

Recurrent Neural Networks: An Introduction

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad



Review: Questions

How to find bias in a model?

Review: Questions

How to find bias in a model?

Change a particular attribute/feature in question, and see if the prediction changes!

Acknowledgements

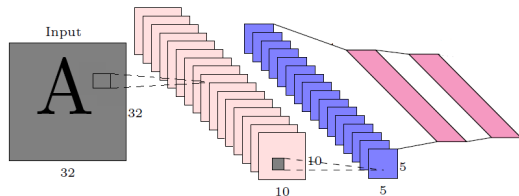
- This lecture's slides are based on:
 - **Lecture 10** of Stanford's **CS231n** course Fei-Fei Li
 - **Lecture 13** of IIT Madras' **CS7015** course by Mitesh Khapra

Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed

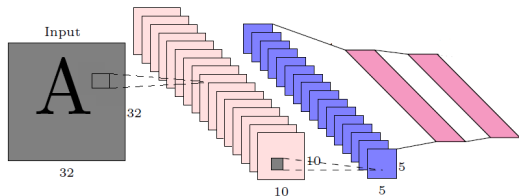
Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed
- E.g., we fed fixed size (32×32) images to convolutional neural networks for image classification



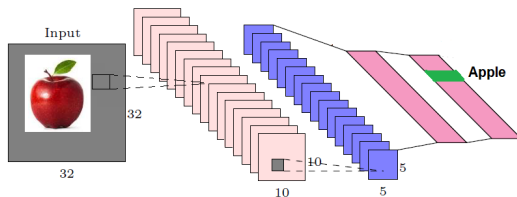
Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed
- E.g., we fed fixed size (32×32) images to convolutional neural networks for image classification
- Each input to network was independent of previous or future inputs



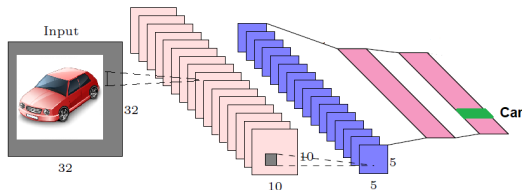
Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed
- E.g., we fed fixed size (32×32) images to convolutional neural networks for image classification
- Each input to network was independent of previous or future inputs
- Computations, outputs and decisions for two successive images are completely independent of each other



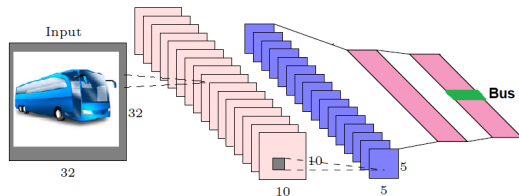
Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed
- E.g., we fed fixed size (32×32) images to convolutional neural networks for image classification
- Each input to network was independent of previous or future inputs
- Computations, outputs and decisions for two successive images are completely independent of each other



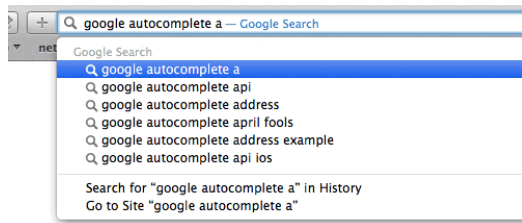
Sequence Learning Problems

- In feedforward and convolutional neural networks, size of the input was always fixed
- E.g., we fed fixed size (32×32) images to convolutional neural networks for image classification
- Each input to network was independent of previous or future inputs
- Computations, outputs and decisions for two successive images are completely independent of each other



Sequence Learning Problems

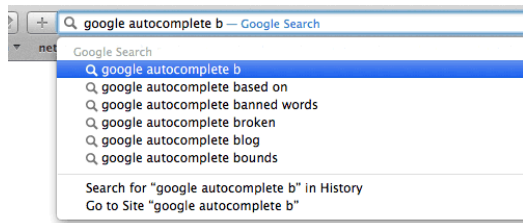
- Consider task of text auto completion



Credit: John Johnston

Sequence Learning Problems

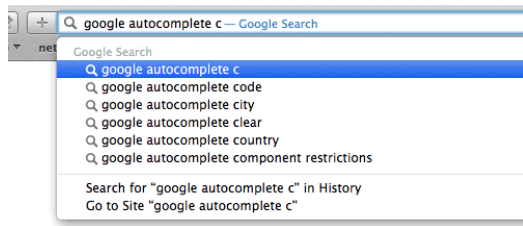
- Consider task of text auto completion



Credit: John Johnston

Sequence Learning Problems

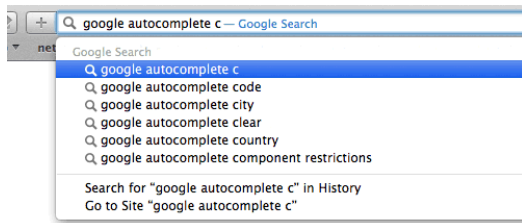
- Consider task of text auto completion
- Successive inputs are no longer independent!



