

Recurrent Neural Networks

Samuel Cheng

School of ECE
University of Oklahoma

April 18, 2024

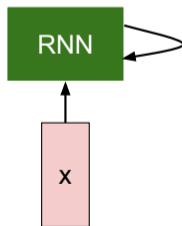
Table of Contents

- 1 Basic RNN
- 2 LSTM
- 3 Example: simple character-level language model
- 4 Example: image captioning
- 5 Overview of echo state networks
- 6 Conclusions

- We looked into couple use cases of CNNs previously
 - Recognition and localization
 - Object detection
 - Some use of CNNs for arts
- Up to now, the network models we have studied are all memoryless
 - We will discuss a non-memoryless model—recurrent neural networks today

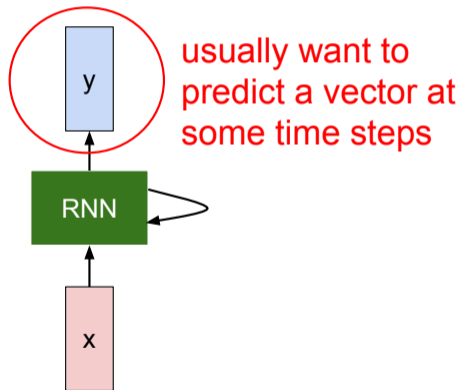
Recurrent neural networks (RNNs)

Recurrent Neural Network



Recurrent neural networks (RNNs)

Recurrent Neural Network



Recurrent neural networks (RNNs)

Recurrent Neural Network

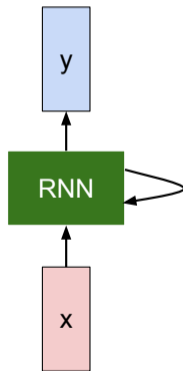
We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

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

new state / some function with parameters W

old state

input vector at some time step



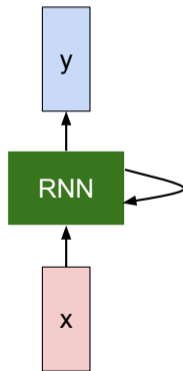
Recurrent neural networks (RNNs)

Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

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

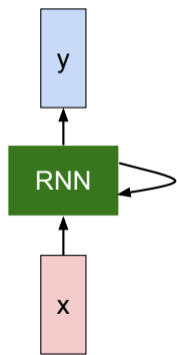
Notice: the same function and the same set of parameters are used at every time step.



Recurrent neural networks (RNNs)

(Vanilla) Recurrent Neural Network

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



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

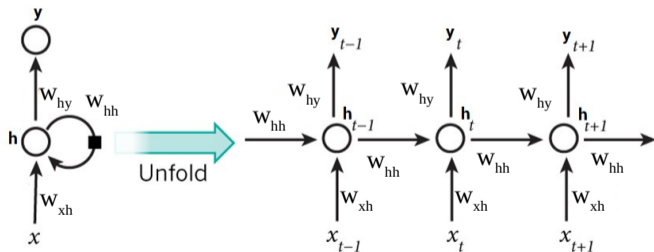


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

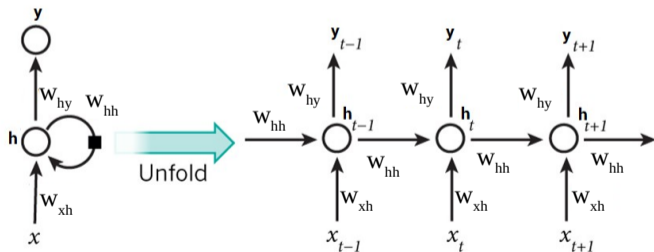
Back-Propagation Through Time (BPTT)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- The backward pass peels activities off the stack to compute the error derivatives at each time step
- After the backward pass we add together the derivatives at all the different times for each weight



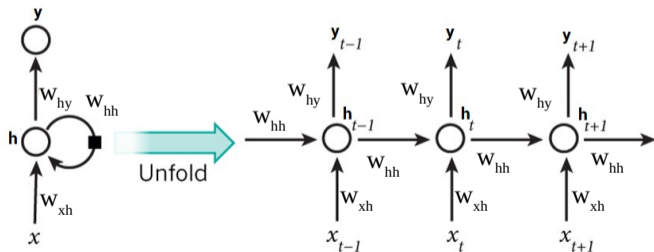
Back-Propagation Through Time (BPTT)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- The backward pass peels activities off the stack to compute the error derivatives at each time step
- After the backward pass we add together the derivatives at all the different times for each weight

