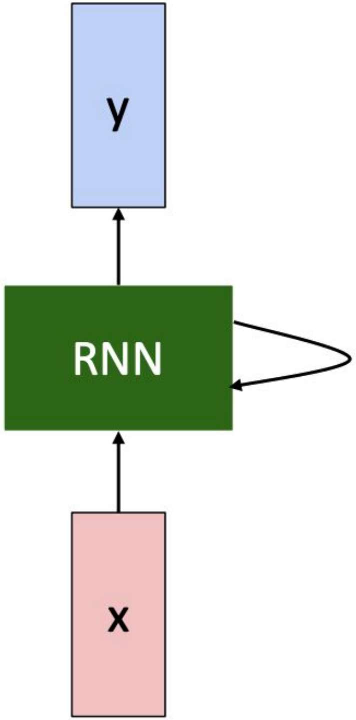


e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

Recurrent Neural Networks



Recurrent Neural Networks



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

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

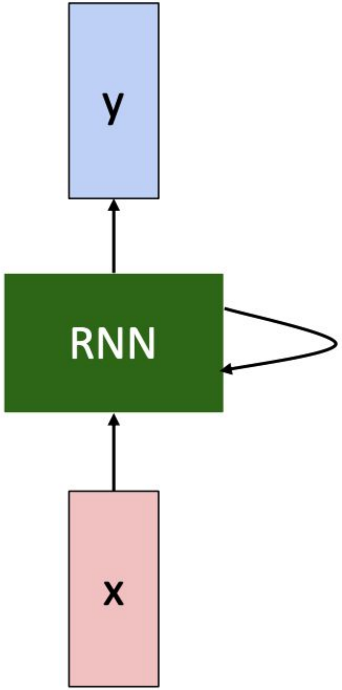
new state

some function with parameters W

old state

input vector at some time step

(Vanilla) Recurrent Neural Networks



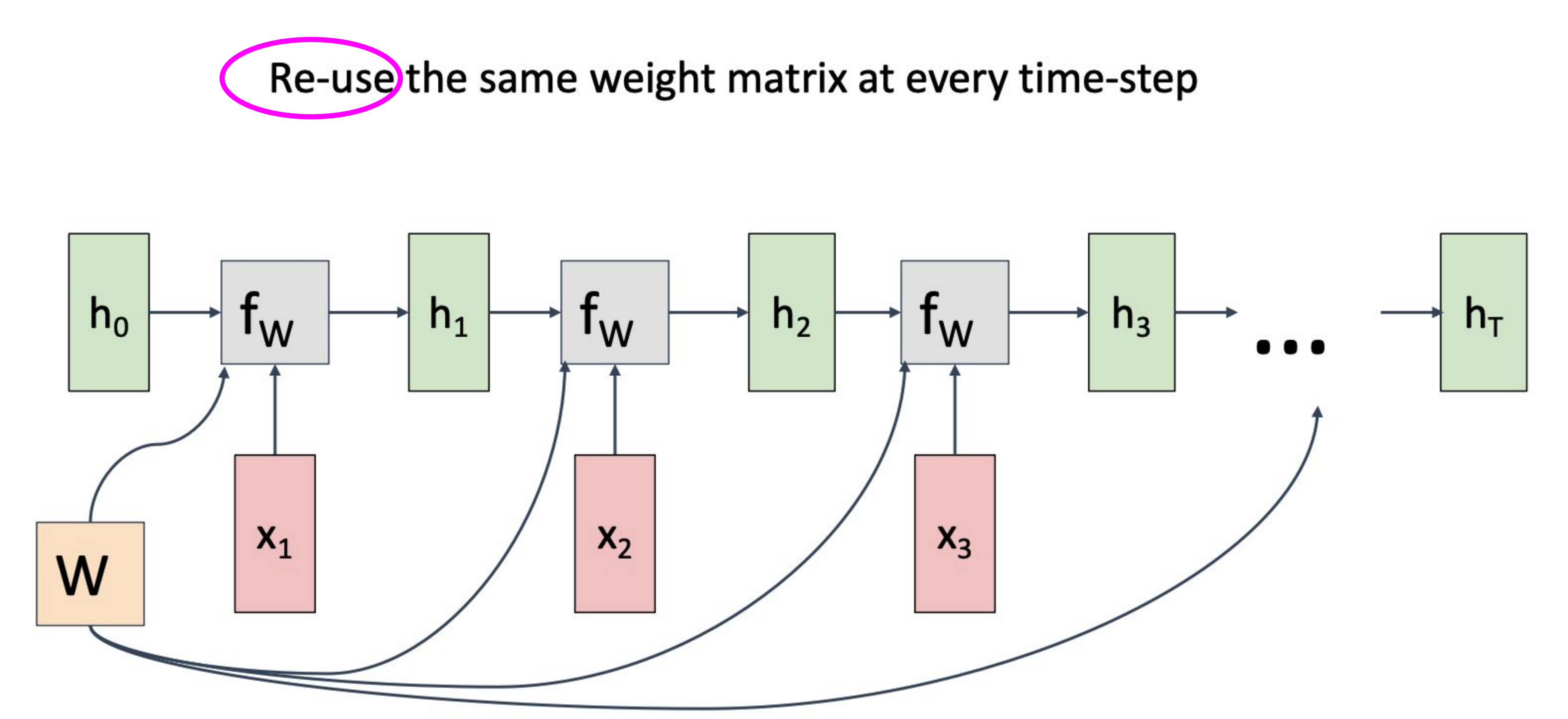
The state consists of a single *"hidden"* vector **h**:

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

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

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

Recurrent Neural Networks: computational graph

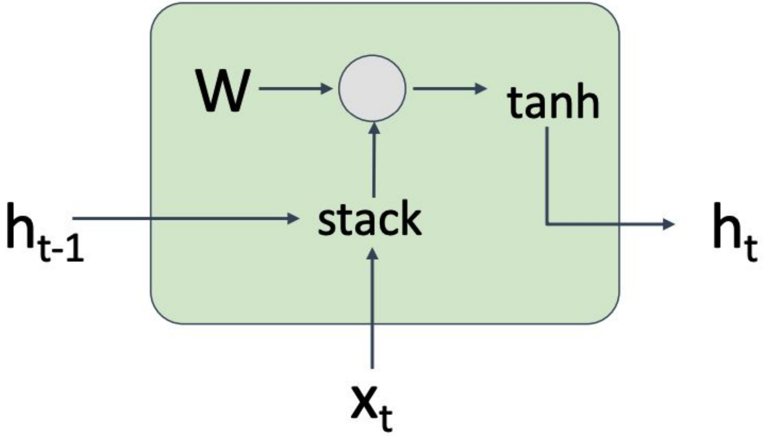


Aha Slides
(In-class participation)

https://ahaslides.com/0BWPC

Q1: why re-use (shared) weights?

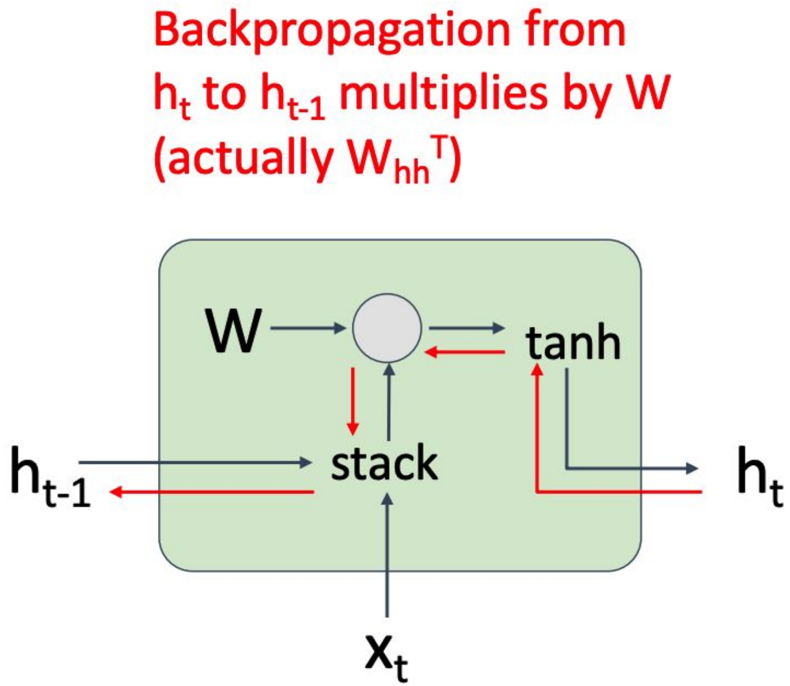
(Vanilla) Recurrent Neural Networks: Gradient Flow