Credit: John Johnston

Sequence Learning Problems

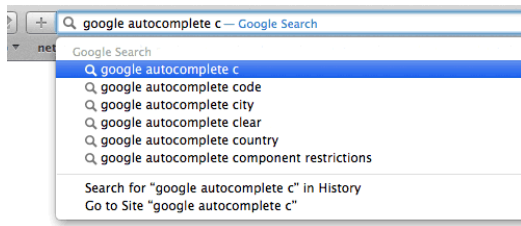
- Consider task of text auto completion
- Successive inputs are no longer independent!
- Length of inputs and number of predictions you need to make are not fixed



Credit: John Johnston

Sequence Learning Problems

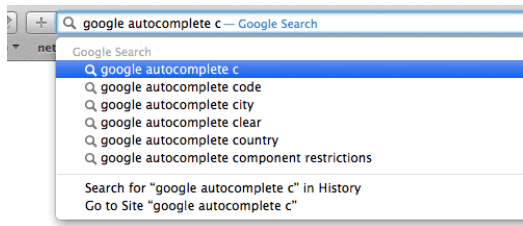
- Consider task of text auto completion
- Successive inputs are no longer independent!
- Length of inputs and number of predictions you need to make are not fixed
- Underlying model is performing same task across all contexts (*input*: character, *output*: character)



Credit: John Johnston

Sequence Learning Problems

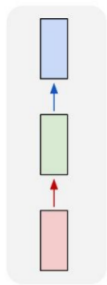
- Consider task of text auto completion
- Successive inputs are no longer independent!
- Length of inputs and number of predictions you need to make are not fixed
- Underlying model is performing same task across all contexts (*input*: character, *output*: character)
- Known as **sequence learning problems**



Credit: John Johnston

Recurrent Neural Networks: Variants

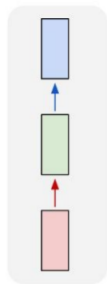
one to one



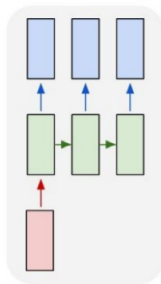
Vanilla Neural Networks

Recurrent Neural Networks: Variants

one to one



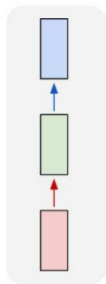
one to many



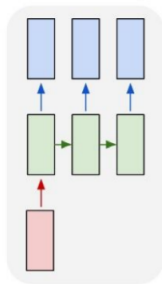
↖ e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Variants

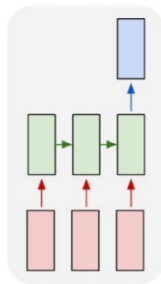
one to one



one to many



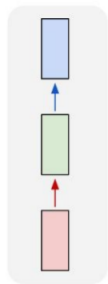
many to one



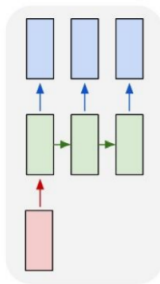
↖ e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Variants

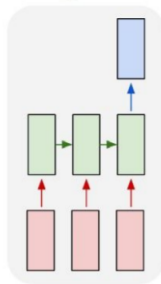
one to one



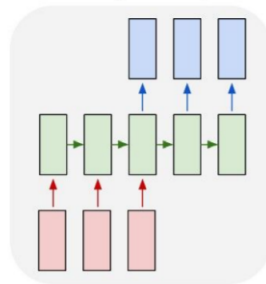
one to many



many to one



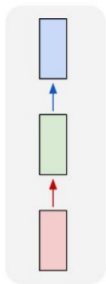
many to many



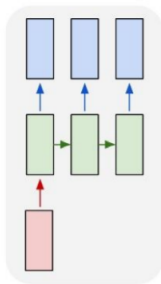
↖ E.g. **Video Captioning**
Sequence of video frames ->
caption

Recurrent Neural Networks: Variants

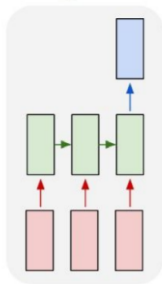
one to one



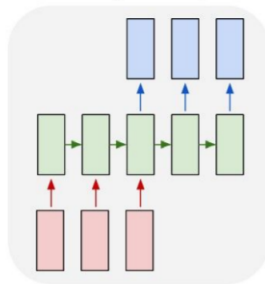
one to many



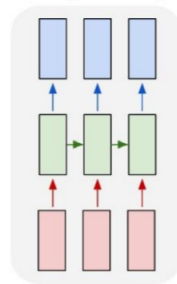
many to one




many to many



many to many



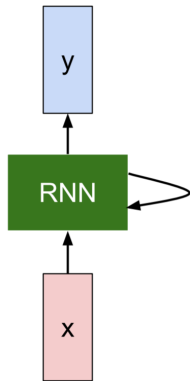
e.g. **Video classification on frame level**



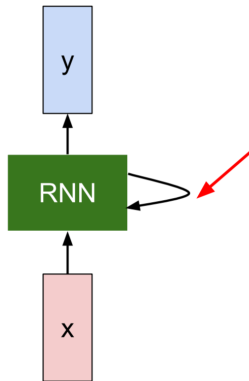
How do we model such tasks involving sequences?