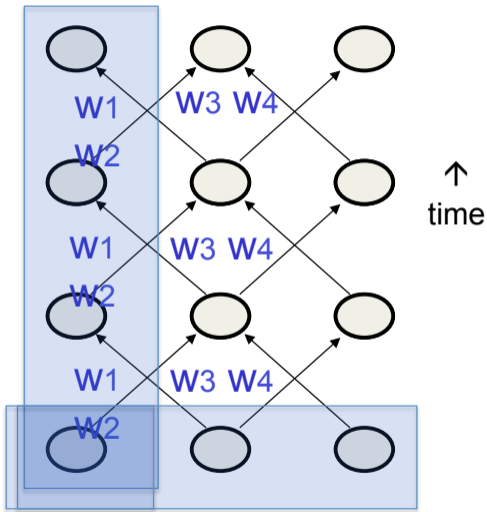


Back-Propagation Through Time (BPTT)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- The backward pass peels activities off the stack to compute the error derivatives at each time step
- After the backward pass we add together the derivatives at all the different times for each weight

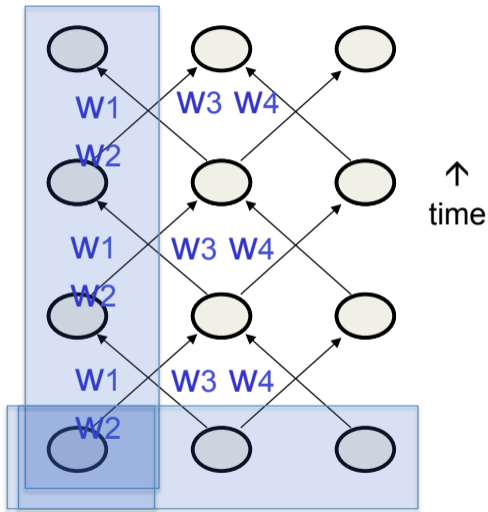


Providing inputs to recurrent networks



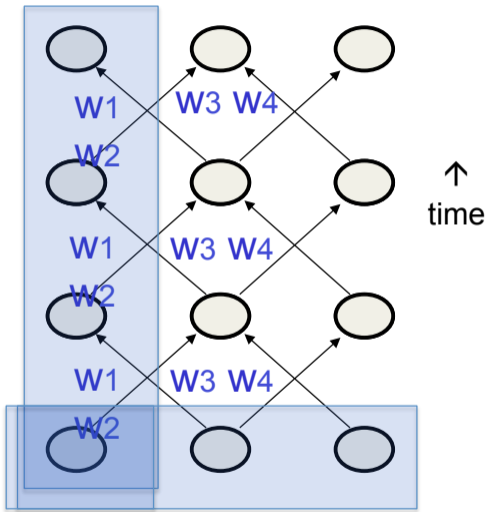
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



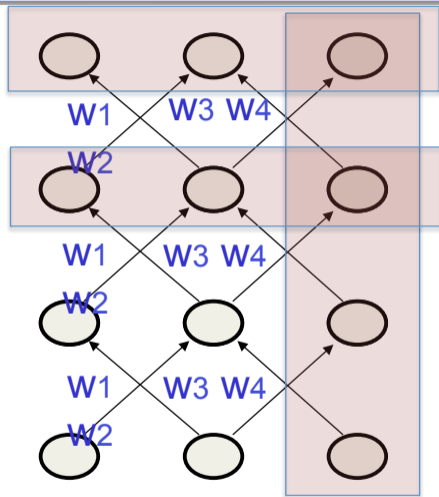
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