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

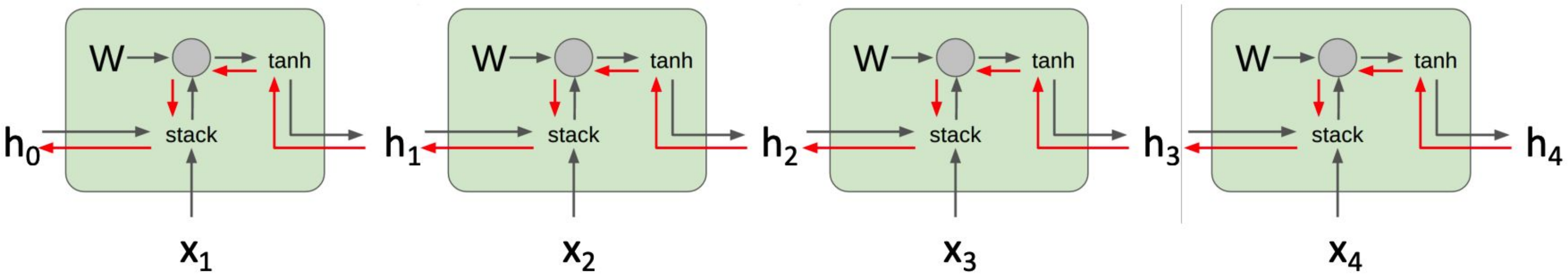
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994 https://ieeexplore.ieee.org/document/279181
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013 https://proceedings.mlr.press/v28/pascanu13.html

(Vanilla) RNN: Gradient Flow



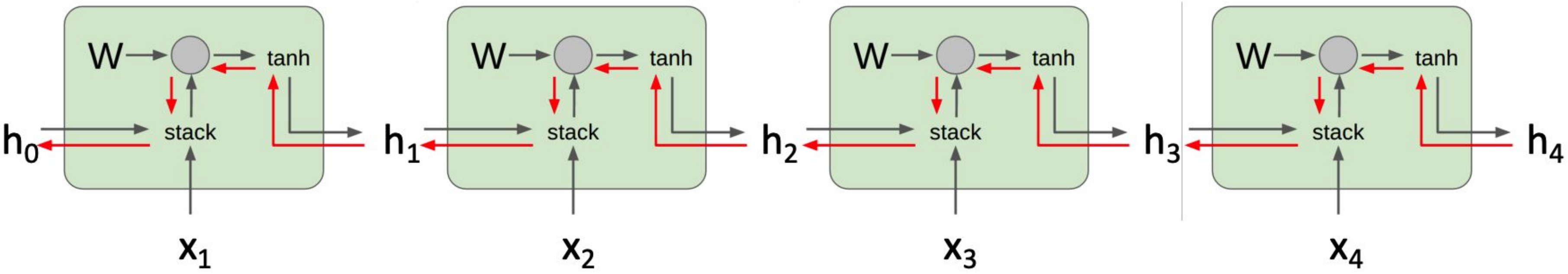
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)
\end{aligned}
$$

(Vanilla) RNN: Gradient Flow



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

(Vanilla) RNN: Gradient Flow

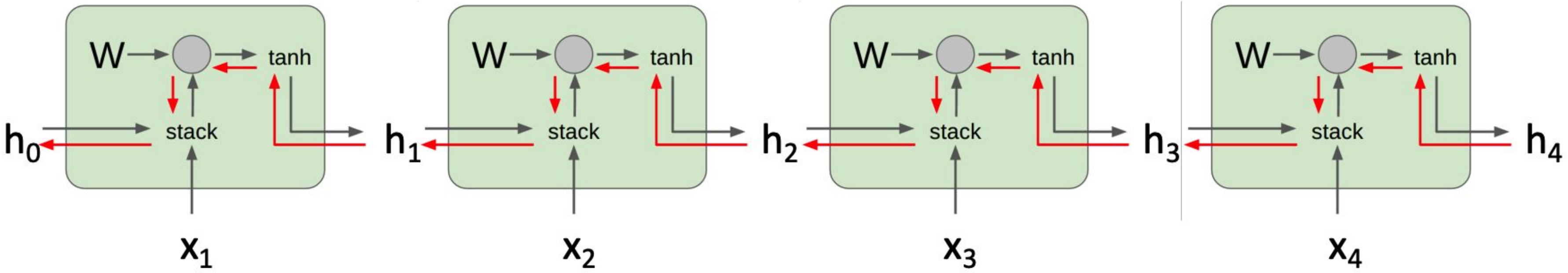


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

(Vanilla) RNN: Gradient Flow



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

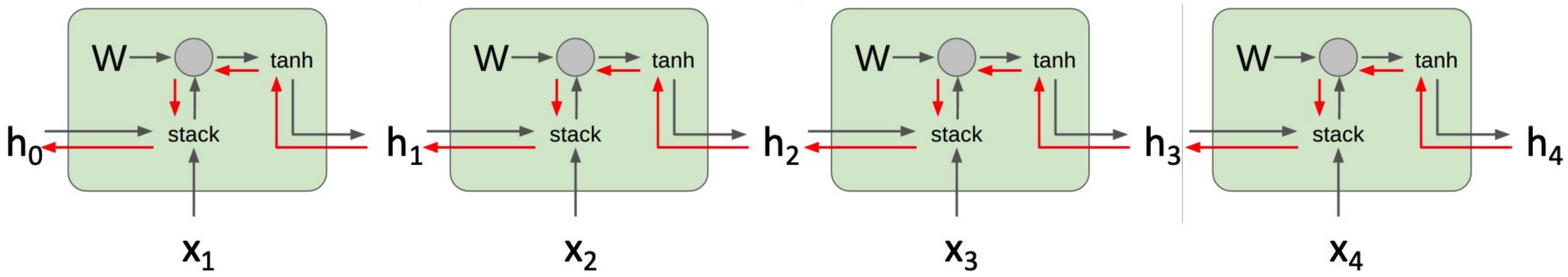
Largest singular value > 1: **Exploding gradients**

Gradient clipping: Scale gradient if its norm is too big

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

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

(Vanilla) RNN: Gradient Flow



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)

Long-range dependency

Vanilla RNN

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

Two vectors at each timestep:

Cell state: $c_t \in \mathbb{R}^H$

Hidden state: $h_t \in \mathbb{R}^H$

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h\right)$$

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

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

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997 https://www.bioinf.jku.at/publications/older/2604.pdf

Long Short Term Memory (LSTM)

Compute four "gates" per timestep:

Input gate: $i_t \in \mathbb{R}^H$

Forget gate: $f_t \in \mathbb{R}^H$

Output gate: $o_t \in \mathbb{R}^H$

"Gate?" gate: $g_t \in \mathbb{R}^H$

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

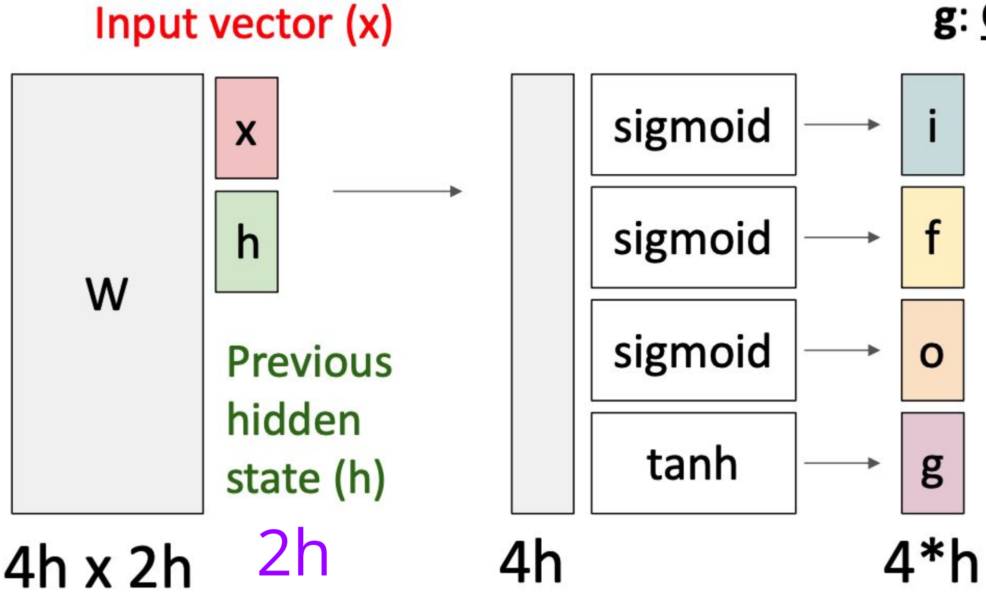
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997 https://www.bioinf.jku.at/publications/older/2604.pdf

Long Short Term Memory (LSTM)