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

Recurrent Neural Network

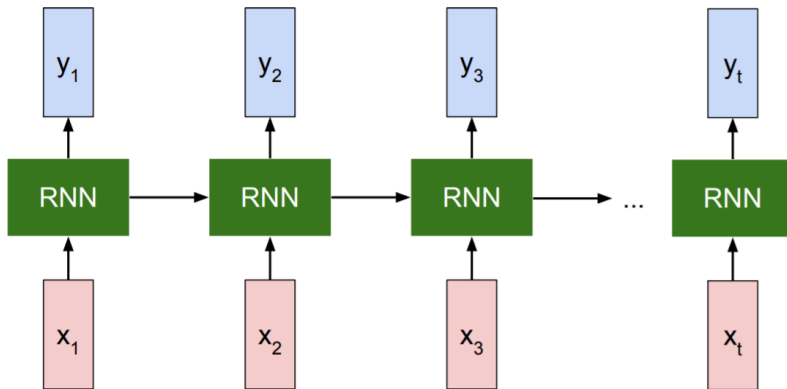


Recurrent Neural Network



Key idea: RNNs have an “internal state” that is updated as a sequence is processed

Recurrent Neural Network: Unfolded



Recurrent Neural Network

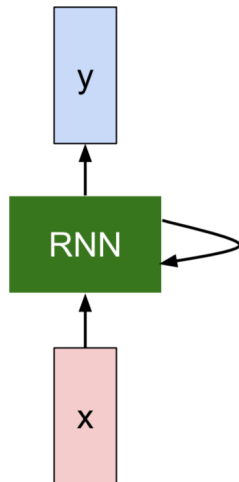
We can process a sequence of vectors x by applying a **recurrence formula** at every time step:

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

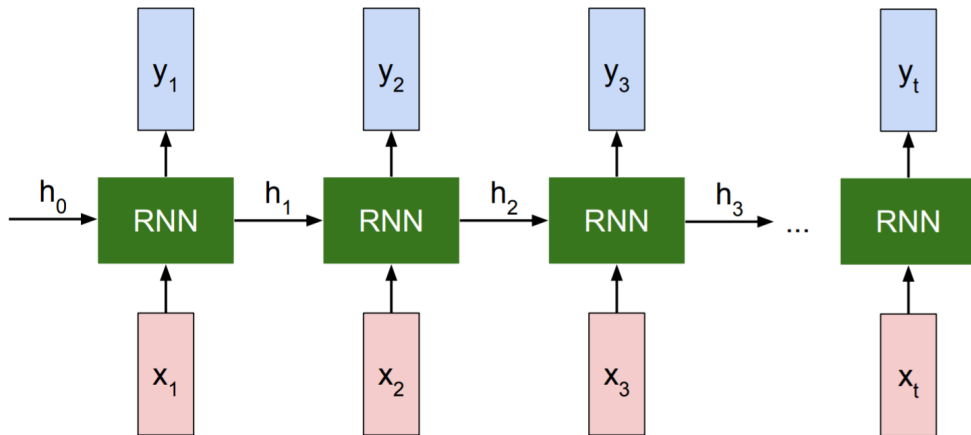
new state / some function with parameters $U \& W$

input vector at some time step

old state



Recurrent Neural Network

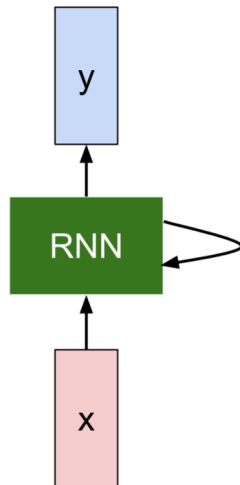


Recurrent Neural Network

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

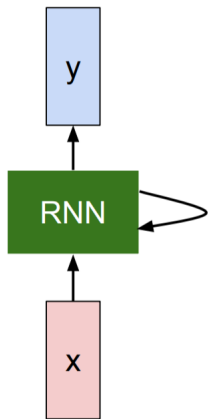
$$h_t = f_{UW}(x_t, h_{t-1})$$

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



(Simple) Recurrent Neural Network

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



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



$$h_t = \tanh(U x_t + W h_{t-1})$$

$$y_t = \text{SoftMax}(V h_t)$$

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

Computational Graphs: A Quick Review

- **Computational graph:** Directed graph where nodes correspond to:
 - Operations
 - Variables

Computational Graphs: A Quick Review

- **Computational graph**: Directed graph where nodes correspond to:
 - Operations
 - Variables
- Values that are fed into nodes and come out of nodes called **tensors** (multi-dimensional array)
 - Subsumes scalars, vectors and matrices as well

Computational Graphs: A Quick Review

- **Computational graph**: Directed graph where nodes correspond to:
 - Operations
 - Variables
- Values that are fed into nodes and come out of nodes called **tensors** (multi-dimensional array)
 - Subsumes scalars, vectors and matrices as well
- Can be instantiated to do two types of computation
 - Forward
 - Backward

Computational Graphs: Creating Expressions

- Nice way to think about mathematical expressions

Computational Graphs: Creating Expressions

- Nice way to think about mathematical expressions

- Consider the expression:

$$e = (a + b) * (b + 1)$$

Computational Graphs: Creating Expressions

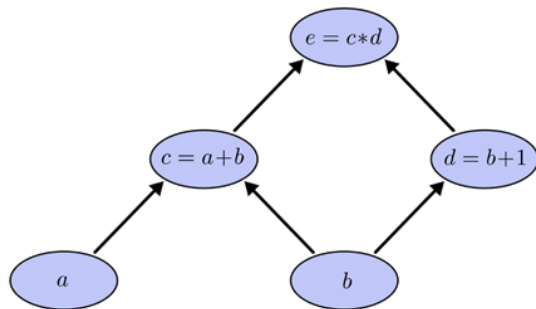
- Nice way to think about mathematical expressions
- Consider the expression:
$$e = (a + b) * (b + 1)$$
- 3 operations:
 - 2 additions
 - 1 multiplication