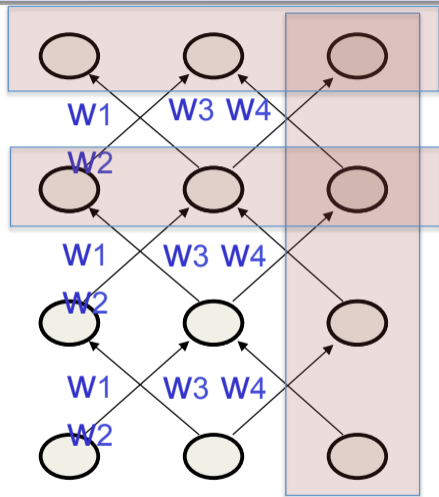
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Teaching recurrent networks to learn signals



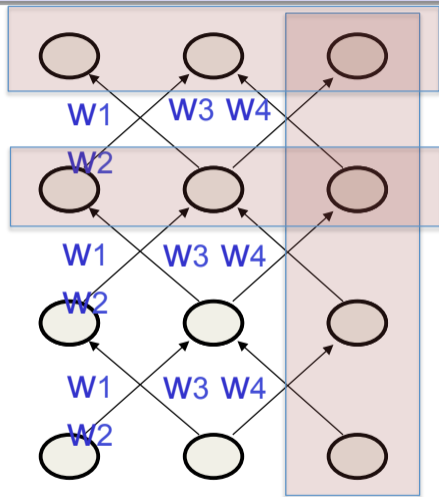
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



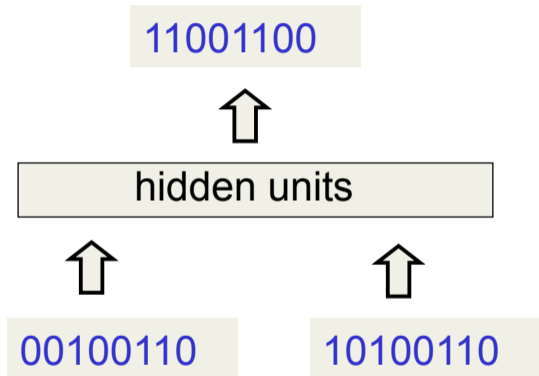
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



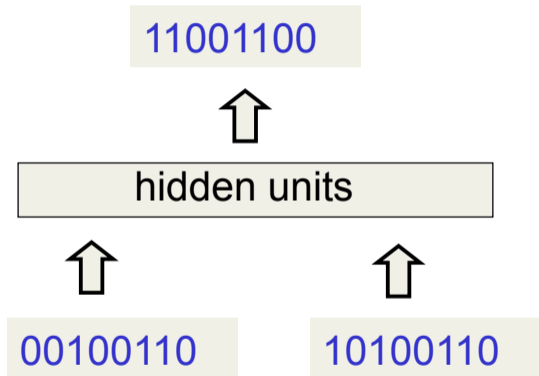
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Toy problem for RNN: binary addition



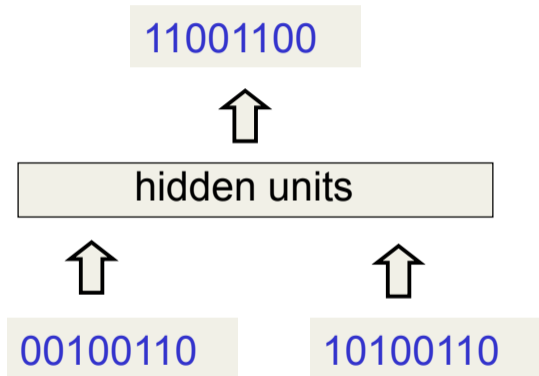
- We can train a feedforward net to do binary addition, but...
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

Toy problem for RNN: binary addition



- We can train a feedforward net to do binary addition, but...
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

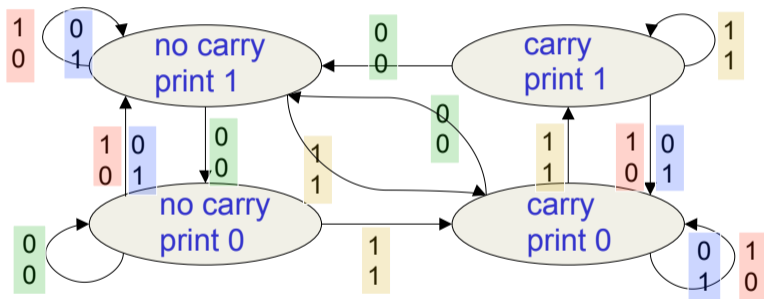
Toy problem for RNN: binary addition



- We can train a feedforward net to do binary addition, but...
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

We are trying to learn this!