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

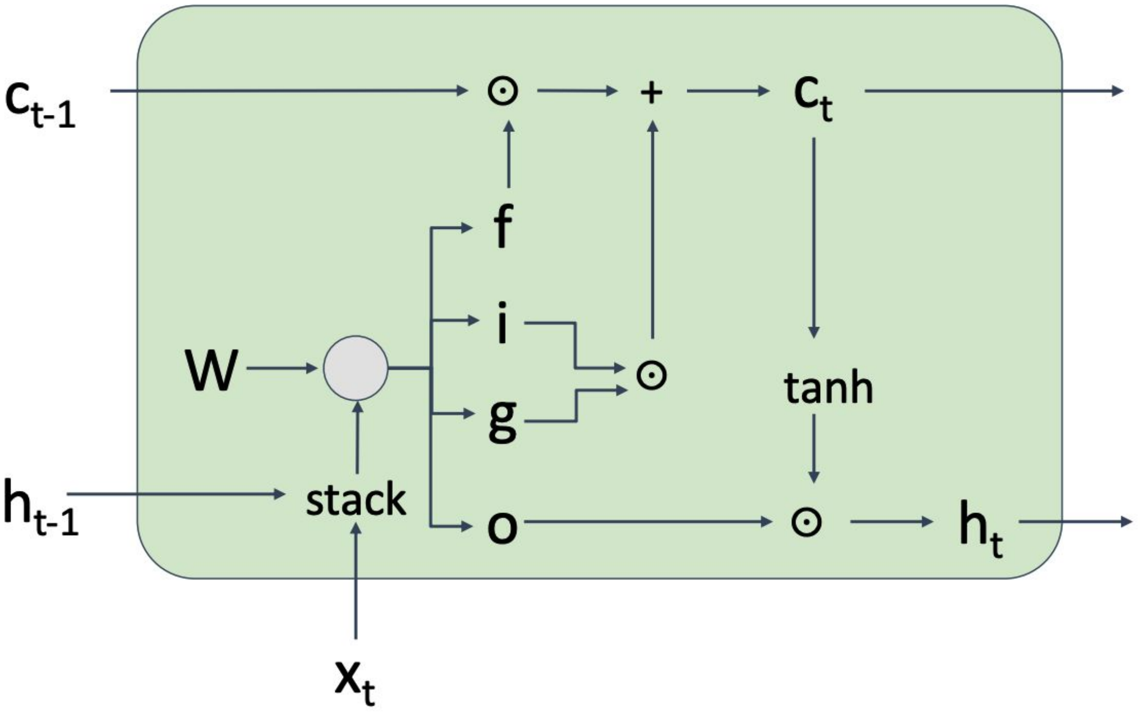


$$\boxed{\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)}$$

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

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

Long Short Term Memory (LSTM)



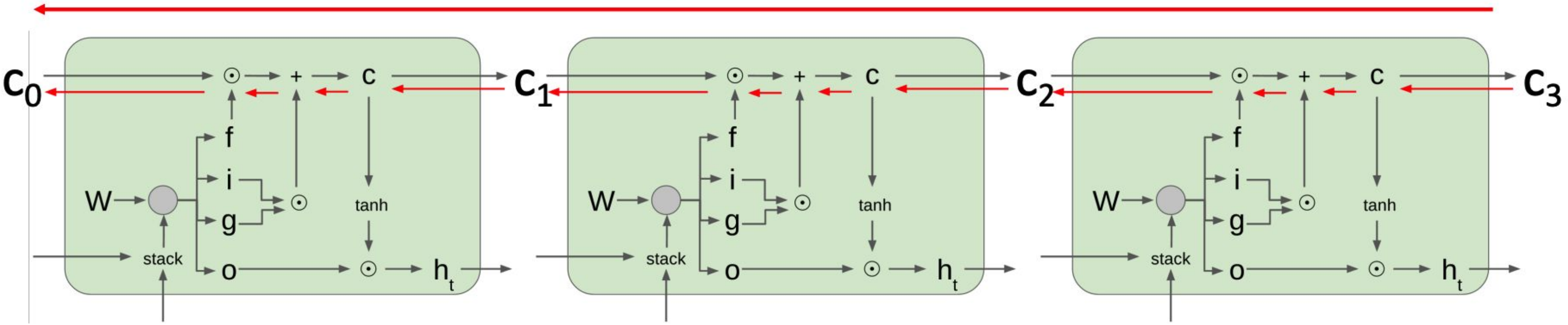
$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_h \right)$$

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

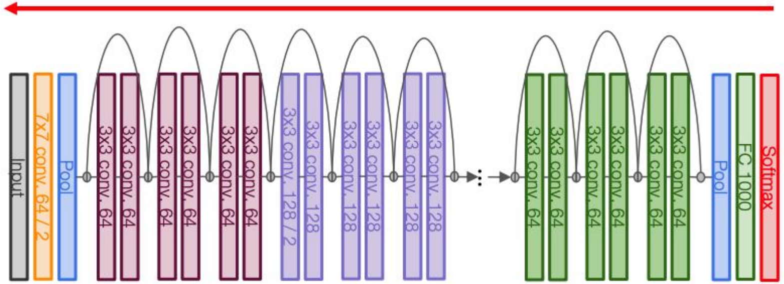
$$h_t = o_t \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

Uninterrupted gradient flow!



Similar to ResNet!



"Highway network"

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46171.pdf

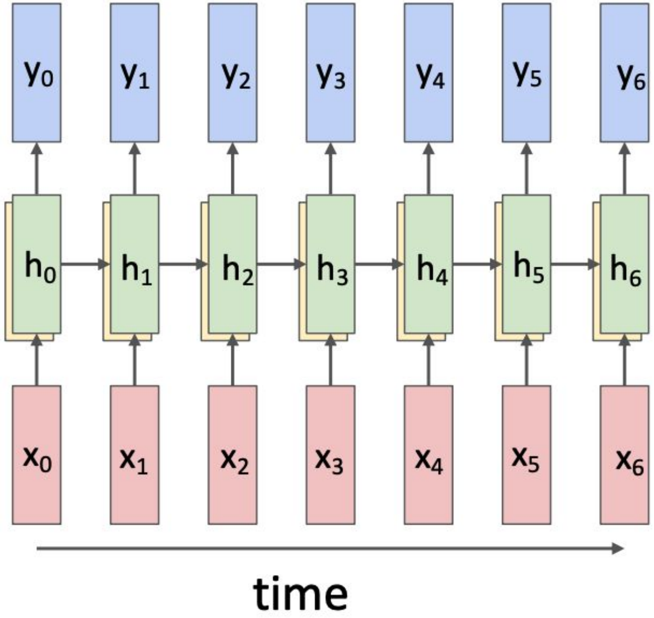
https://arxiv.org/pdf/1505.00387

Single-Layer RNN

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

LSTM:

$$\begin{pmatrix}i_t\\ f_t\\ o_t\\ g_t\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix} + b_h\right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$



Multi-Layer RNN

$$h_t^{\ell} = \tanh\left(W\begin{pmatrix}h_{t-1}^{\ell}\\ h_t^{\ell-1}\end{pmatrix} + b_h^{\ell}\right)$$

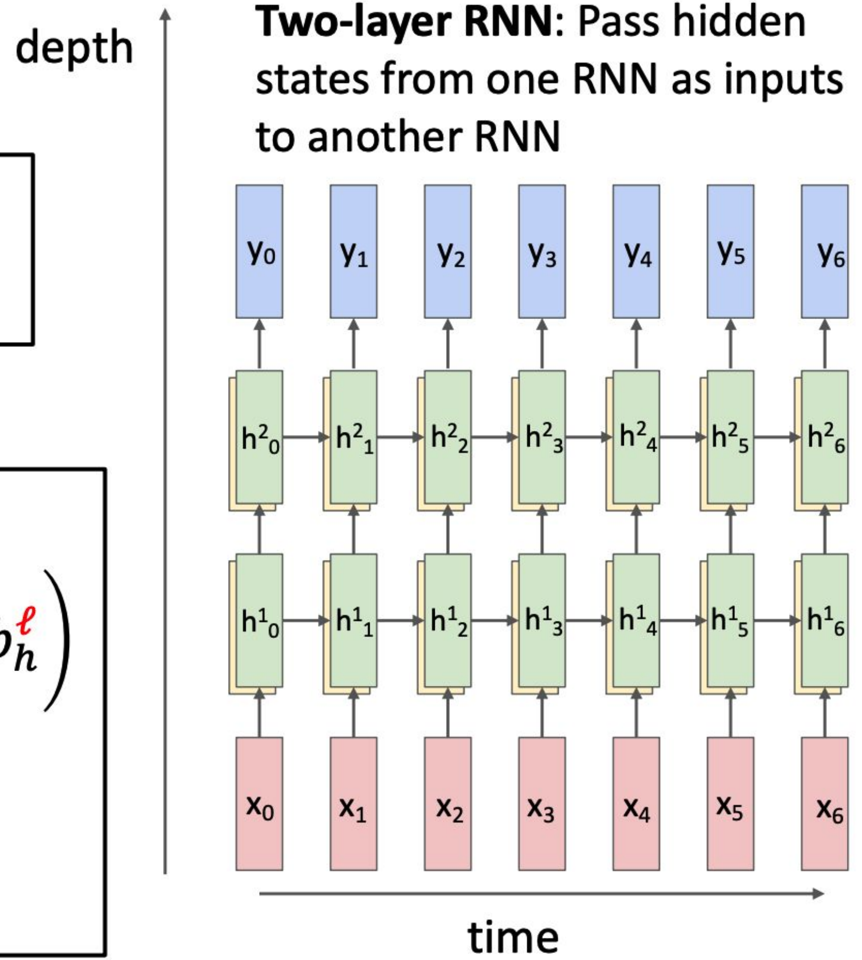
LSTM:

$$\begin{pmatrix}i_t^{\ell}\\ f_t^{\ell}\\ o_t^{\ell}\\ g_t^{\ell}\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}\left(W\begin{pmatrix}h_{t-1}^{\ell}\\ h_t^{\ell-1}\end{pmatrix} + b_h^{\ell}\right)$$

$$c_t^{\ell} = f_t^{\ell} \odot c_{t-1}^{\ell} + i_t^{\ell} \odot g_t^{\ell}$$

$$h_t^{\ell} = o_t^{\ell} \odot \tanh(c_t^{\ell})$$

Two-layer RNN: Pass hidden states from one RNN as inputs to another RNN



Add Three-layer, etc.