Computational Graphs: Creating Expressions

- Nice way to think about mathematical expressions
- Consider the expression:
$$e = (a + b) * (b + 1)$$
- 3 operations:
 - 2 additions
 - 1 multiplication
- Intermediate steps
 - $c = a + b$
 - $d = b + 1$
 - $e = c * d$

Computational Graphs: Creating Expressions

- Nice way to think about mathematical expressions
- Consider the expression:
 $e = (a + b) * (b + 1)$
- 3 operations:
 - 2 additions
 - 1 multiplication
- Intermediate steps
 - $c = a + b$
 - $d = b + 1$
 - $e = c * d$



Credit: Christopher Olah

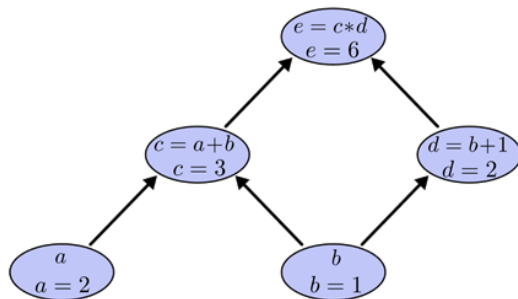
Computational Graphs: Evaluating Expressions

- To evaluate the expression
 - Set input variable to certain values
 - Compute nodes up through the graph

Credit: Christopher Olah

Computational Graphs: Evaluating Expressions

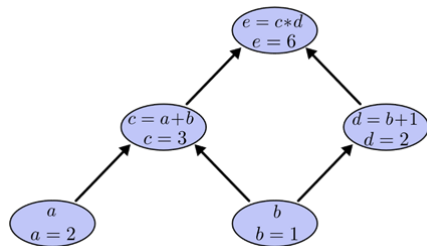
- To evaluate the expression
 - Set input variable to certain values
 - Compute nodes up through the graph



Credit: Christopher Olah

Computational Graphs: Computing Derivatives

- How?

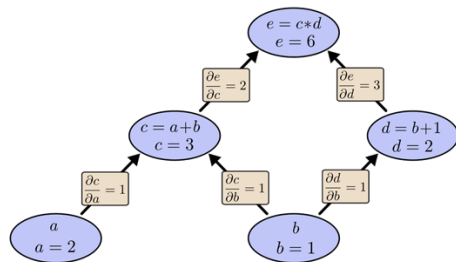


Computational Graphs: Computing Derivatives

- How?
- Key is to understand derivatives on edges (where changes - e.g. how a affects c - are tracked)

Computational Graphs: Computing Derivatives

- How?
- Key is to understand derivatives on edges (where changes - e.g. how a affects c - are tracked)



Credit: Christopher Olah

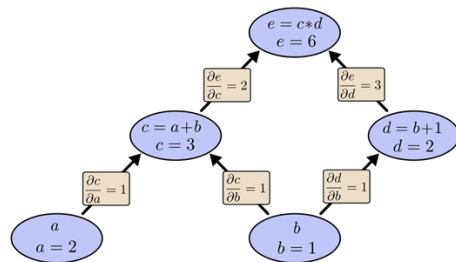
Vineeth N B (IIT-H)

§8.1 Introduction to RNNs

17 / 25

Computational Graphs: Computing Derivatives

- How?
- Key is to understand derivatives on edges (where changes - e.g. how a affects c - are tracked)
- We then apply **sum rule** and **product rule** appropriately to gradients



Credit: Christopher Olah

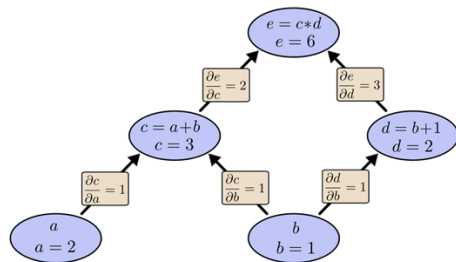
Vineeth N B (IIT-H)

§8.1 Introduction to RNNs

17 / 25

Computational Graphs: Computing Derivatives

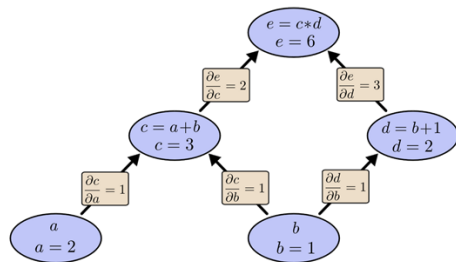
- How?
- Key is to understand derivatives on edges (where changes - e.g. how a affects c - are tracked)
- We then apply **sum rule** and **product rule** appropriately to gradients
- General rule is to sum over all possible paths from one node to other, multiplying derivatives on each edge of path together



Computational Graphs: Computing Derivatives

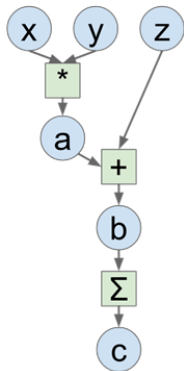
- How?
- Key is to understand derivatives on edges (where changes - e.g. how a affects c - are tracked)
- We then apply **sum rule** and **product rule** appropriately to gradients
- General rule is to sum over all possible paths from one node to other, multiplying derivatives on each edge of path together
- E.g. to get derivative of e w.r.t. b :