The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - So RNNs have difficulty dealing with long-range dependencies

Four effective ways to learn an RNN

- **Long Short Term Memory:** Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:** Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:** Initialize the input→ hidden and hidden→hidden and output→ hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden→output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

From Hinton 2012 Cousera course

Four effective ways to learn an RNN

- **Long Short Term Memory:** Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:** Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:** Initialize the input→ hidden and hidden→hidden and output→ hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden→output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

From Hinton 2012 Coursera course

Four effective ways to learn an RNN

- **Long Short Term Memory:** Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:** Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:** Initialize the input \rightarrow hidden and hidden \rightarrow hidden and output \rightarrow hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden \rightarrow output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

From Hinton 2012 Cousera course

Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

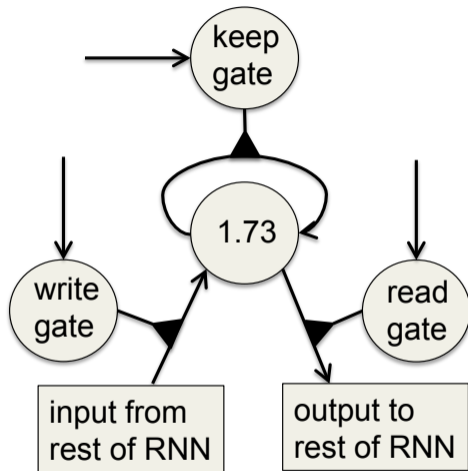
Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

Long Short Term Memory (LSTM)

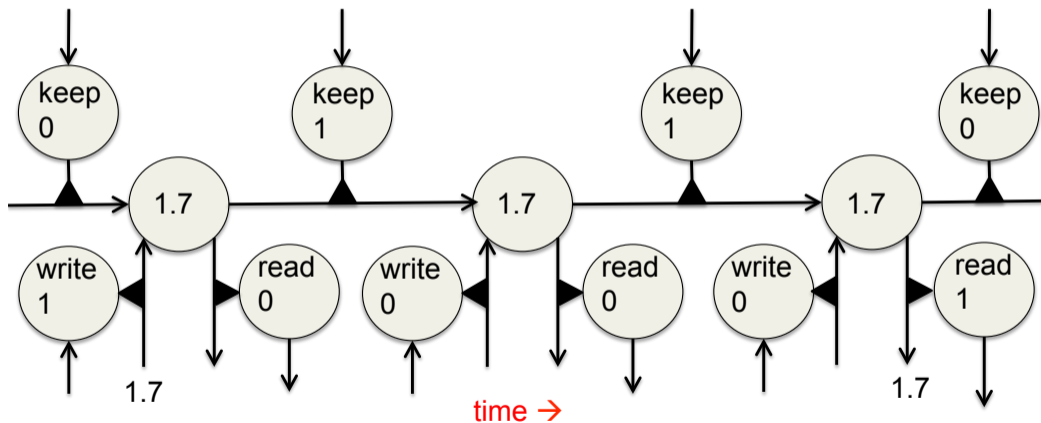
- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

Implementing a memory cell in a neural network



- To preserve information for a long time in the activities of an RNN, we use a circuit mimicking an analog memory cell
 - Information is kept in the cell when "keep" gate is on
 - Information is stored in the cell by activating its write gate
 - Information is retrieved by activating the read gate
 - We can backpropagate through this circuit because logitics are have nice derivatives

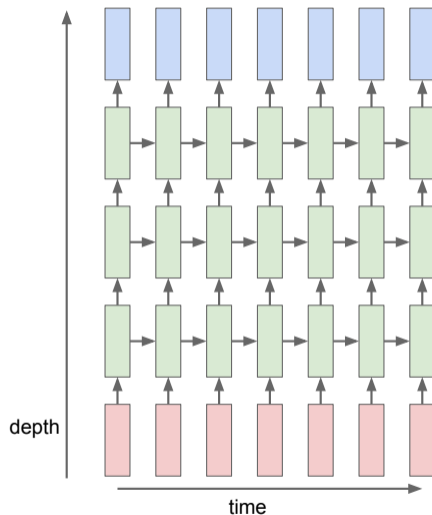
Backpropagation through a memory cell



RNN:

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

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



RNN:

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

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

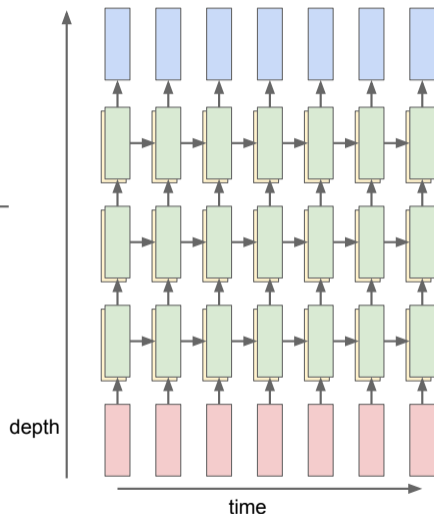
LSTM:

$$W^l [4n \times 2n]$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

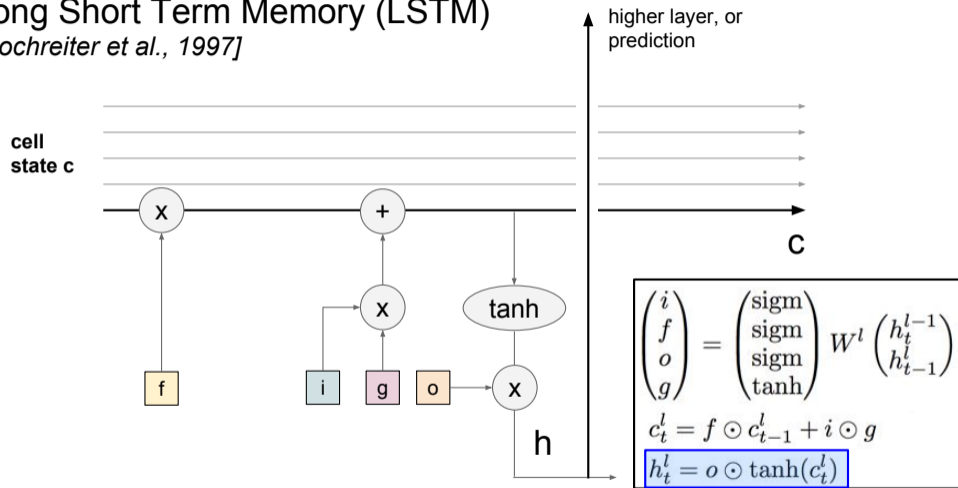
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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

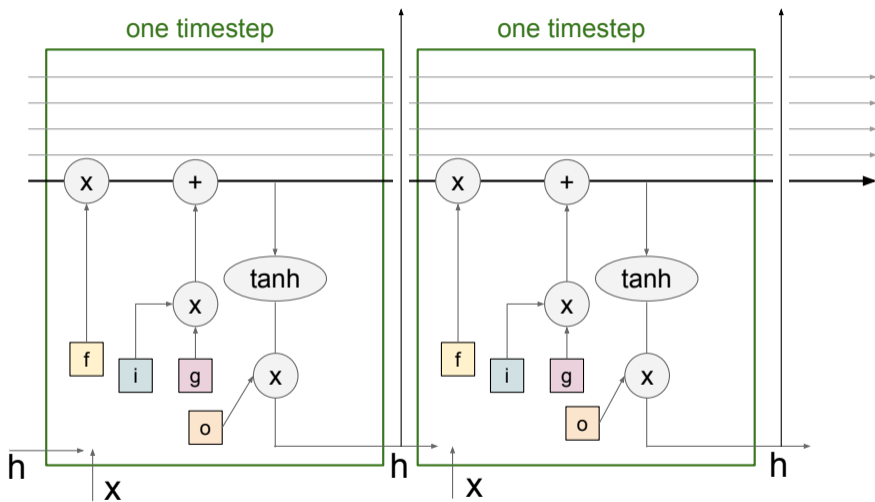


Long Short Term Memory (LSTM)

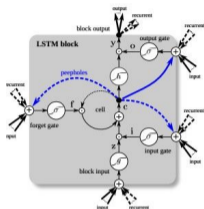
[Hochreiter et al., 1997]