Vanilla RNN - 112 Lines of Python Code

Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy

min-char-rnn.py

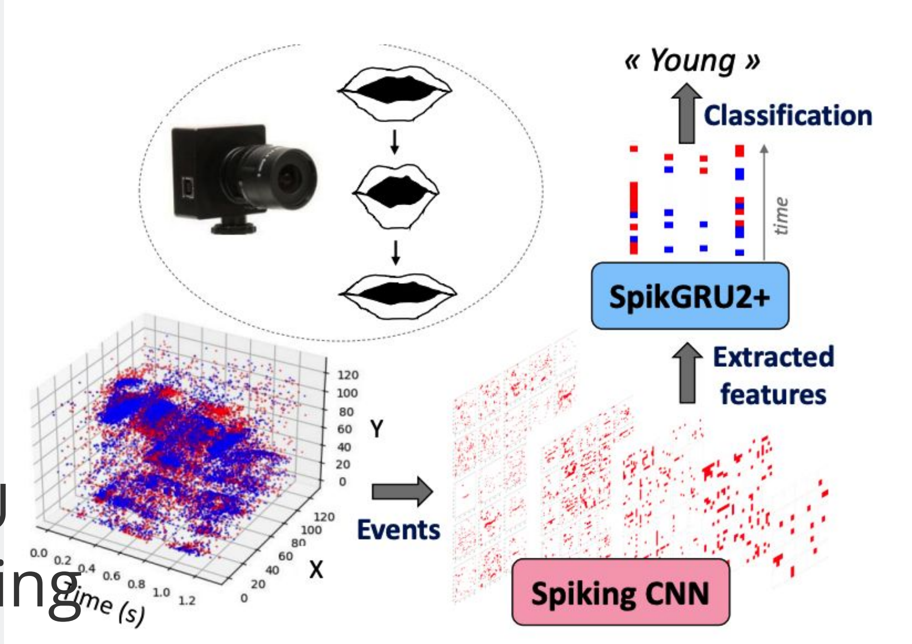
```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

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

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
```

https://gist.github.com/karpathy/d4dee566867f8291f086

ROBOTICS

Gated Recurrent Unit (GRU)

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_T \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

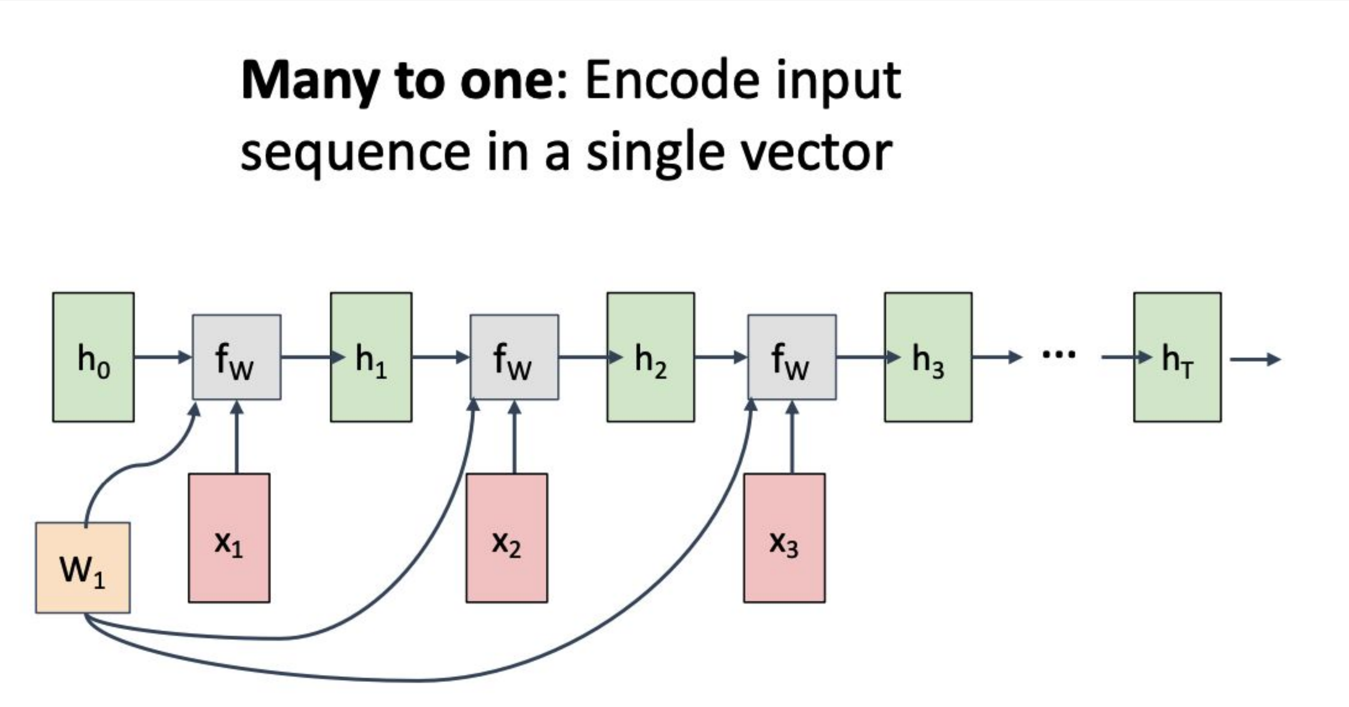


E.g., CVPR 2024
Spiking Neural Networks + GRU
for speech recognition/lip reading

Cho et al "Learning phrase representations using RNN encoder-decoder for statistical machine translation", 2014
https://arxiv.org/abs/1406.1078

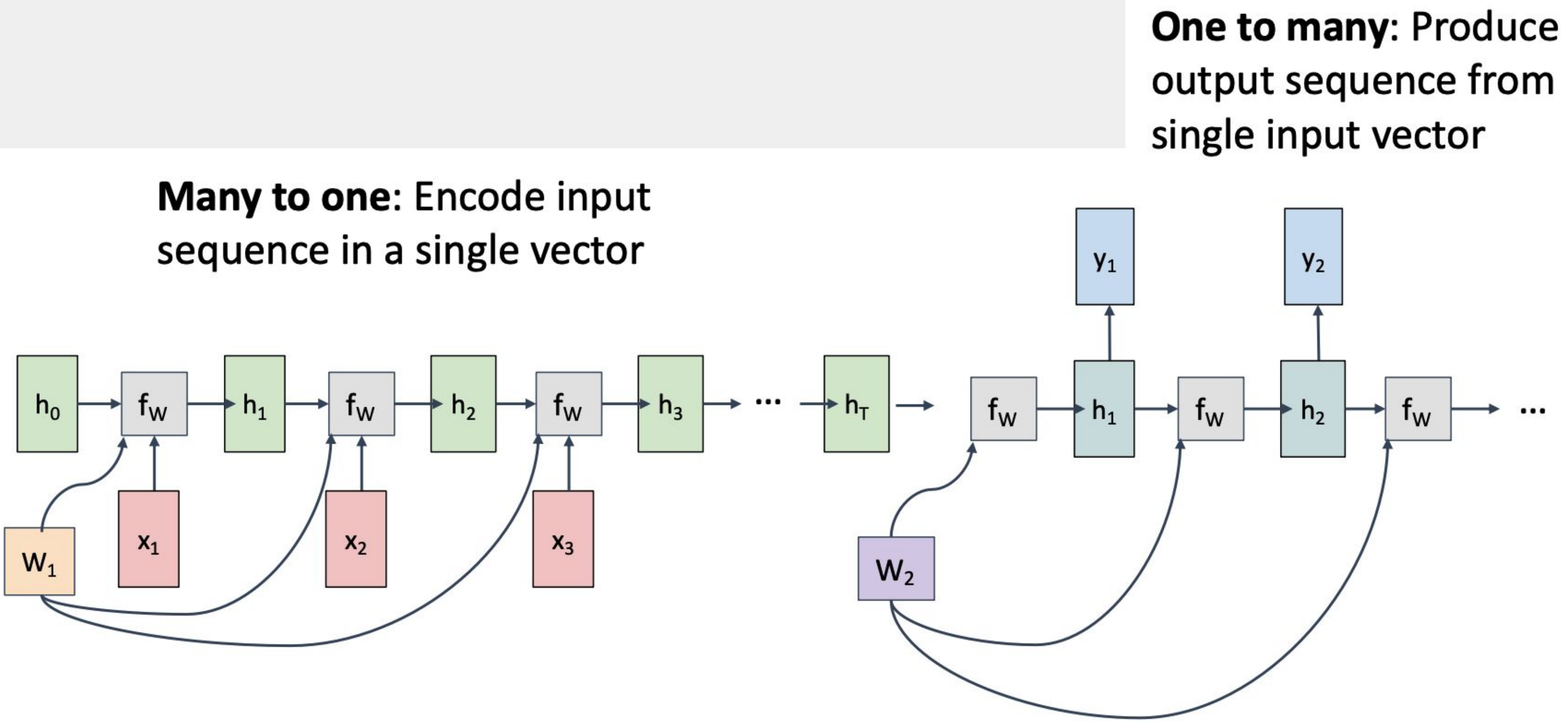
https://openaccess.thecvf.com/content/CVPR2024W/EVW/papers/Dampfhoffer_Neuromorphic_Lip-Reading_with_Signed_Spiking_Gated_Recurrent_Units_CVPRW_2024_paper.pdf
Spike GPT https://arxiv.org/pdf/2302.13939

