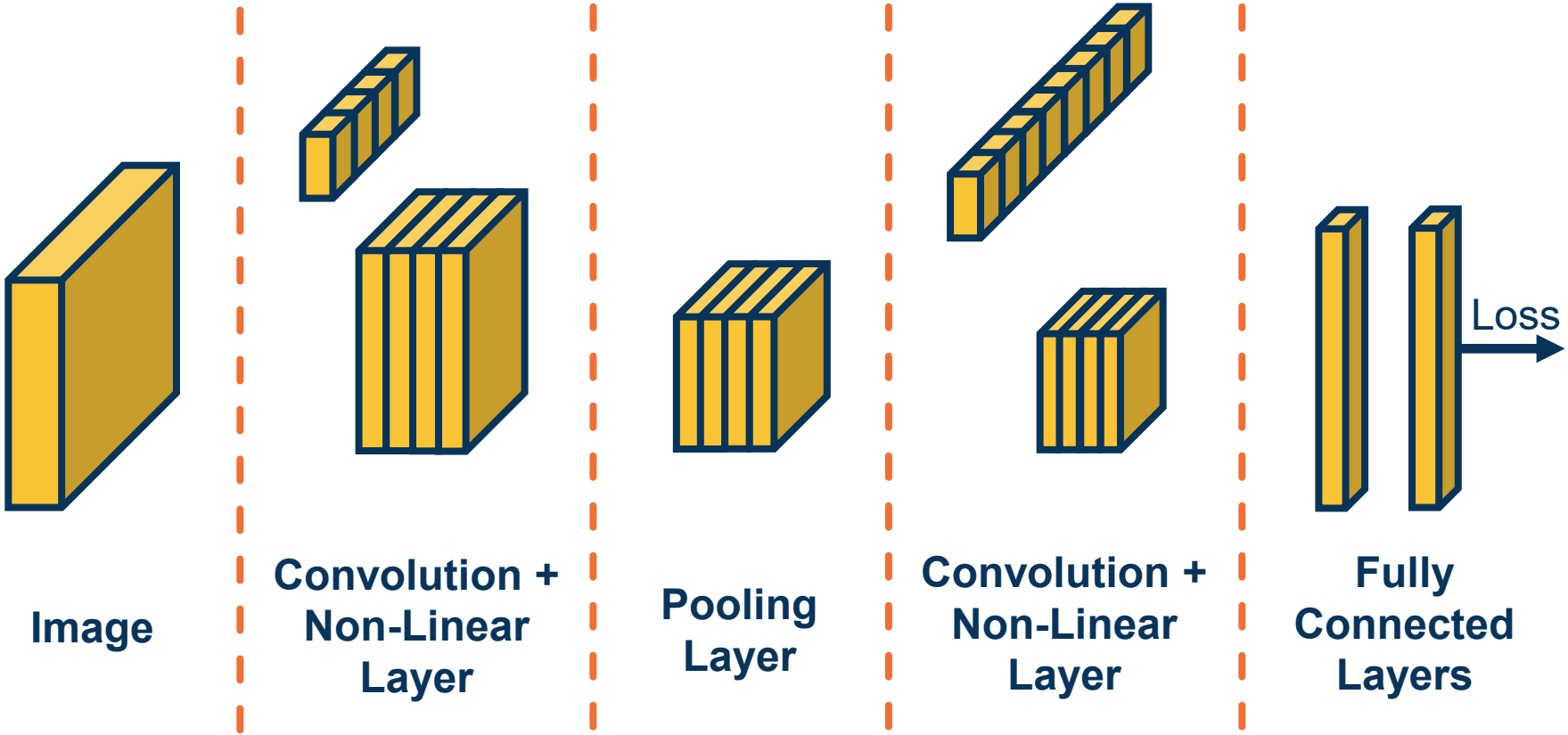
Topics:

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

CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 3 – Due 03/14**
  - Sequential models
- **FB/Meta Office hours Wed. 02/25 3pm EST!**
  - Language Models



Adding a Fully Connected Layer

Step 3: (Continue to) train on new dataset

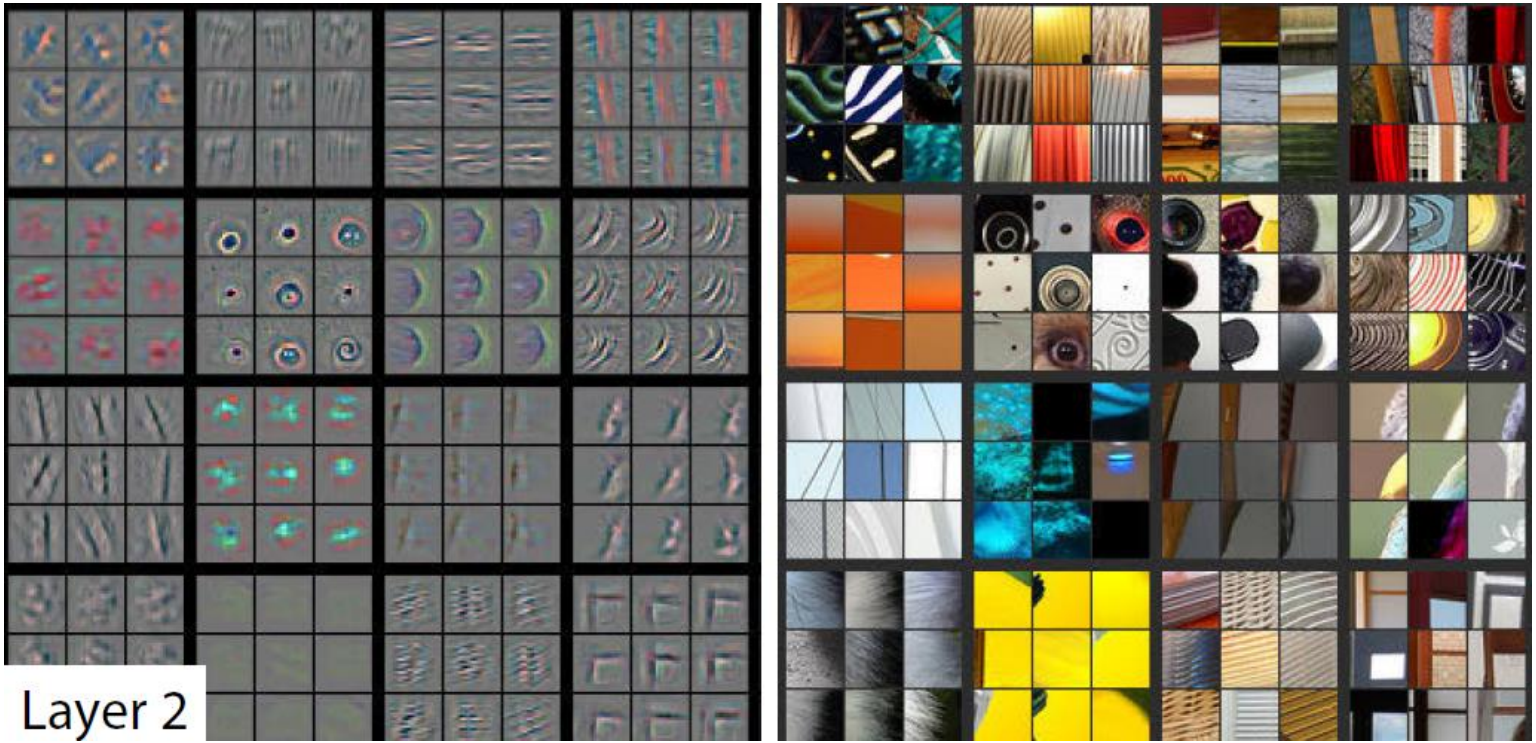
- **Finetune:** Update all parameters
- **Freeze** feature layer: Update only last layer weights (used when not enough data)

Replace last layer with new fully-connected for output nodes per new category

What do CNNs Learn?