$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3$$



Computational Graphs: PyTorch Example

- In PyTorch, for e.g., changes are tracked on the go during forward pass allowing for dynamic graph creation
- Gradients are calculated only when `backward()` function is triggered



```
import torch
from torch.autograd import Variable

#----- Define Variables to build computational graph -----#
x = Variable(torch.tensor([1.0, 2.0]).cuda(), requires_grad = True)
y = Variable(torch.tensor([2.0, 3.0]).cuda(), requires_grad = True)
z = Variable(torch.tensor([4.0, 3.0]).cuda(), requires_grad = True)

#----- Forward Pass -----#
a = x * y
b = a + z
c = torch.sum(b)

#----- Compute Gradients -----#
c.backward()

print(x.grad.data) # out = [2., 3.]
print(y.grad.data) # out = [1., 2.]
print(z.grad.data) # out = [1., 1.]
```

Computational Graphs: MLP

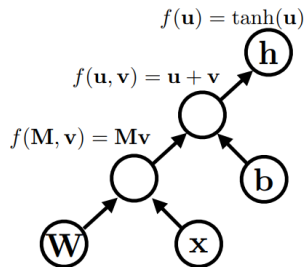
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$

Computational Graphs: MLP

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

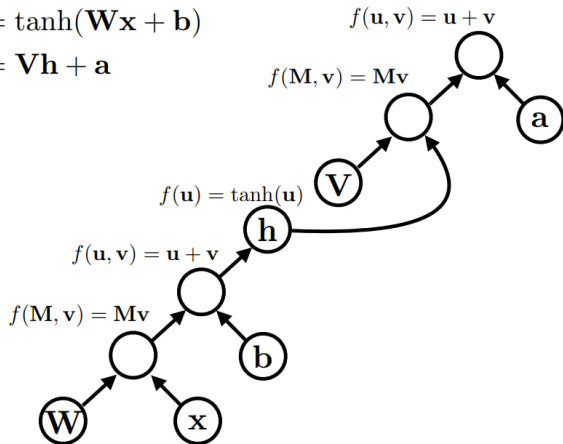
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



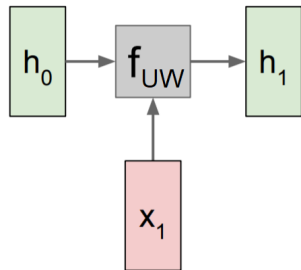
Computational Graphs: MLP

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

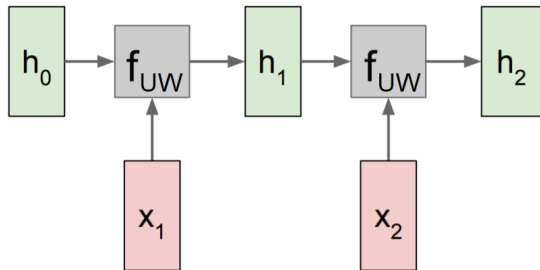
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