LSTM



LSTM variants and friends



[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$\begin{aligned}
 r_t &= \text{sigm}(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
 z_t &= \text{sigm}(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
 \tilde{h}_t &= \text{tanh}(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
 h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
 \end{aligned}$$

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

MUT1:

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

MUT2:

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

MUT3:

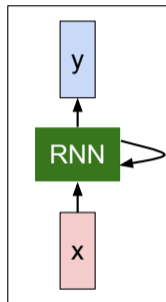
$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz} \text{tanh}(h_t) + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \text{tanh}(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &+ h_t \odot (1 - z)
 \end{aligned}$$

Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

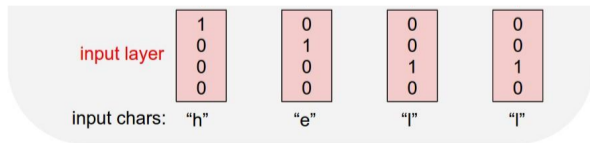


Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

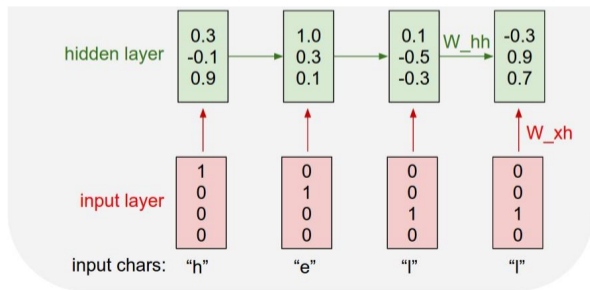
Example training
sequence:
“hello”



Simplest model: a first attempt

Character-level
language model
exampleVocabulary:
[h,e,l,o]Example training
sequence:
"hello"

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

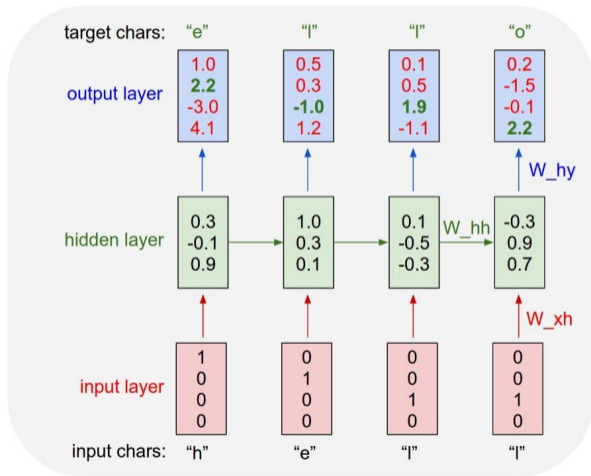


Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



Sampling

- Start the model with its default hidden state
- Give it a “burn-in” sequence of characters and let it update its hidden state after each character
- Then look at the probability distribution it predicts for the next character
- Pick a character randomly from that distribution and tell the net that this was the character that actually occurred
 - i.e. tell it that its guess was correct, whatever it guessed
- Continue to let it pick characters until bored