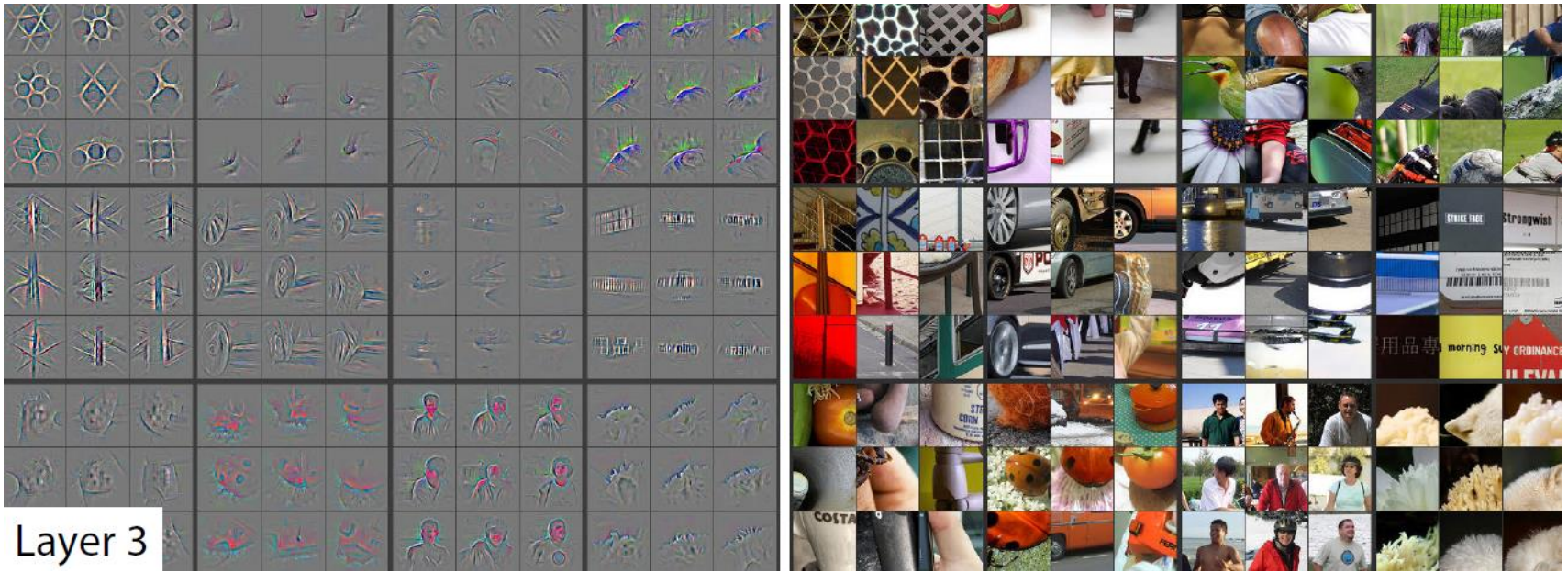
Georgia Tech

VGG Layer-by-Layer Visualization



From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

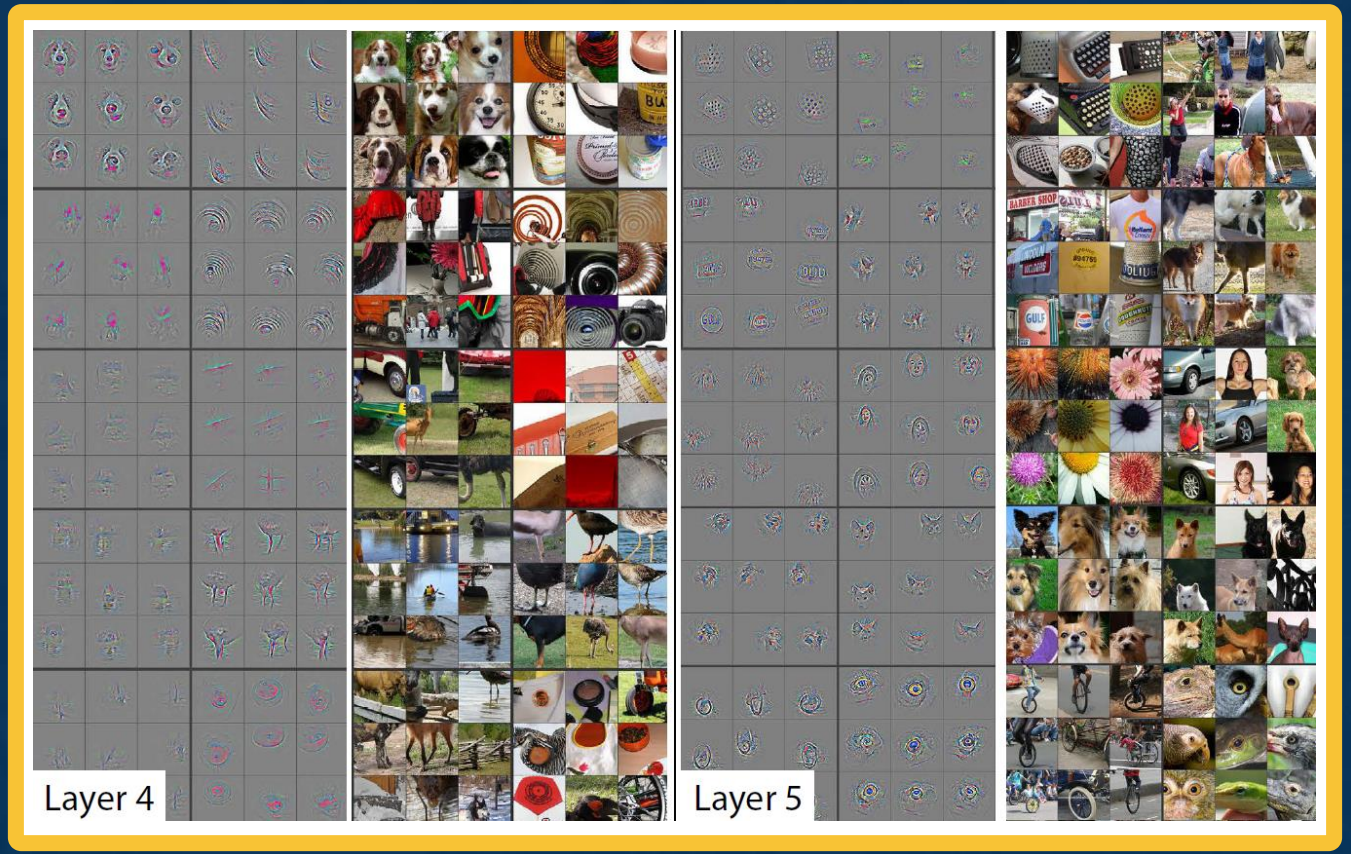
VGG Layer-by-Layer Visualization



From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

Georgia Tech

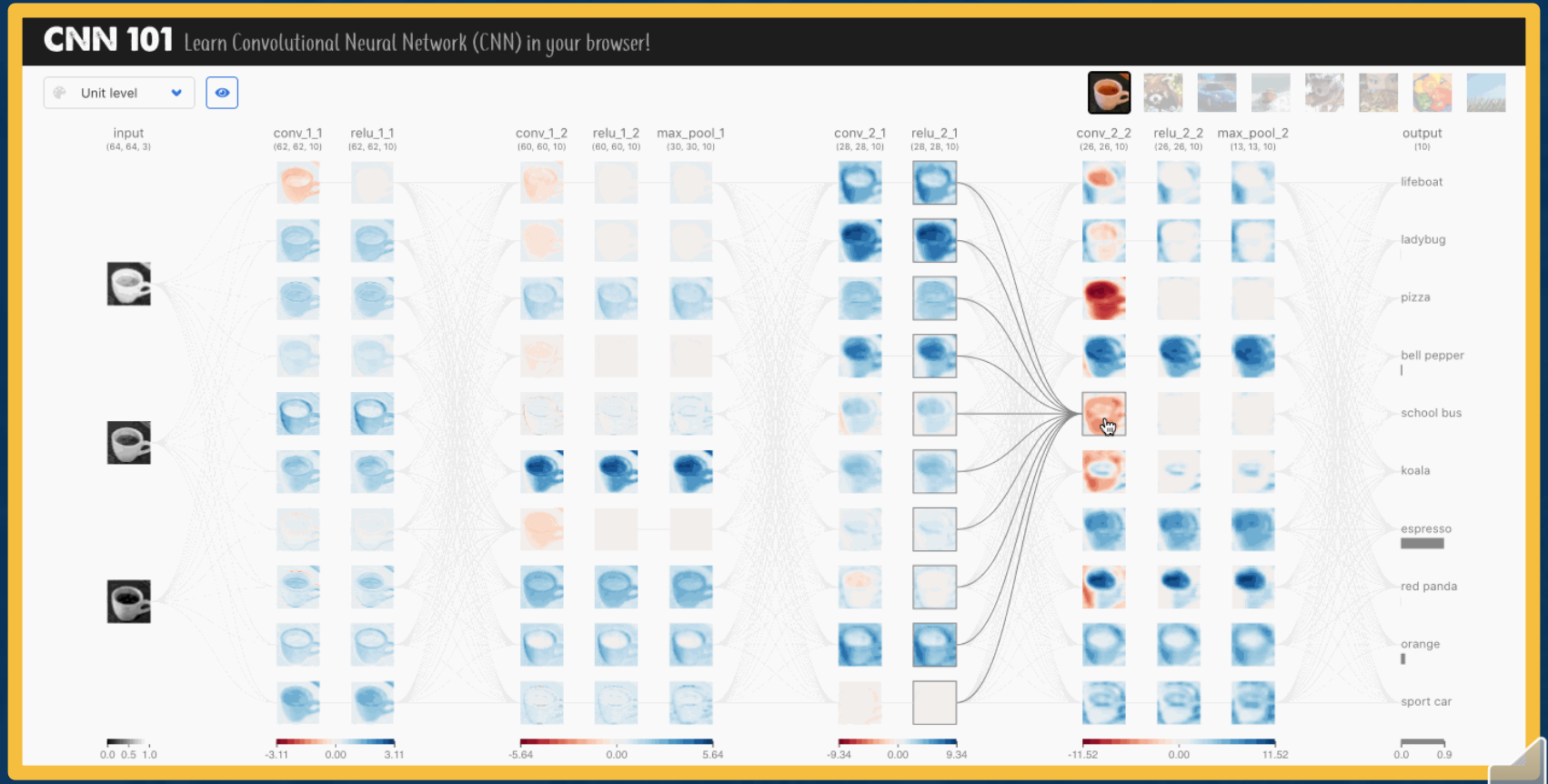
VGG Layer-by-Layer Visualization



From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

Georgia Tech

CNN101 and CNN Explainer



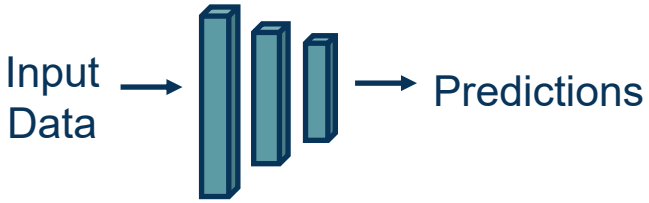
https://poloclub.github.io/cnn-explainer/