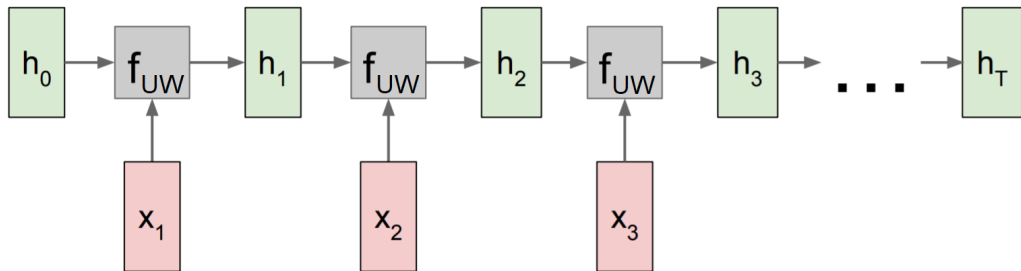
Back to RNNs: Computational Graph



Back to RNNs: Computational Graph

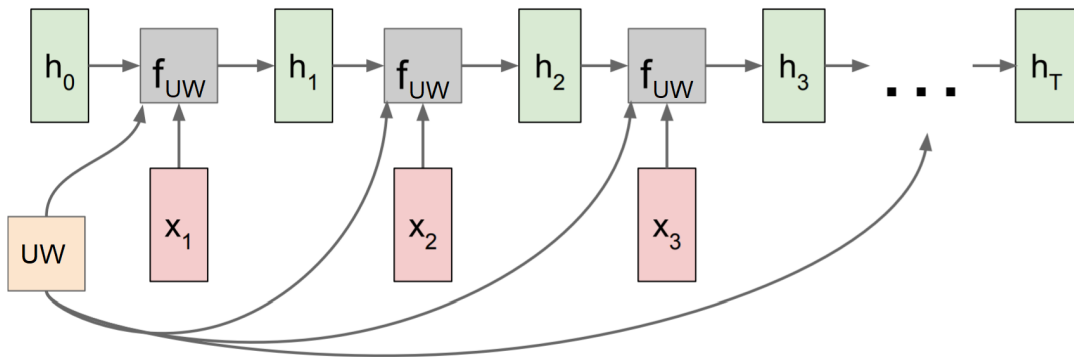


Back to RNNs: Computational Graph

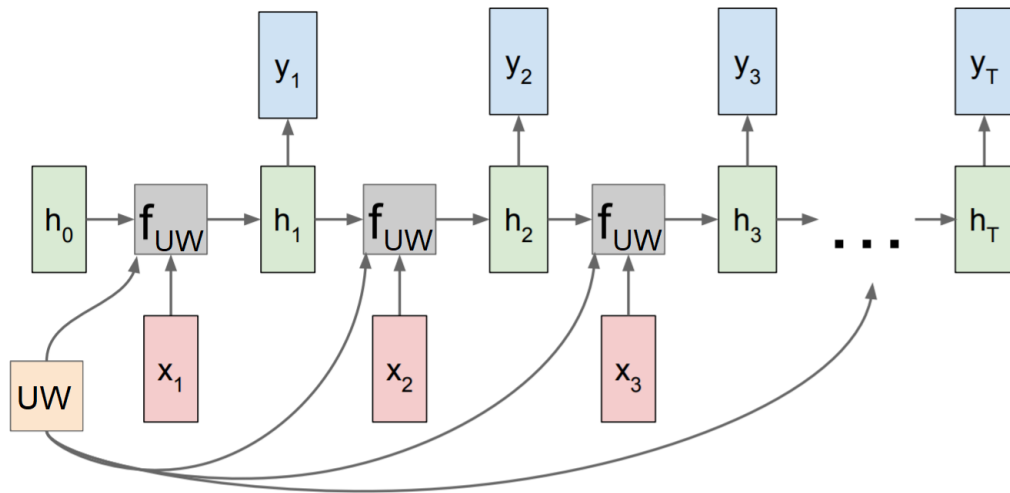


Back to RNNs: Computational Graph

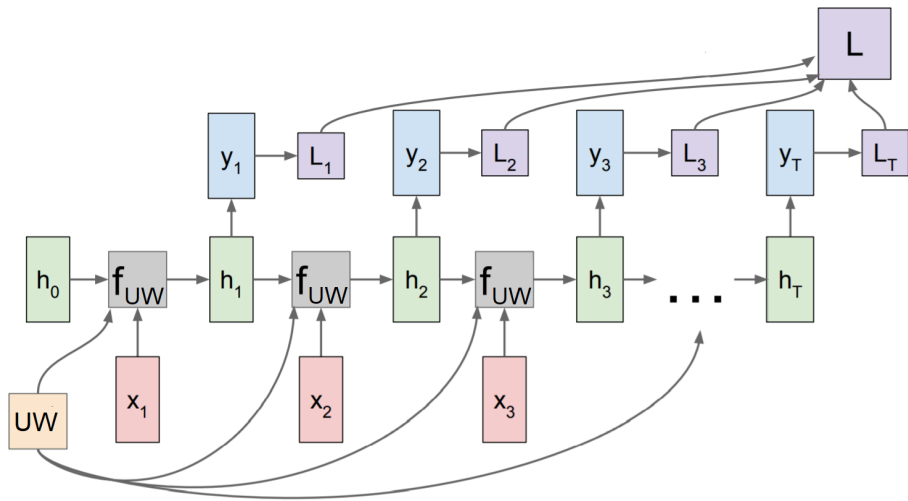
Re-use the same weight matrix at every time-step