min-char-rnn.py gist: 112 lines of Python

```

1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 20 # number of steps to unroll the rnn for
18 learning_rate = 1e-1
19
20 # model parameters
21 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 woi = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 boh = np.random.randn(hidden_size, 1) # hidden bias
25 by = np.zeros(vocab_size, 1) # output bias
26
27 def lossfun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is list array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], []
34     h0 = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs.append([np.zeros(hidden_size, 1)] + encode(ix_to_char[input_t]))
39         hs.append([np.zeros(1)] + h0)
40         h0 = np.tanh(np.dot(wih, xs[t]) + np.dot(whh, hs[t-1]) + boh) # hidden state
41         ys.append(np.dot(woi, h0) + by) # unnormalized log probabilities for next chars
42         ps.append(np.exp(ys[t]) / np.sum(np.exp(ys))) # probabilities for next chars
43         loss += -np.log(ps[t][target_t], 0) # softmax (cross-entropy) loss
44     # backward pass: compute gradients going backwards
45     dwh, dwhh, dby = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(boh)
46     dho, dhoi, dby = np.zeros_like(h0), np.zeros_like(woi), np.zeros_like(boh)
47     dhwent = np.zeros_like(h0)
48     for t in reversed(range(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dho = np.dot(dy, h0[t:T])
52         dby += dy
53         dh = np.dot(woi.T, dy) + dhwent # backprop into h
54         dhwent = (1 - h0[t] * h0[t]) * dh # backprop through tanh nonlinearity
55         dhoi = dhwent
56         dwh += np.dot(dhwent, xs[t:T])
57         dwhh += np.dot(dhwent, h0[t-1:T])
58         dhwent = np.dot(whh.T, dhwent)
59     for dparam in [dwh, dwhh, dby, dho, dhoi]:
60         return loss, dwh, dwhh, dby, dho, dhoi
61
62 def sample(h, seed_ix, n):
63     """
64     sample a sequence of integers from the model
65     h is memory state, seed_ix is seed letter for first time step
66     """
67     x = np.zeros(vocab_size, 1)
68     [seed_ix] = 1
69     loss = []
70     for t in xrange(n):
71         h = np.tanh(np.dot(wih, x) + np.dot(whh, h) + boh)
72         y = np.dot(woi, h) + by
73         p = np.exp(y) / np.sum(np.exp(y))
74         ix = np.random.choice(range(vocab_size), p=p, rand())
75         x = np.zeros(vocab_size, 1)
76         x[ix] = 1
77         loss.append(ix)
78     return loss
79
80 n, p = 0, 0
81 w, w_h, w_o, h = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(woi)
82 w_hh, w_hb, w_by = np.zeros_like(wih), np.zeros_like(whh), np.zeros_like(woi)
83 smooth_loss = -np.log(1.0/vocab_size)/seq_length # loss at iteration 0
84 while True:
85     # prepare inputs (we're sampling from left to right in steps seq_length long)
86     if p == seq_length:
87         hprev = np.zeros(hidden_size, 1) # reset RNN memory
88         p = 0 # 0 from start of data
89         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
90         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
91
92         # a sample from the model now and then
93         if n % 100 == 0:
94             sample_ix = sample(hprev, inputs[0], 200)
95             txt = ''.join(ix_to_char[ix] for ix in sample_ix)
96             print '----%s%s-----' % (txt, 1)
97
98         # forward seq_length characters through the net and fetch gradients
99         loss, dwh, dwhh, dby, dho, dhoi, hprev = lossfun(inputs, targets, hprev)
100         smooth_loss = smooth_loss * 0.999 + loss * 0.001
101         if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
102
103         # perform parameter update with Adagrad
104         for param, dparam, mem in zip([wih, whh, woi, whh, woi, woi, woi, woi],
105                                     [dwh, dwhh, dby, dho, dhoi, dho, dhoi],
106                                     [None, None, None, None, None, None, None]):
107             mem += dparam * dparam
108             param -= learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
109
110         p += seq_length # move data pointer
111         n += 1 # iteration counter

```

(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

at first:

tynnd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
 plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng



train more

"Tmont thithey" fomesscerliund
 Keushey. Thom here
 sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y fil on aseterlome
 coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."



train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
 her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
 how, and Gogition is so overelical and offer.



train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
 princess, Princess Mary was easier, fed in had oftened him.
 Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.


VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:





















O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

open source textbook on algebraic geometry

 **The Stacks Project**

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex 	pdf 
	2. Conventions	online	tex 	pdf 
	3. Set Theory	online	tex 	pdf 
	4. Categories	online	tex 	pdf 
	5. Topology	online	tex 	pdf 
	6. Sheaves on Spaces	online	tex 	pdf 
	7. Sites and Sheaves	online	tex 	pdf 
	8. Stacks	online	tex 	pdf 
	9. Fields	online	tex 	pdf 
	10. Commutative Algebra	online	tex 	pdf 

Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source 

For $\bigoplus_{n=1, \dots, m}$ where $\mathcal{L}_{m,*} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparico in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ?? . Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S'')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of \mathcal{X}' , and T_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{\text{fppf}}^{\text{opp}}, (\text{Sch}/S)_{\text{fppf}}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ?? . It may replace S by $X_{\text{spaces, étale}}$ which gives an open subspace of X and T equal to $S_{Z_{\text{ar}}}$, see Descent, Lemma ?? . Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (3) by the construction in the description.