https://fredhohman.com/papers/cnn101

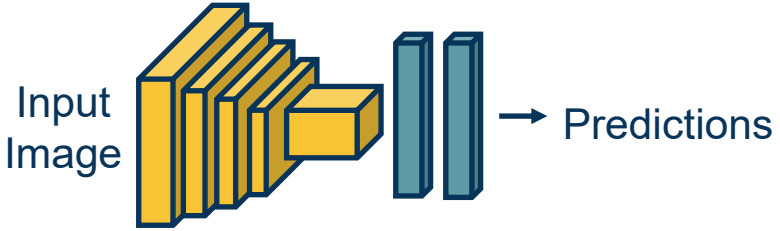
Georgia Tech

Module 3
Introduction

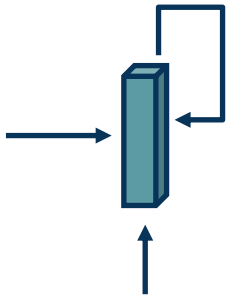
Georgia Tech



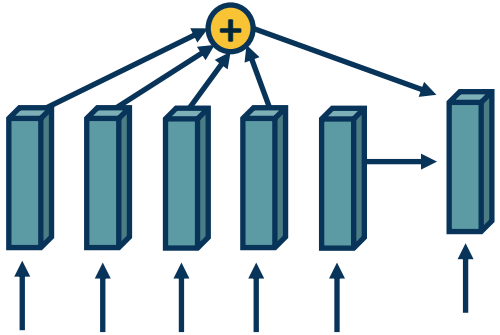
Fully Connected Neural Networks



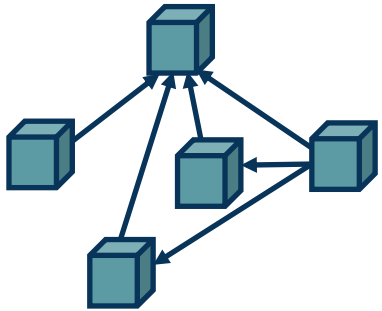
Convolutional Neural Networks



Recurrent Neural Networks



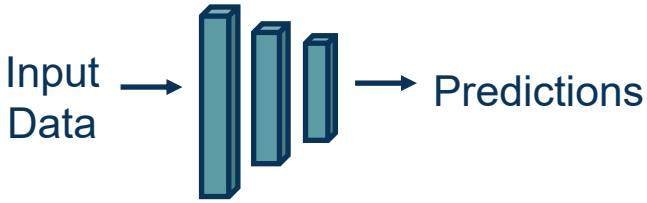
Attention-Based Networks



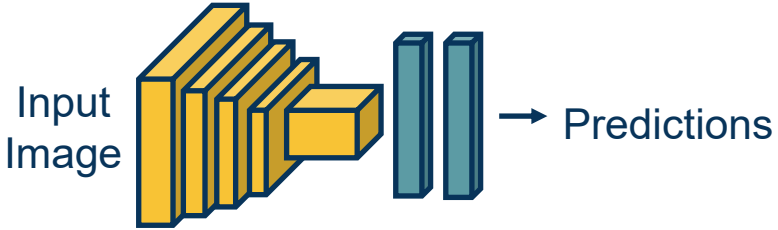
Graph-Based Networks

The Space of Architectures

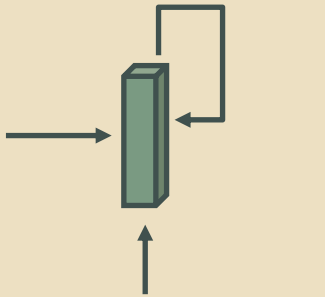
Georgia Tech



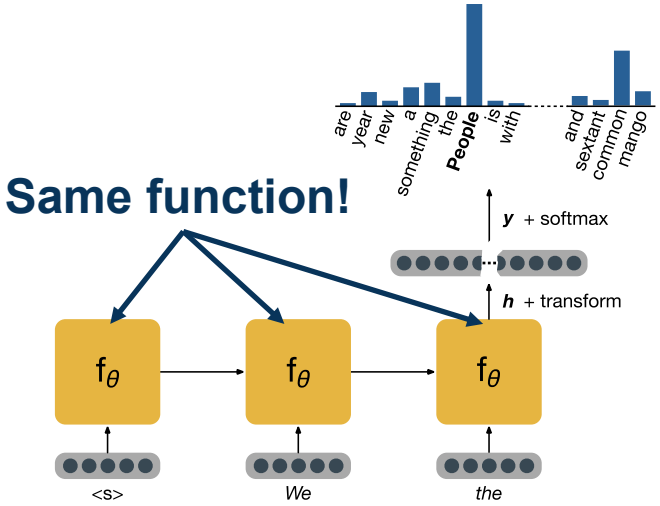
Fully Connected Neural Networks



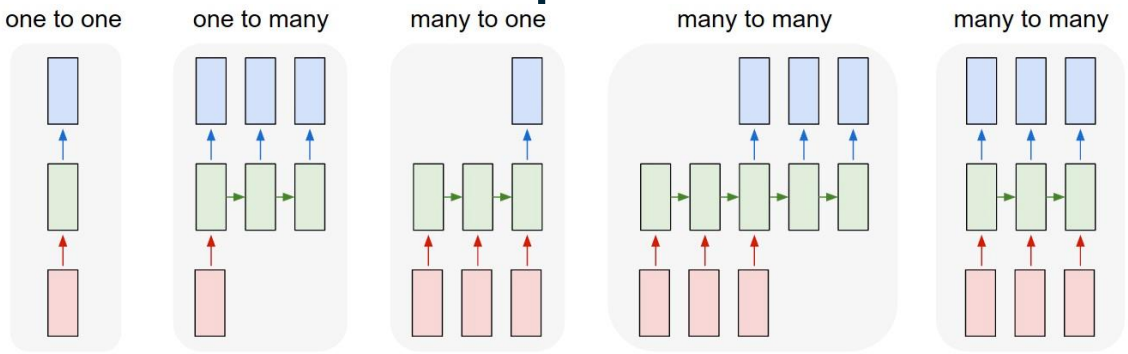
Convolutional Neural Networks



Recurrent Neural Networks



New Topic: RNNs



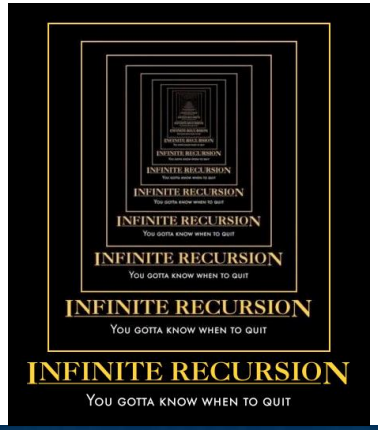


Image Credit: Andrej Karpathy

Why model sequences?