Seq2Seq: Sequence to Sequence



https://proceedings.neurips.cc/paper_files/paper/2014/file/5a18e133cbf9f257297f410bb7eca942-Paper.pdf

Seq2Seq: Sequence to Sequence



https://proceedings.neurips.cc/paper_files/paper/2014/file/5a18e133cbf9f257297f410bb7eca942-Paper.pdf

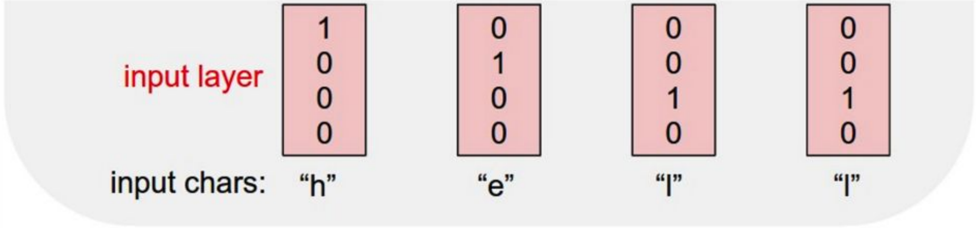
M | ROBOTICS

Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

Training sequence: "hello"

Vocabulary: [h, e, l, o]



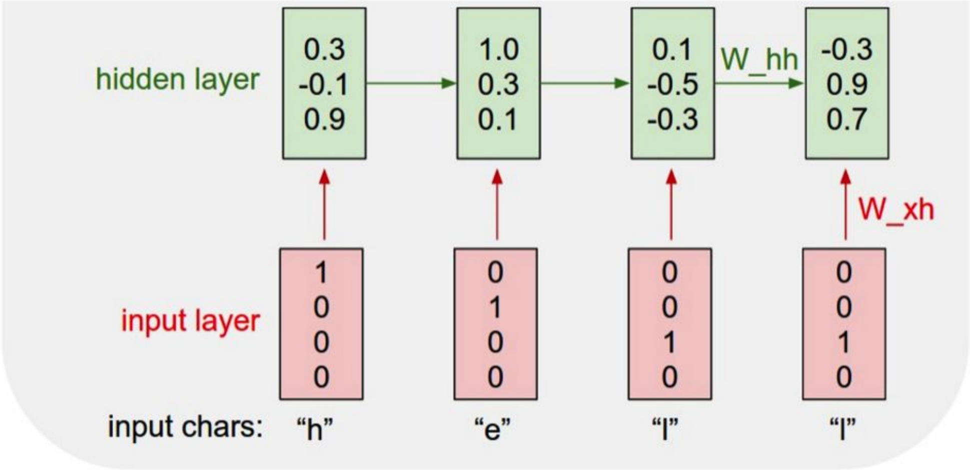
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



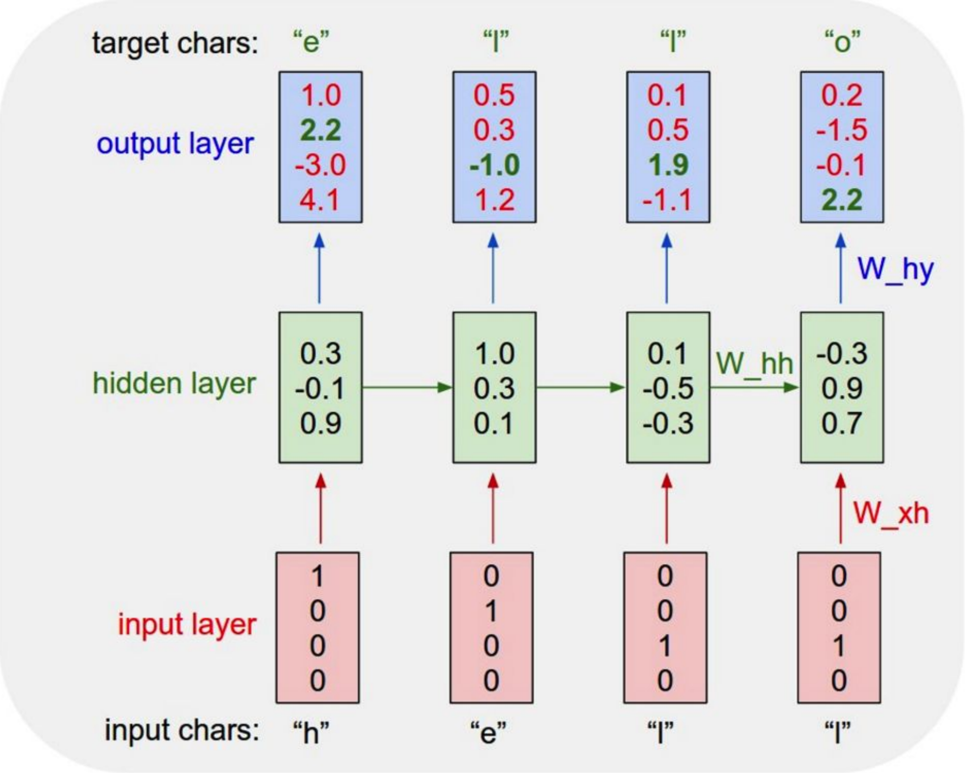
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



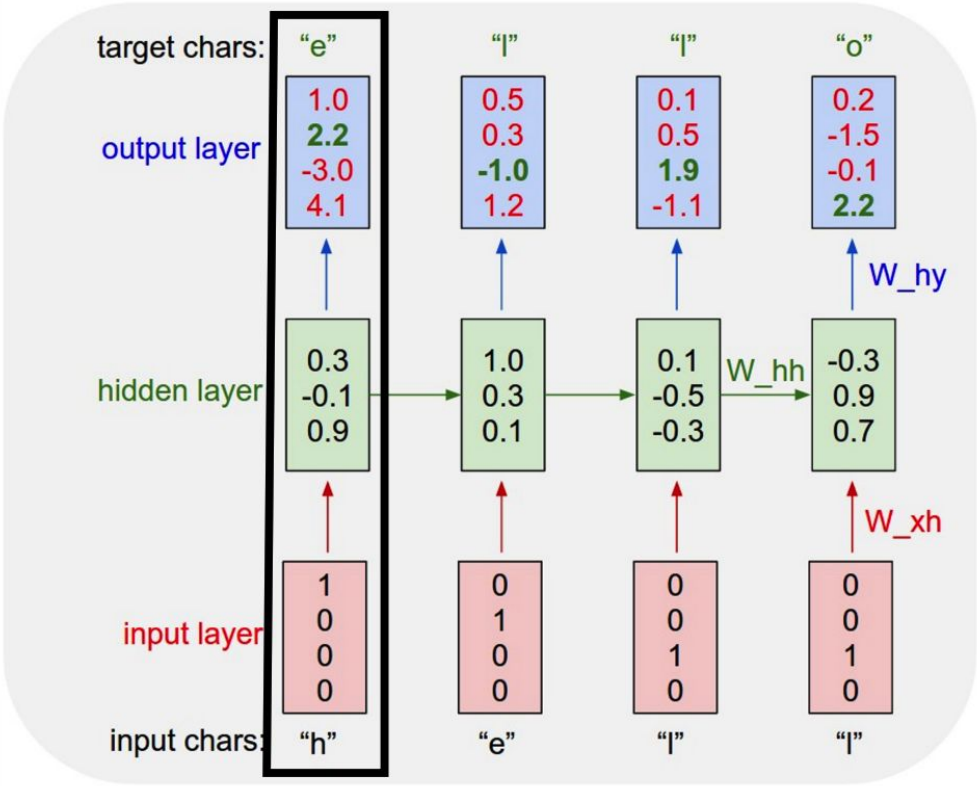
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