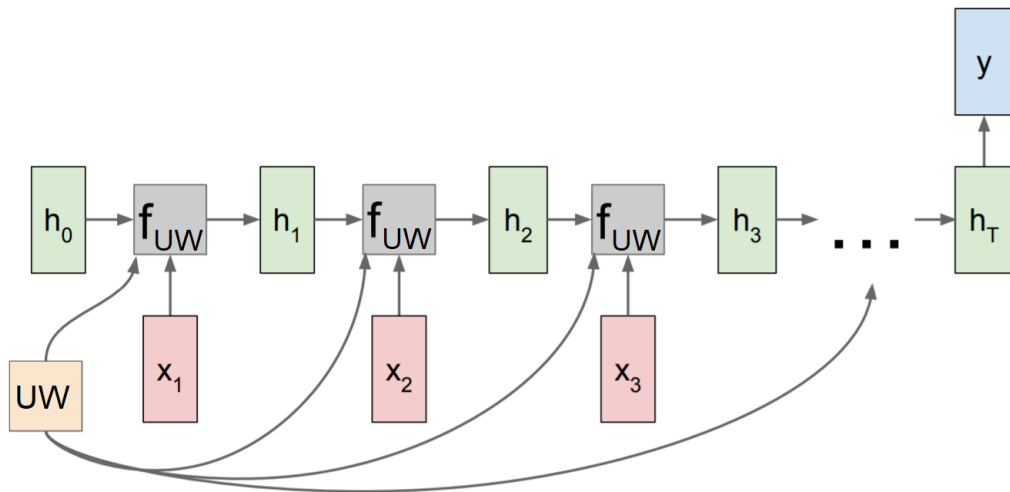
RNN Computational Graph: Many-to-Many



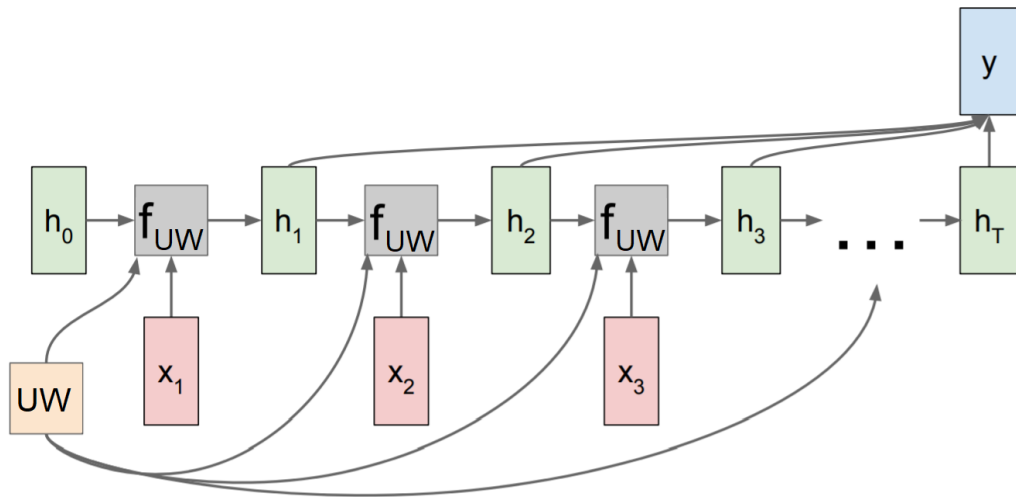
RNN Computational Graph: Many-to-Many



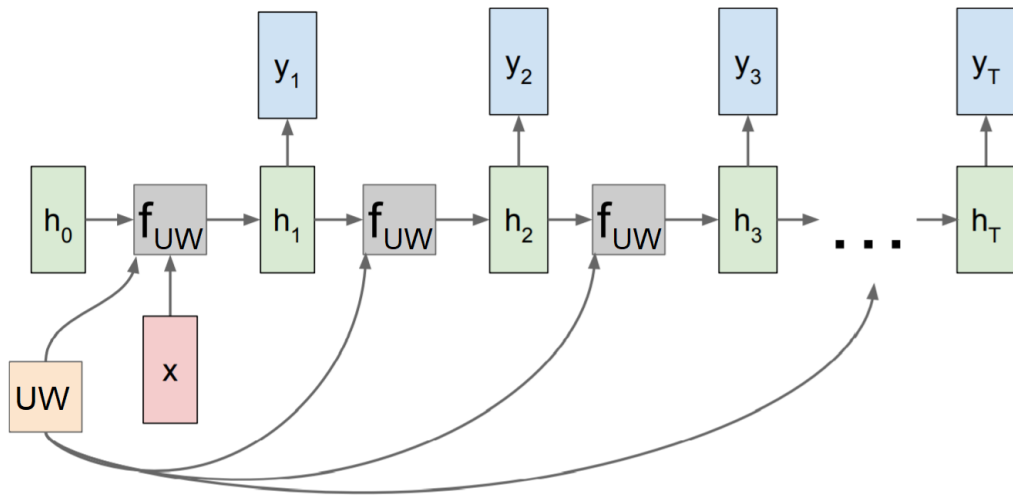
RNN Computational Graph: Many-to-One



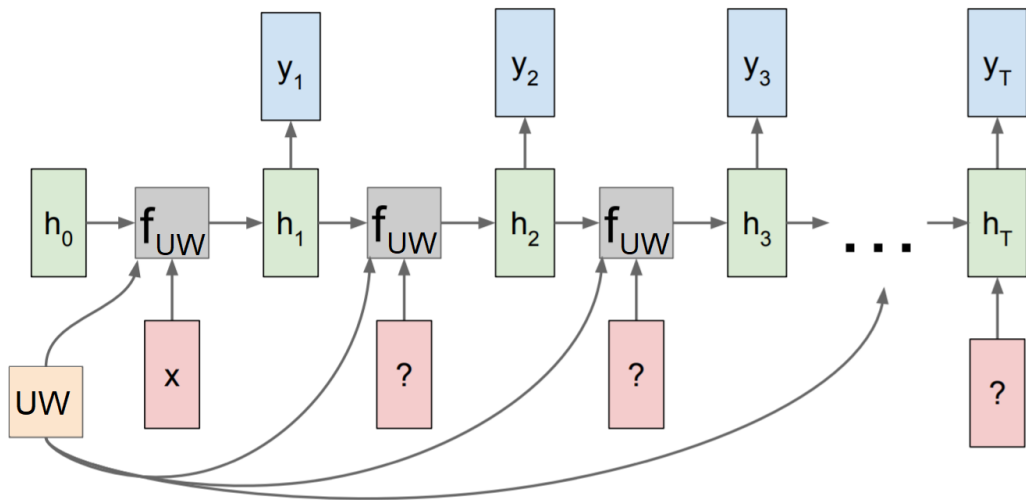
RNN Computational Graph: Many-to-One



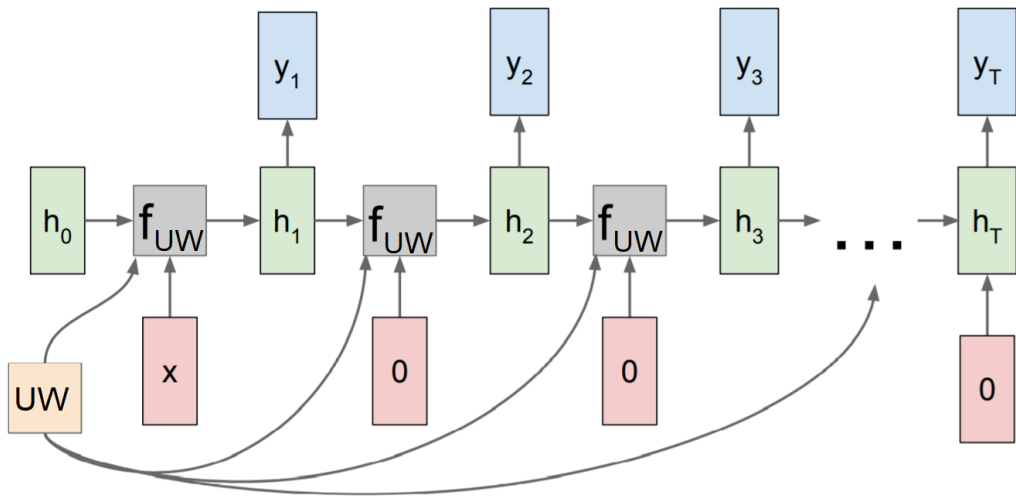
RNN Computational Graph: One-to-Many



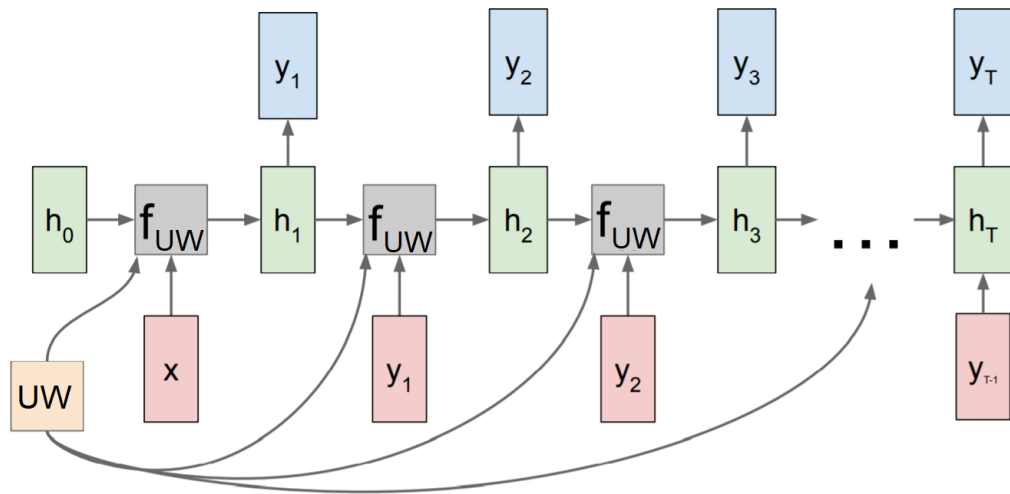
RNN Computational Graph: One-to-Many



RNN Computational Graph: One-to-Many

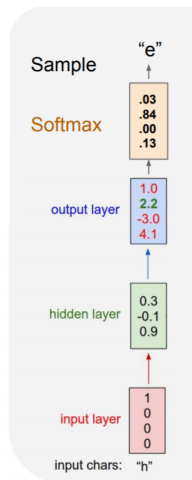


RNN Computational Graph: One-to-Many



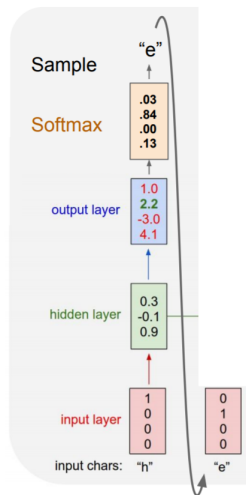
Example: Character-level Language Model

- Vocabulary: [h,e,l,o]
- At test time, sample characters one at a time, feed output back to model



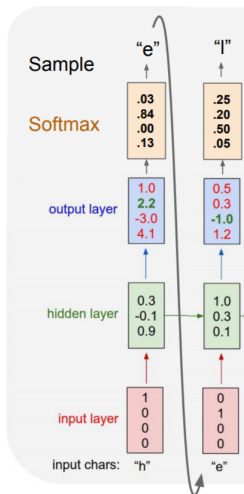
Example: Character-level Language Model

- Vocabulary: [h,e,l,o]