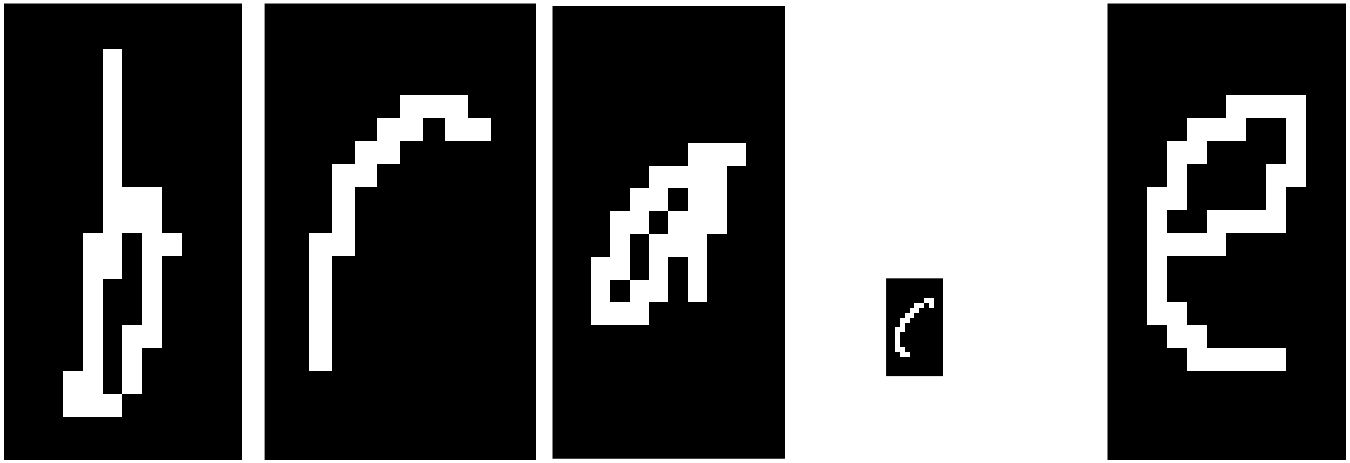


Figure Credit: Carlos Guestrin

Sequences are everywhere…

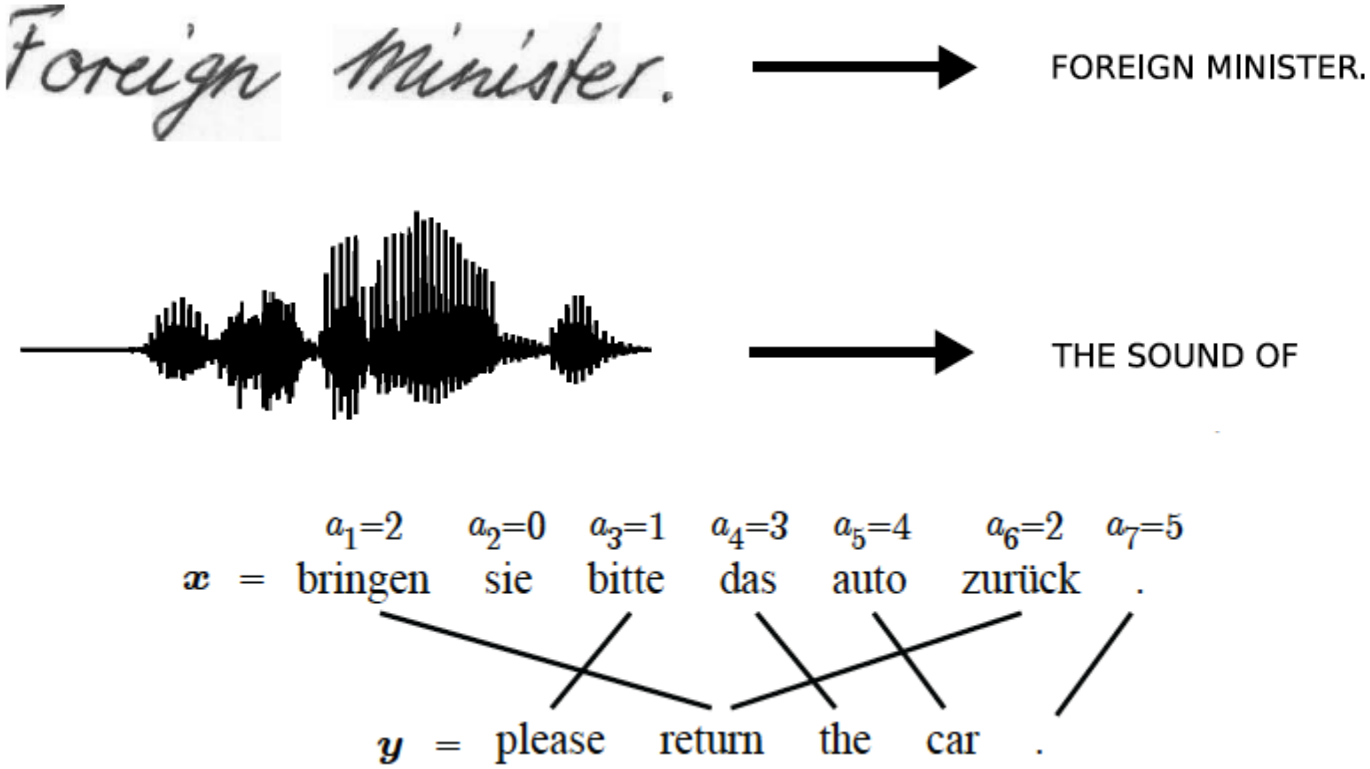
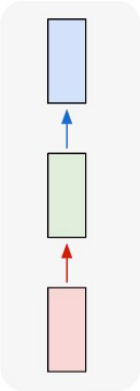


Image Credit: Alex Graves and Kevin Gimpel

Sequences in Input or Output?

- It's a spectrum…

one to one



Input: No sequence

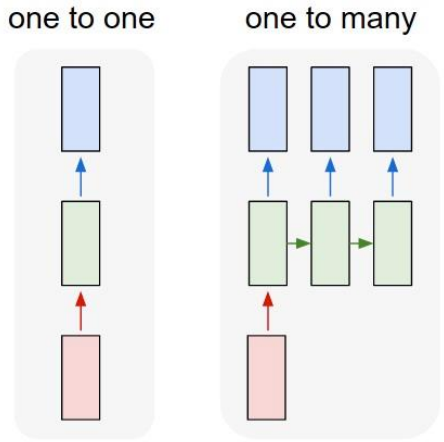
Output: No sequence

Example: "standard" classification / regression problems

Image Credit: Andrej Karpathy

Sequences in Input or Output?

- It's a spectrum…



Input: No sequence

Output: No sequence

Example: "standard" classification / regression problems

Input: No sequence

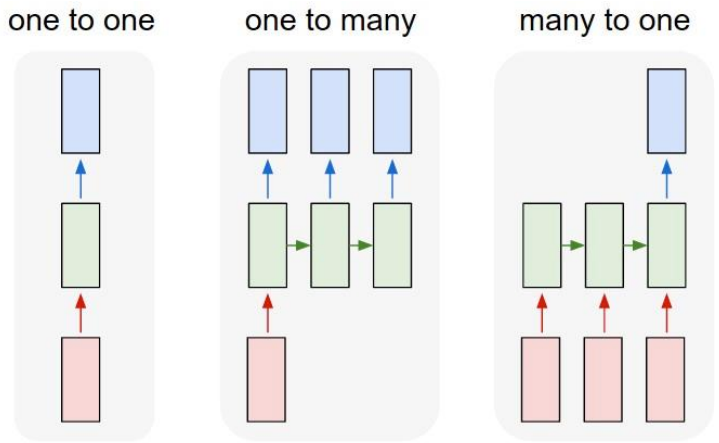
Output: Sequence

Example: Im2Caption

Image Credit: Andrej Karpathy

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

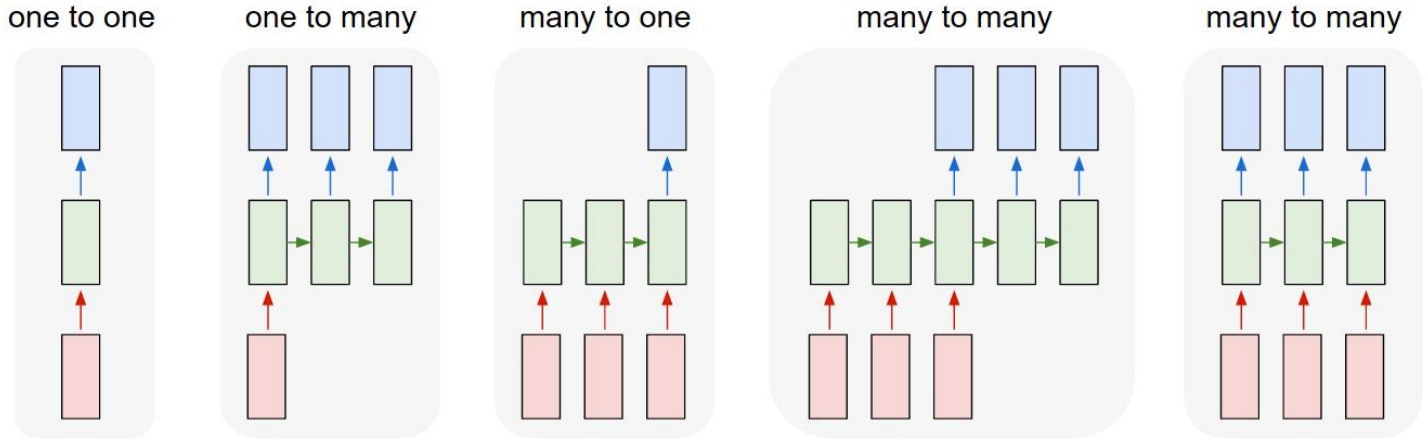
Input: Sequence

Output: No sequence

Example: sentence classification, multiple-choice question answering

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

What's wrong with MLPs?