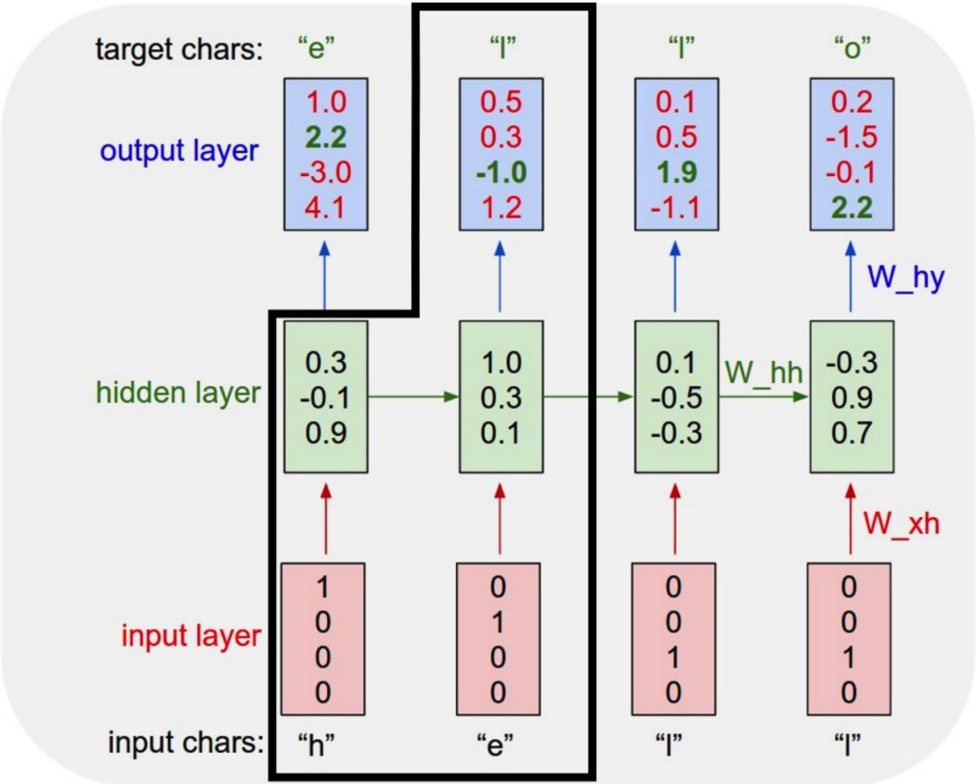
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



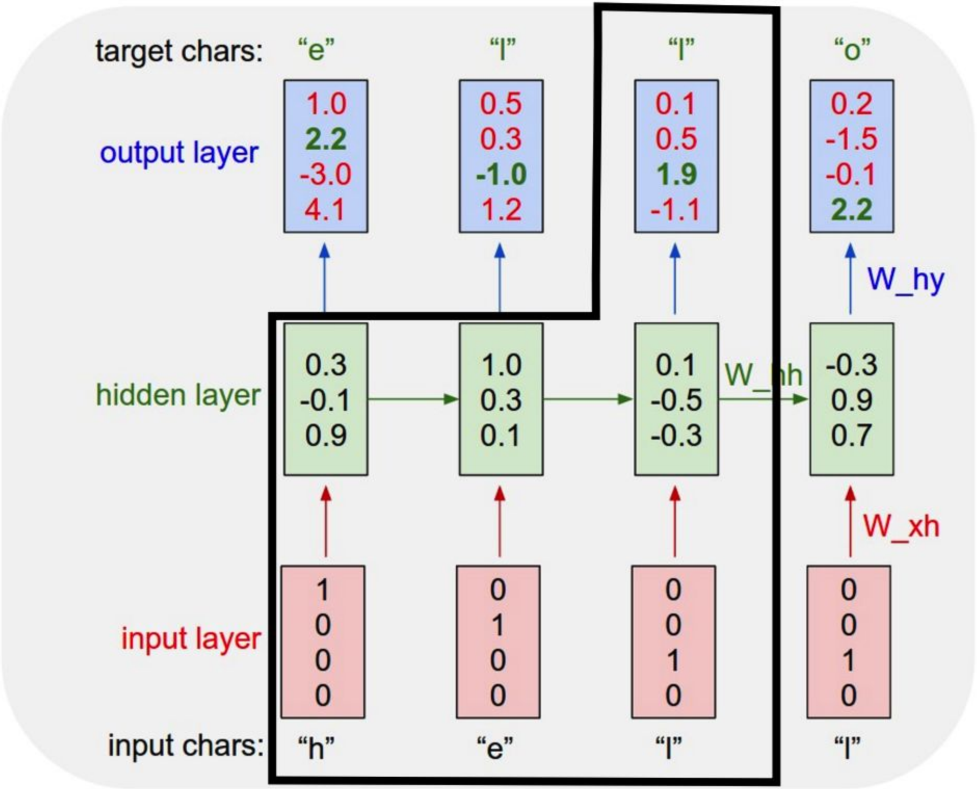
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



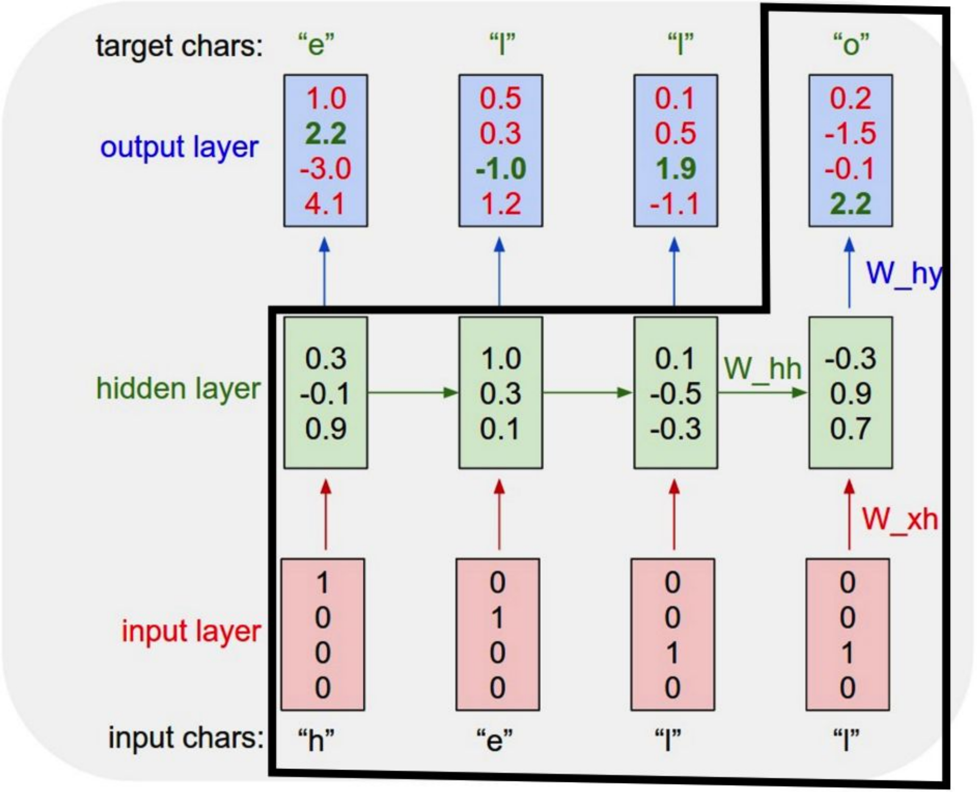
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]