- At test time, sample characters one at a time, feed output back to model



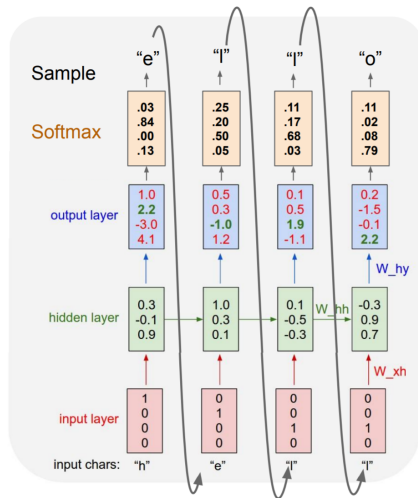
Example: Character-level Language Model

- Vocabulary: [h,e,l,o]
- At test time, sample characters one at a time, feed output back to model



Example: Character-level Language Model

- Vocabulary: [h,e,l,o]
- At test time, sample characters one at a time, feed output back to model



Homework

Readings

- [Chapter 10](#) of Deep Learning Book (Goodfellow et al)
- Andrej Karpathy's [blog post](#) on RNNs (Important)
- (Additional) [Lecture 10](#) - Stanford CS231n
- (Additional) [Lecture 13](#) - IIT Madras CS7015

Questions

- Can RNNs have more than one hidden layer?
- The state (h_t) of an RNN records information from all previous time steps. At each new timestep, the old information gets *morphed* slightly by the current input. What would happen if we *morphed* the state too much?