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

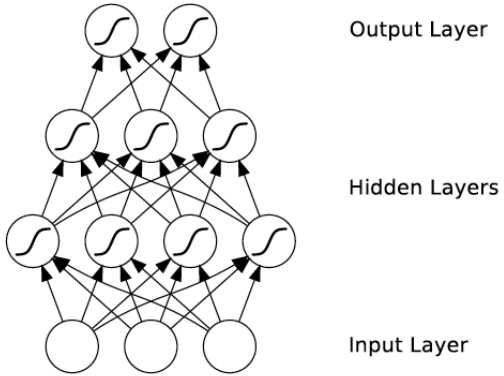


Image Credit: Alex Graves, book

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

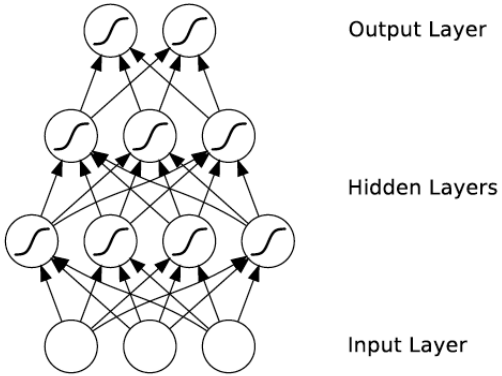


Image Credit: Alex Graves, book

3 Key Ideas

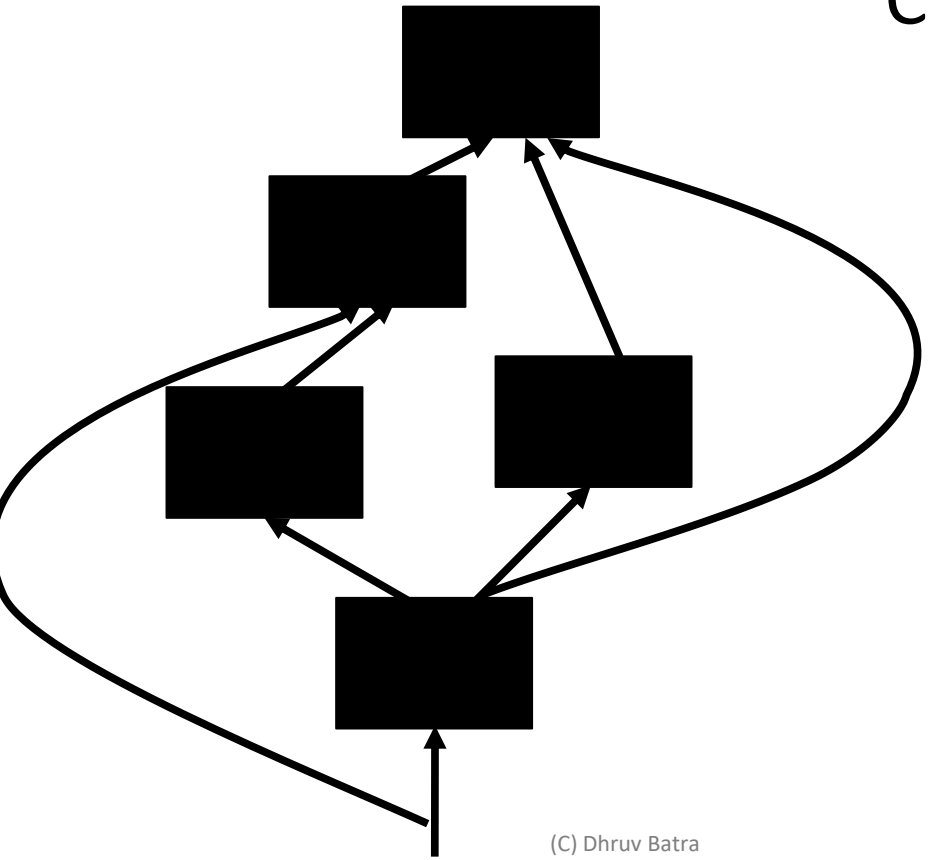
- The notion of memory (state)
  - We want to propagate information across the sequence
  - We will do this with *state*, represented by a vector (embedding/representation)
    - Key idea will be mixing new inputs with this state, to yield a new state
    - All represented as vector operations
  - Just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

3 Key Ideas

- The notion of memory (state)

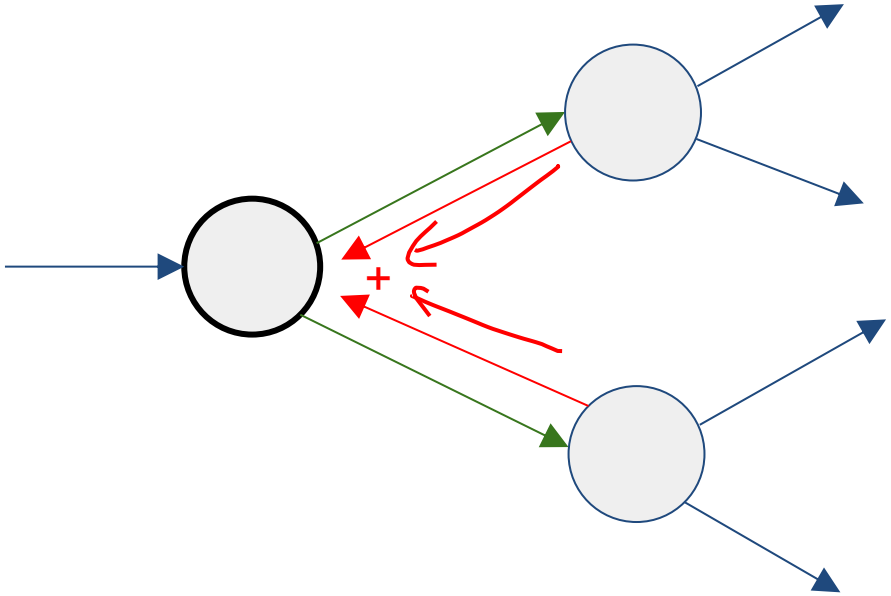
- Parameter Sharing
  - in computation graphs = adding gradients

Computational Graph



Slide Credit: Marc'Aurelio Ranzato

Gradients add at branches



3 Key Ideas

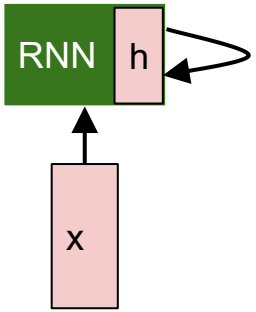
- The notion of memory (state)
- Parameter Sharing
  - in computation graphs = adding gradients
- “Unrolling”
  - in computation graphs with parameter sharing

New Words

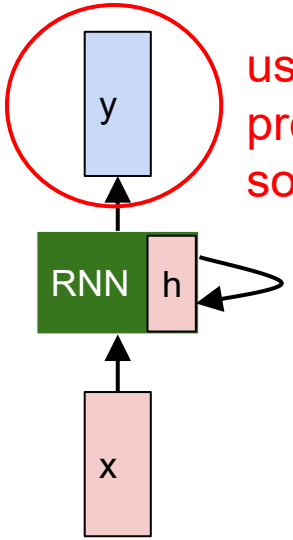
- Recurrent Neural Networks (RNNs)
- Recursive Neural Networks
  - General family; think graphs instead of chains
- Types:
  - “Vanilla” RNNs (Elman Networks)
  - Long Short Term Memory (LSTMs)
  - Gated Recurrent Units (GRUs)
  - …
- Algorithms
  - BackProp Through Time (BPTT)
  - BackProp Through Structure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

Recurrent Neural Network

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



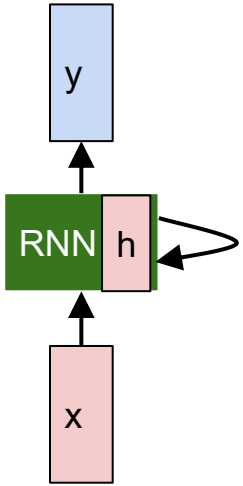
Recurrent Neural Network



usually want to predict a vector at some time steps

(Vanilla) Recurrent Neural Network

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



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

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

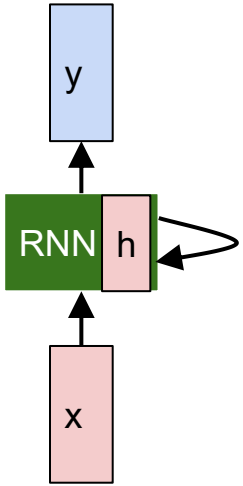
$$\downarrow$$

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

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

(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:

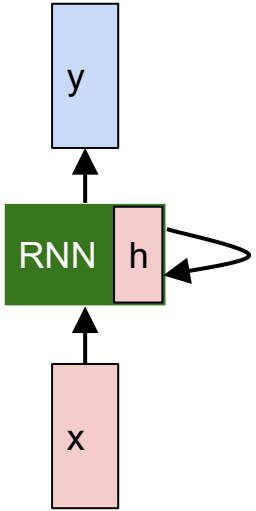
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

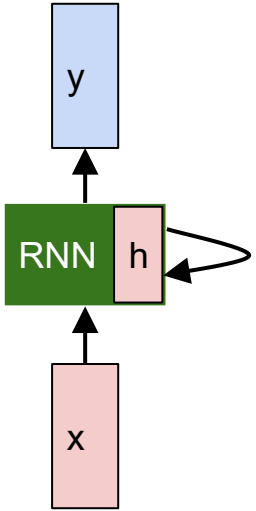
input vector at some time step



Recurrent Neural Network

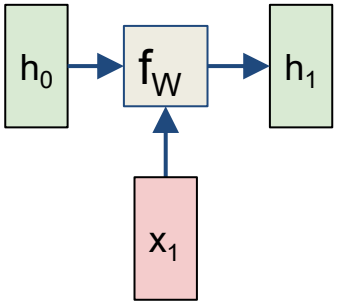
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

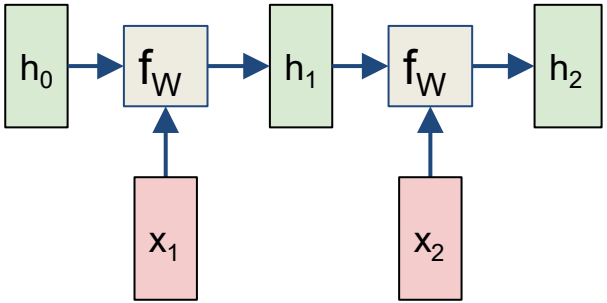


Notice: the same function and the same set of parameters are used at every time step.