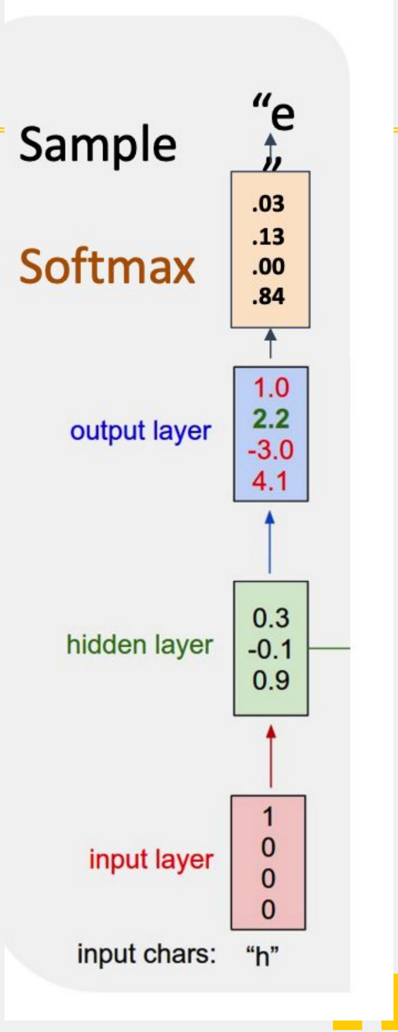


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

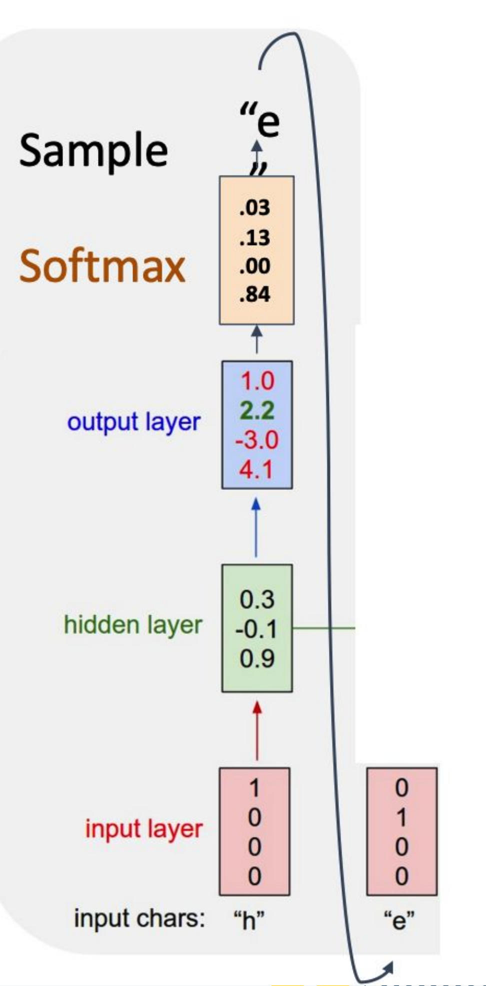


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

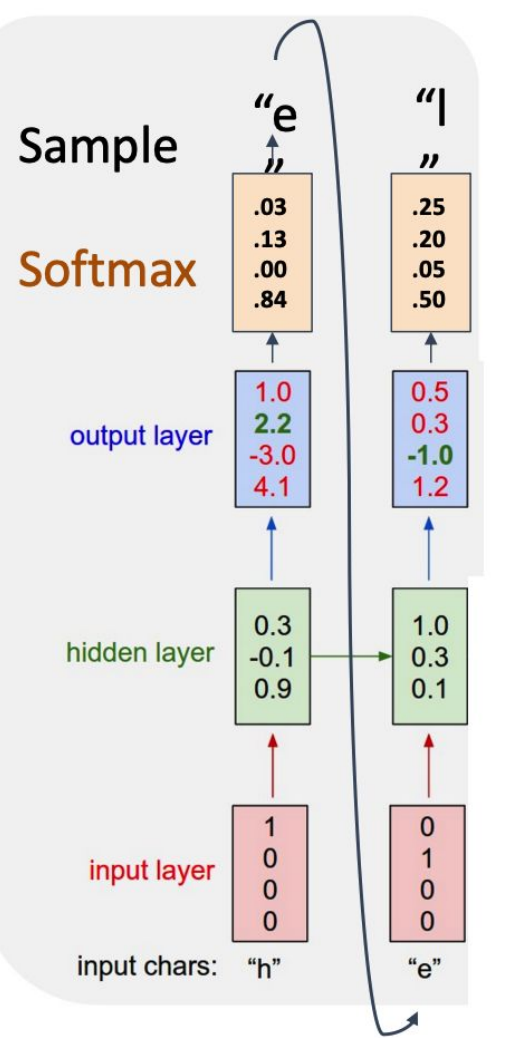


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

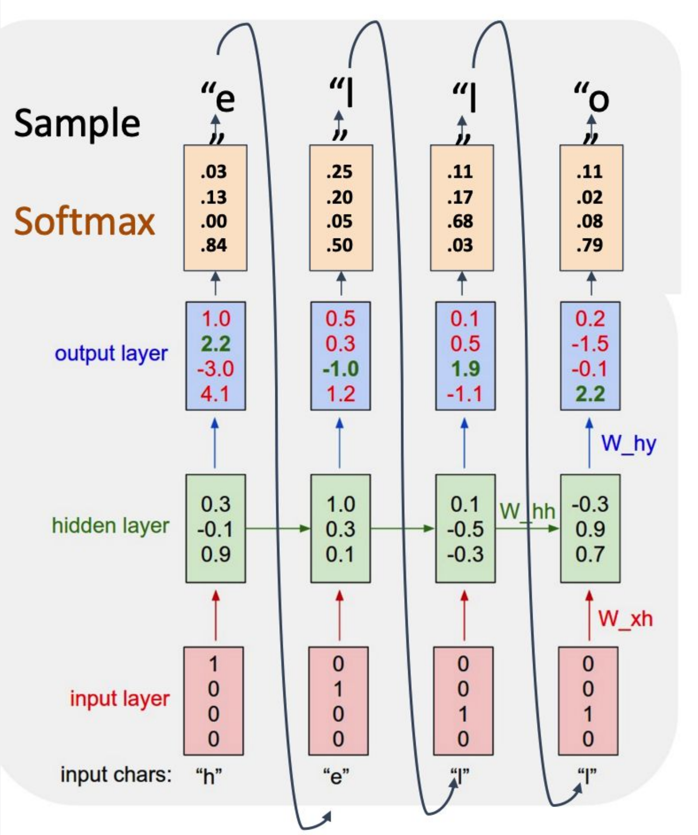


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

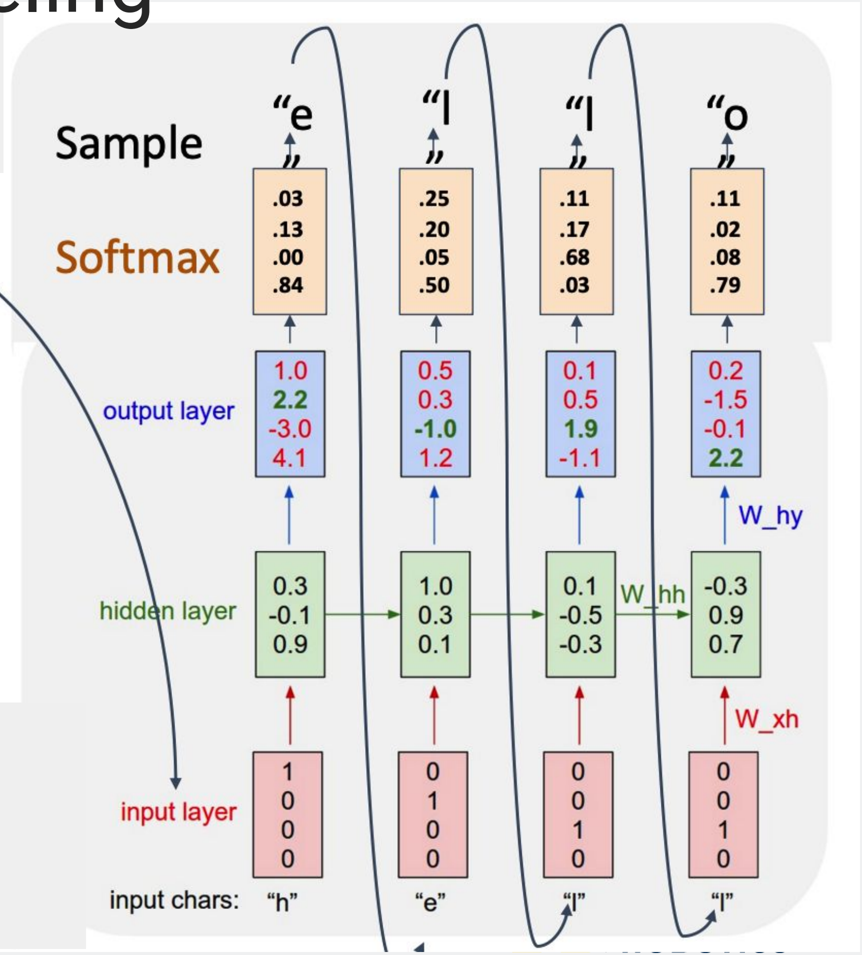


ROBOTICS

Example: Language Modeling

So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$



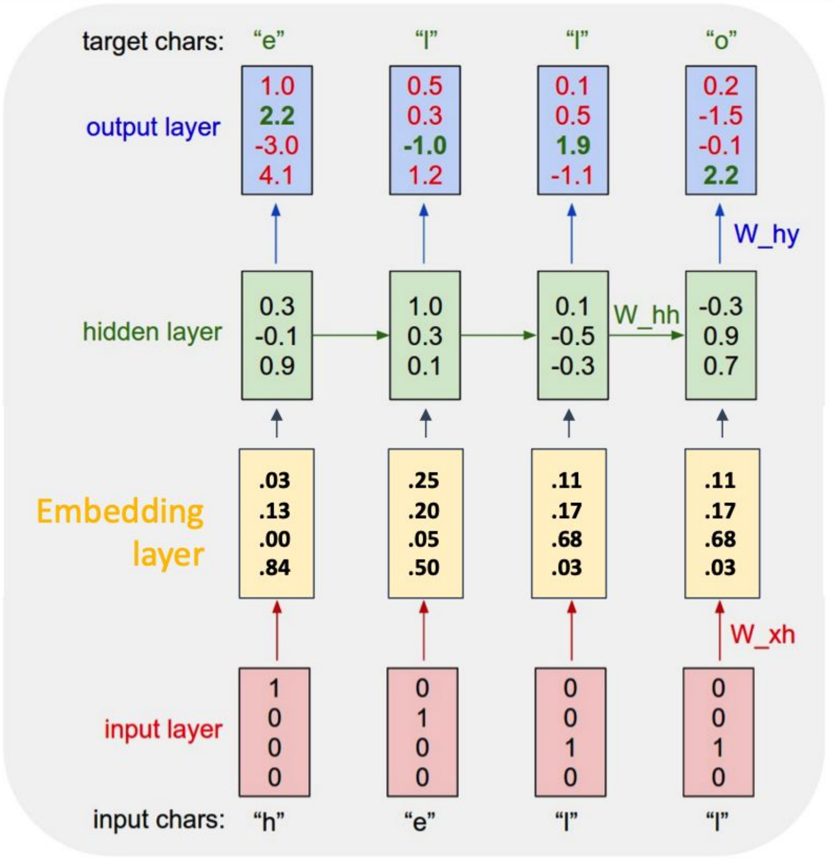
Example: Language Modeling

*NN likes dense real-valued vectors

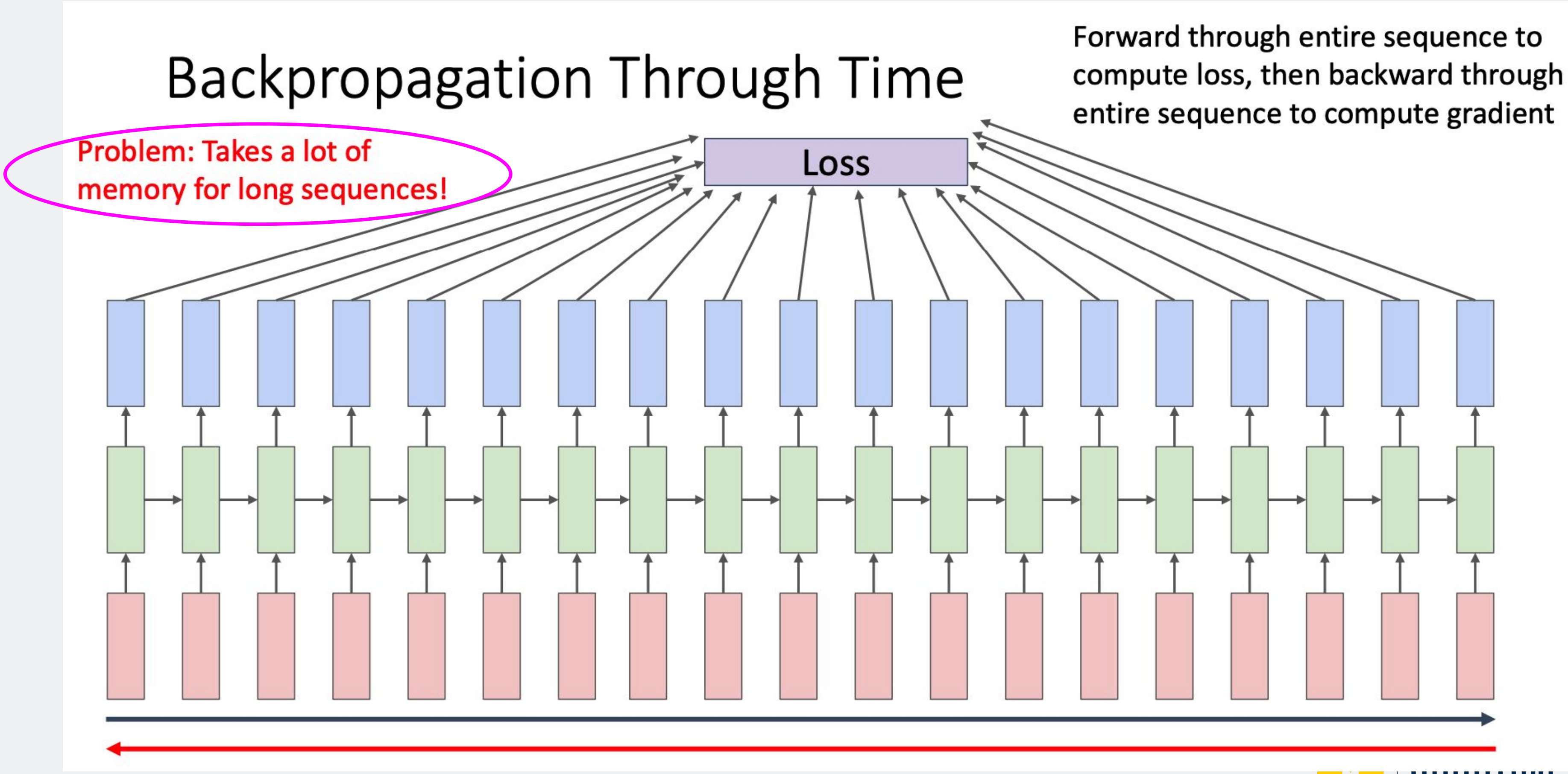
So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer

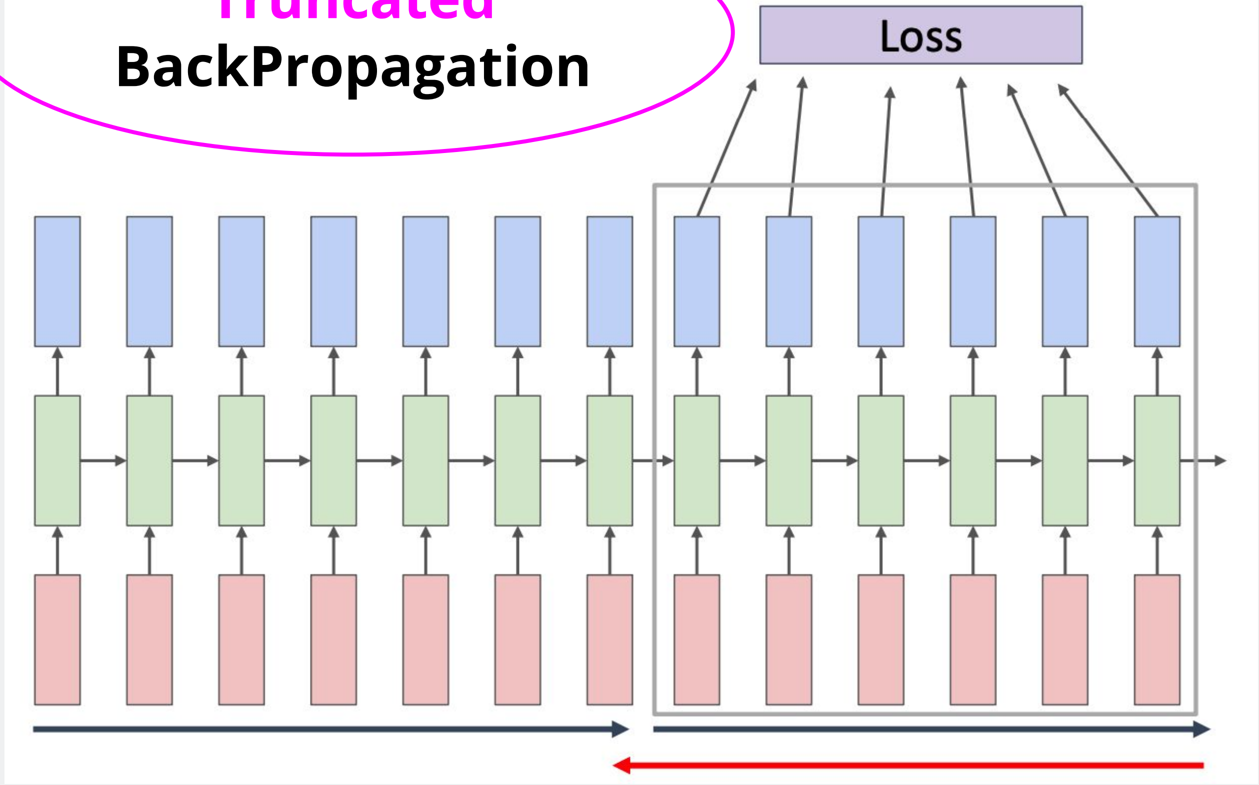


Example: Language Modeling (Backprop)



Example: Language Modeling (Backprop)

Truncated BackPropagation



- Run forward and backward through chunks of the sequence instead of whole sequence
- Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

MidTerm, P4, Final Project

P4 Reminder

https://deeprob.org/w25/projects/project4/
Due March 30, 2025

Two parts:

1. PoseCNN (see Lecture 13 for more hints)
2. Vision Transformer (this week)

Start NOW!!!

Final Project Teams Assigned

https://docs.google.com/spreadsheets/d/1FjWAjJ8p26xZmZaqsW4lew8H4iKe0FA78Q30eZ38g7A/edit?usp=sharing

Next TO-DOs: (final project total - 23% grade)
April 1st, 5-min poster "lightning talk", 5% grade

18% grade

April 22nd, final project showcase @FRB atrium
April 28th, final project (report, code, video/website) DUE

Final Project Teams Assigned

Reminder: Group Collaboration Policy (refer to Course Information Document)

- **"I participated and contributed to team discussions on each problem, and I attest to the integrity of each solution. Our team met as a group on [DATE(S)]. "**
- **"Contribution of Authors: [Team member A] did [Task XXX]; [Team members B and C] did [Task YYY]; [Team members A, B and C] did [ZZZ]. [All authors] [gave feedback on the software development, contributed to writing the report/making the demo presentation, and approved the final version for submission.]"** (*Modify the texts in brackets according to your specific team situation and member contribution. Ideally, each member/subset of members contributed to something unique, and all authors contributed to giving feedback and writing/making the final report/demo/presentations and approving the final version for submission.)

➡ Set and assign smaller goals to each person early!!!

GenAI: Permitted with disclosure (specify prompt, platform, results). Suggest verify and edit on top of GenAI results (if using).

https://genai.umich.edu/