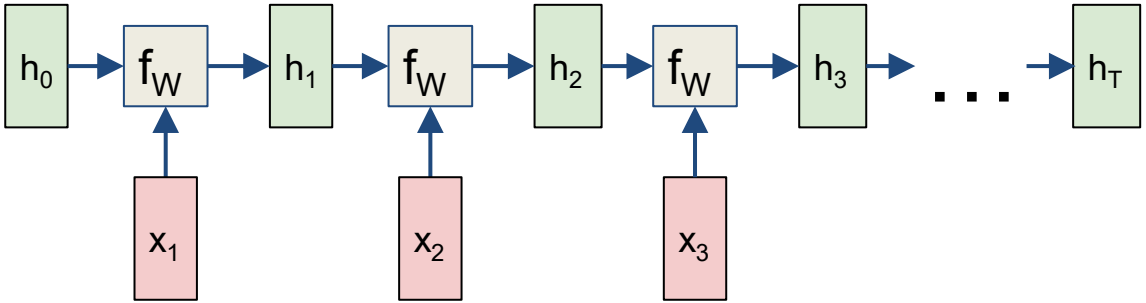
RNN: Computational Graph



RNN: Computational Graph

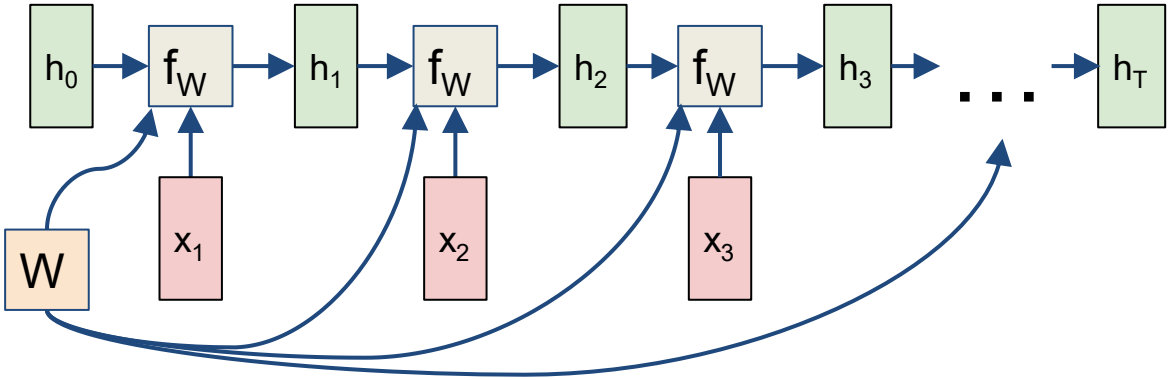


RNN: Computational Graph

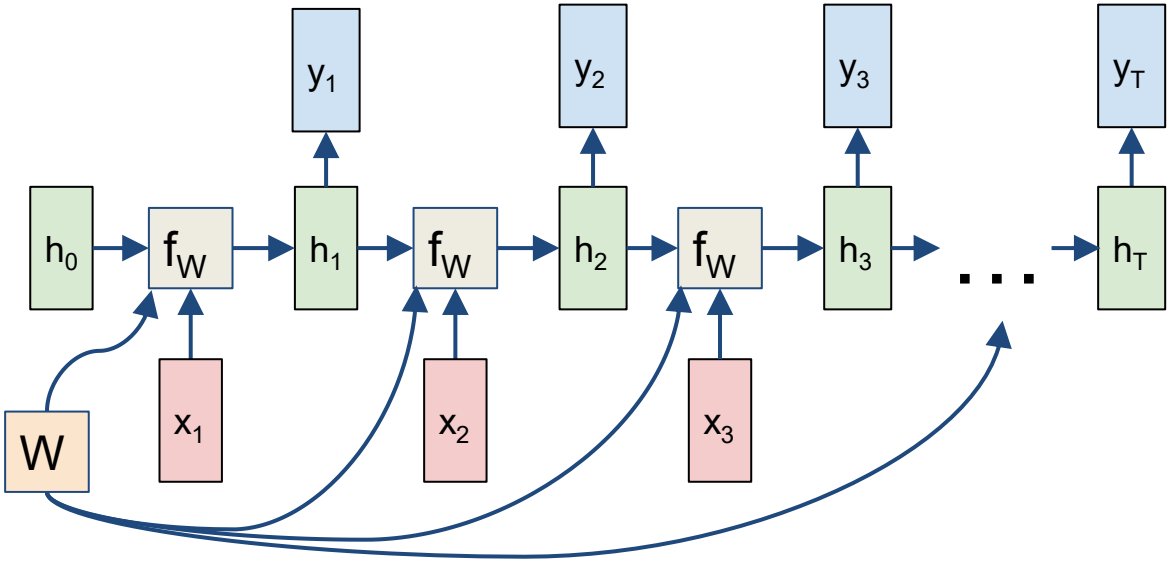


RNN: Computational Graph

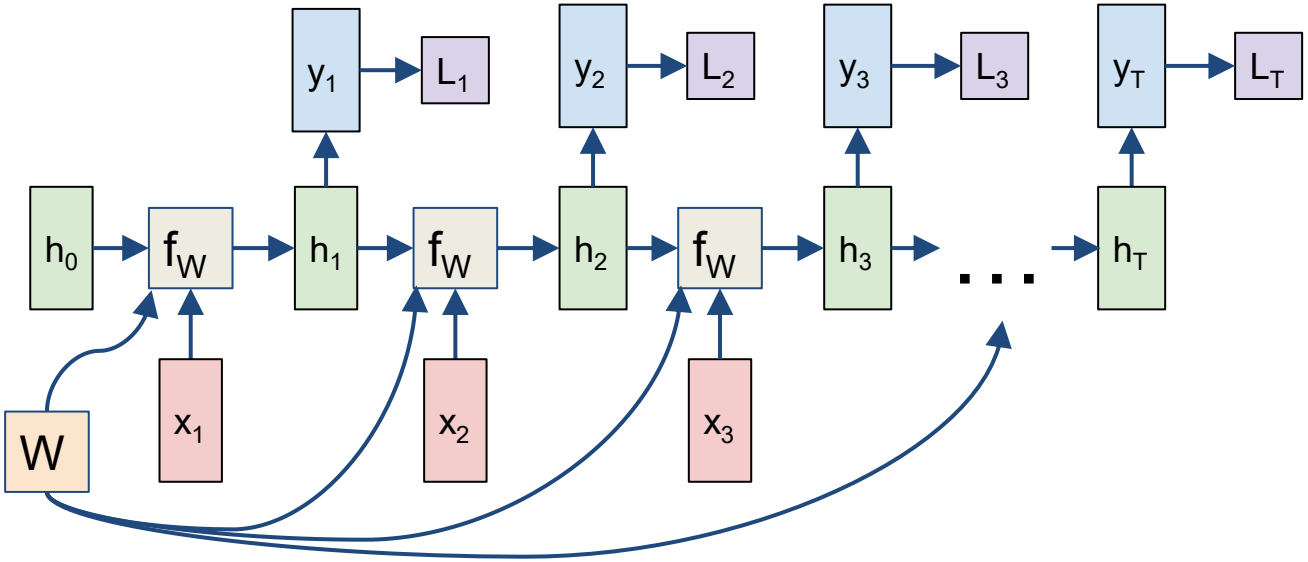
Re-use the same weight matrix at every time-step



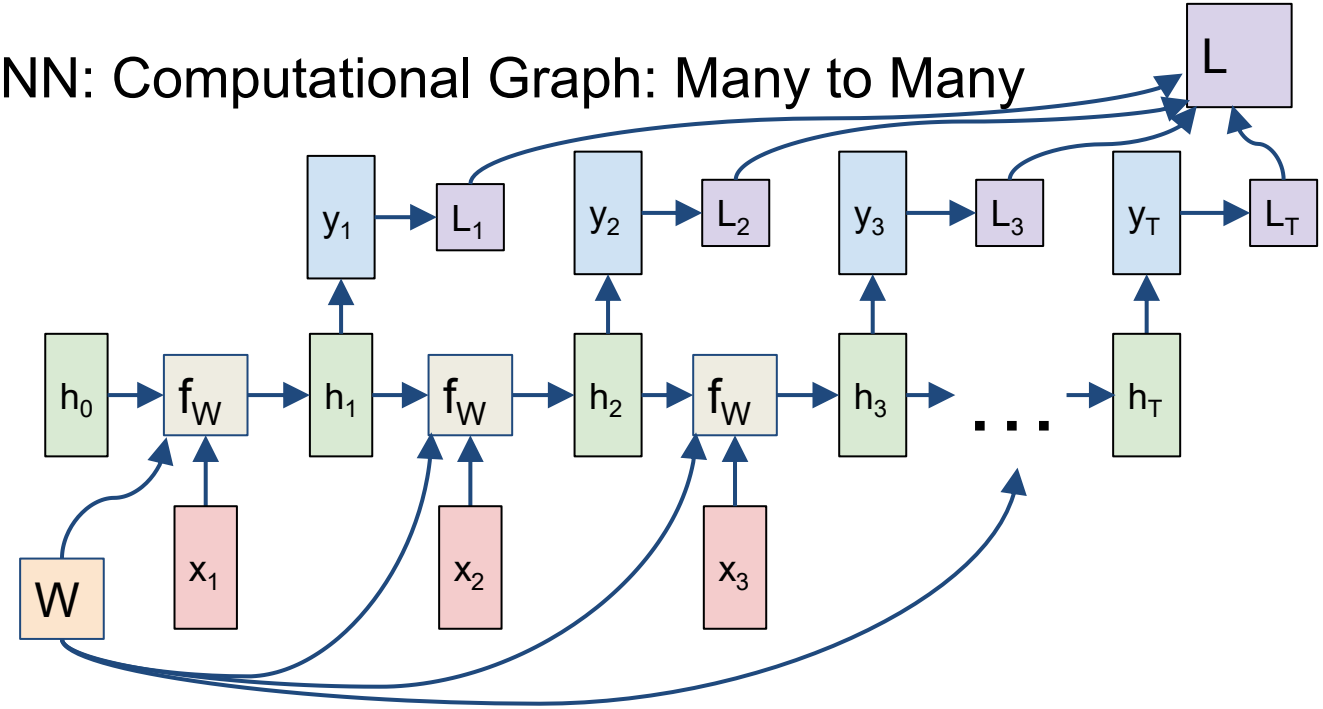
RNN: Computational Graph: Many to Many



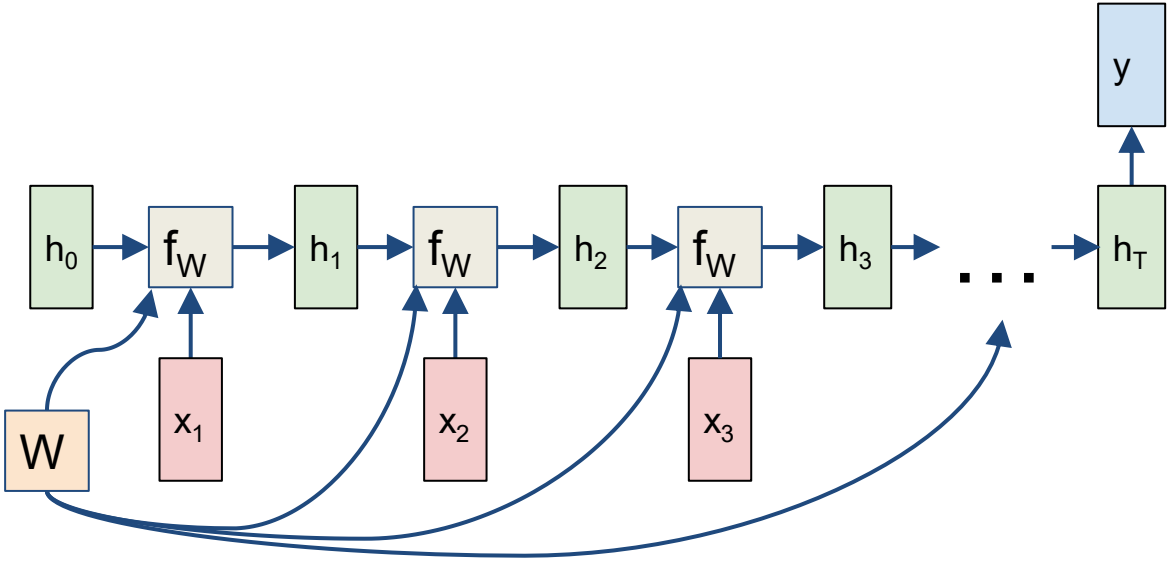
RNN: Computational Graph: Many to Many



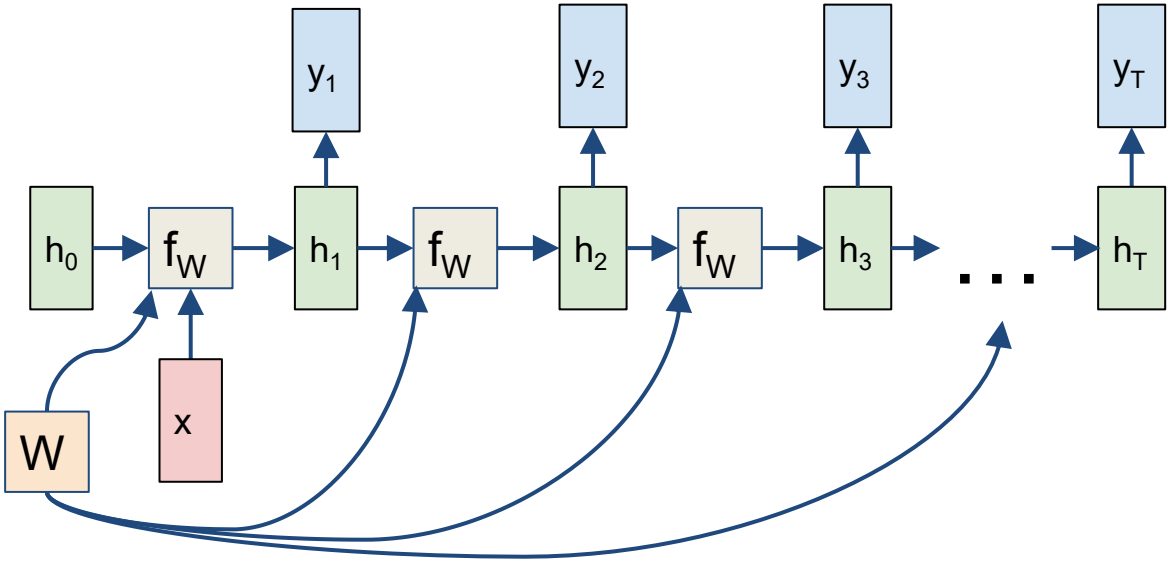
RNN: Computational Graph: Many to Many



RNN: Computational Graph: Many to One

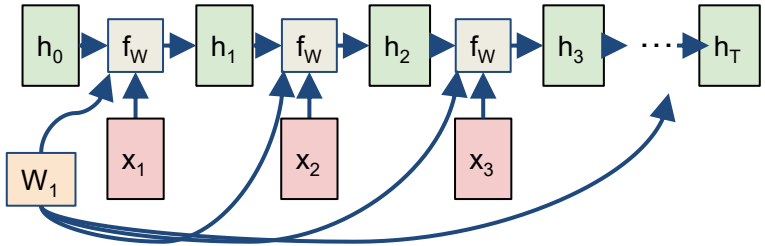


RNN: Computational Graph: One to Many



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

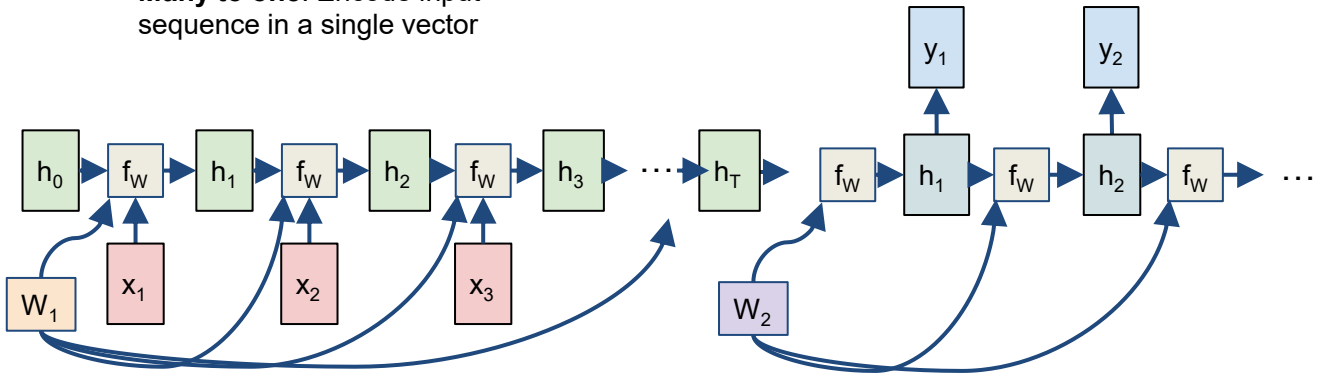
Many to one: Encode input sequence in a single vector



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"

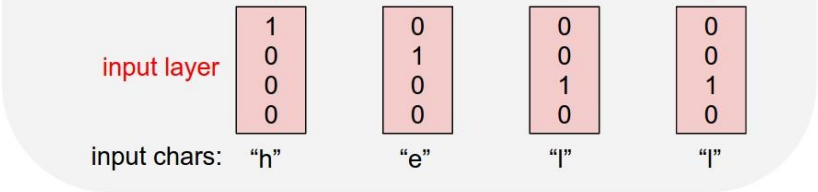
- **Inference:** Just generate me new text
  - Can condition on some initial input (**prompt**)

```
#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”

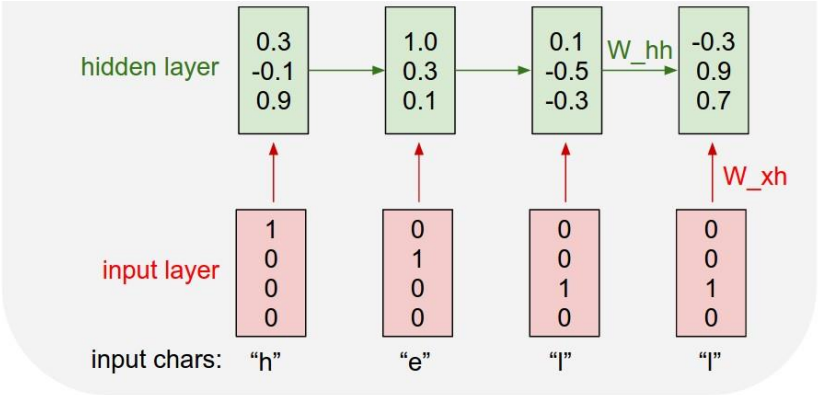


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

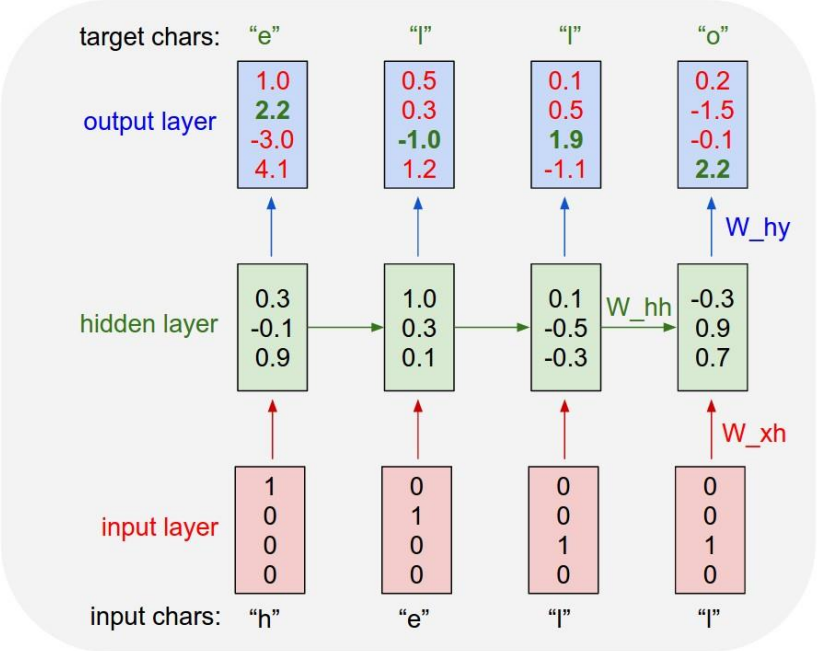
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

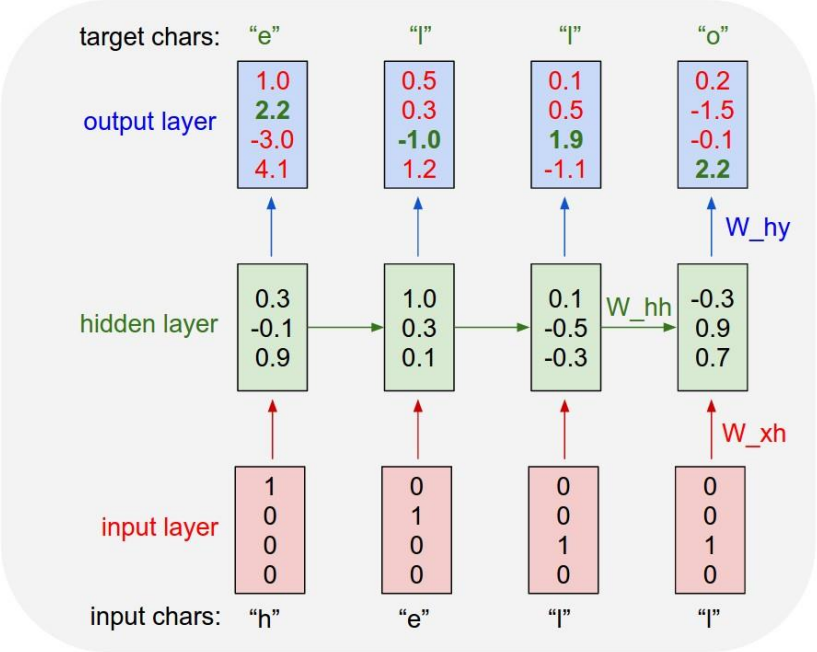


Training Time: MLE / “Teacher Forcing”

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

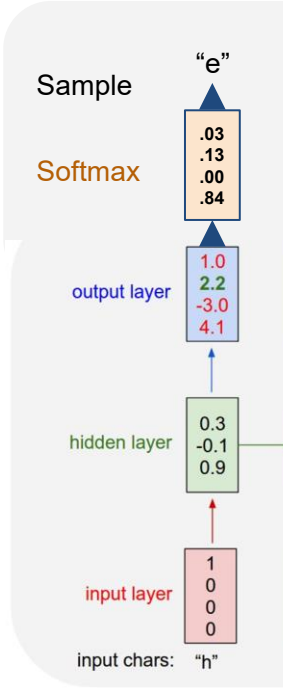


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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

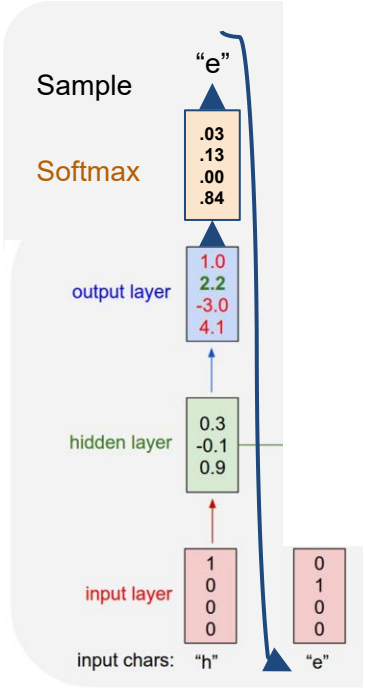


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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

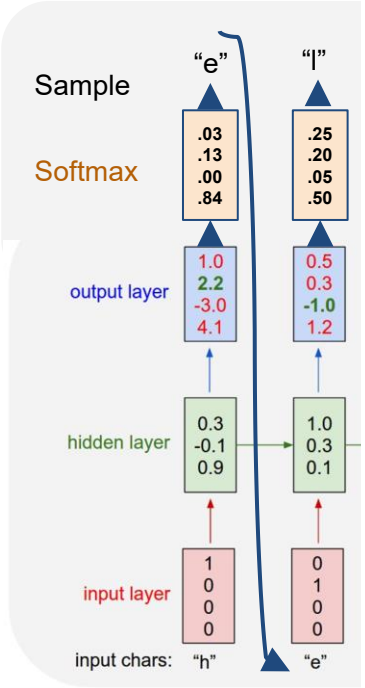


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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

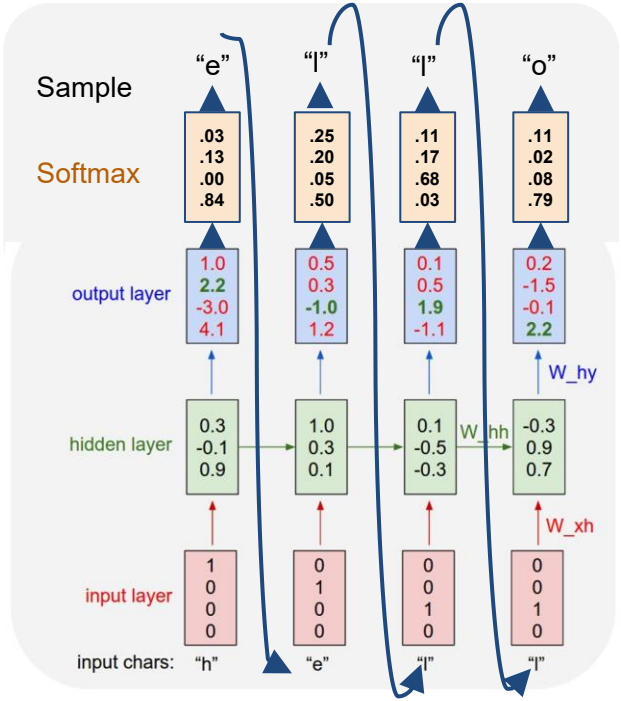


Test Time: Sample / Argmax / Beam Search

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

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

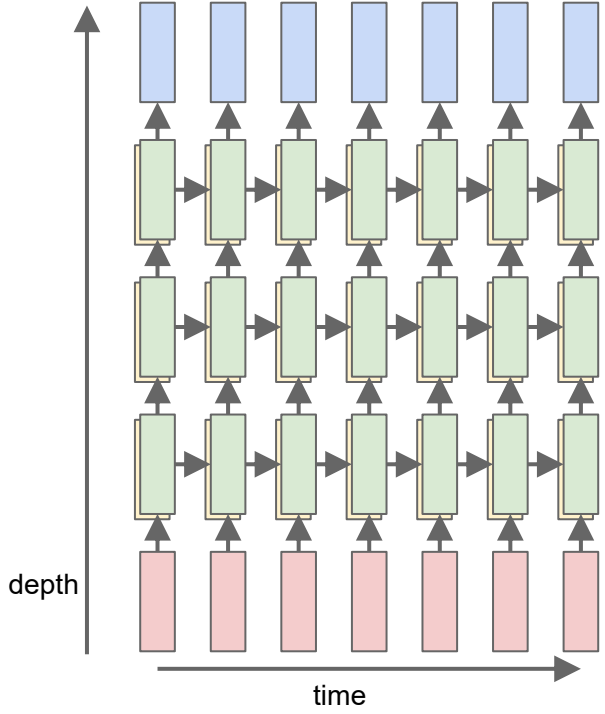


Can also feed in predictions during training (student forcing)

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;
}
```