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X, \mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S . Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X . But given a scheme U and a surjective étale morphism $U \rightarrow X$. Let $U \cap U = \coprod_{i=1, \dots, n} U_i$ be the scheme X over S at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$. \square

The following lemma surjective restocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X, \dots, 0}$.

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{I}_1 \subset \mathcal{I}_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \mathbb{A}_2$ works.

Lemma 0.3. In Situation ?? . Hence we may assume $\mathfrak{q}' = 0$.

Proof. We will use the property we see that \mathfrak{p} is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F -algebra where δ_{n+1} is a scheme over S . \square

Proof. Omitted. □

Lemma 0.1. *Let \mathcal{C} be a set of the construction.*

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. *This is an integer Z is injective.*

Proof. See Spaces, Lemma ?? □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

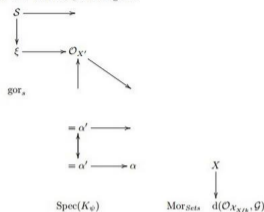
be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_* . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??

A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a "field"

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\mathbb{F}}^{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \rightarrow \mathcal{O}_{X_x}^{-1} \mathcal{O}_{X_x}(\mathcal{O}_{X_x}^{\mathbb{F}})$$

is an isomorphism of covering of \mathcal{O}_{X_x} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S .

If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum \mathcal{O}_{X_x} is a closed immersion, see Lemma ?? . This is a sequence of \mathcal{F} is a similar morphism.

The screenshot shows the GitHub interface for the 'torvalds / linux' repository. At the top, there is a search bar and navigation links for 'Explore', 'Gist', 'Blog', and 'Help'. The repository name 'torvalds / linux' is prominently displayed, along with statistics: 3,711 Watchers, 23,054 Stars, and 9,141 Forks. Below this, the repository is described as the 'Linux kernel source tree'. Key statistics include 520,037 commits, 1 branch, 420 releases, and 5,039 contributors. The current branch is 'master'. A recent merge of the 'drm-fixes' branch is highlighted. A table of recent merges is shown, including updates to 'Documentation', 'arch', 'block', 'crypto', 'drivers', 'firmware', 'fs', and 'include'. On the right side, there are links for 'Code', 'Pull requests' (74), 'Pulse', and 'Graphs'. At the bottom right, the 'HTTPS clone URL' is provided as 'https://github.com/torvalds/linux.git', along with buttons for 'Clone in Desktop' and 'Download ZIP'.

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated C code

```
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#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>
```

```
#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)

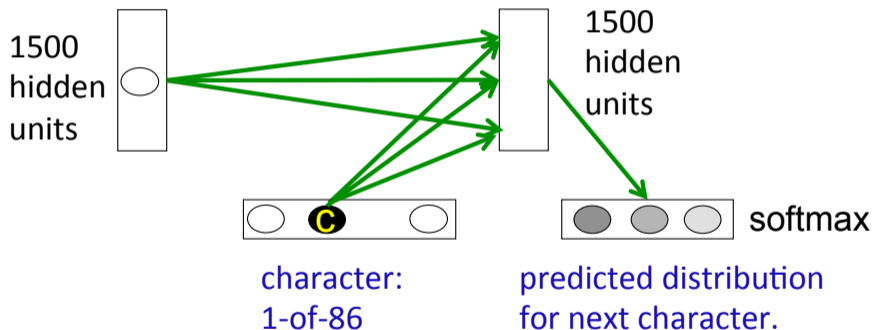
static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

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;
}

```

Hinton's work

An obvious recurrent neural net

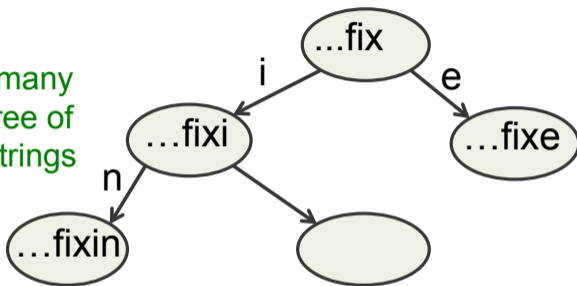


It's a lot easier to predict 86 characters than 100,000 words.

A slight tweak: Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



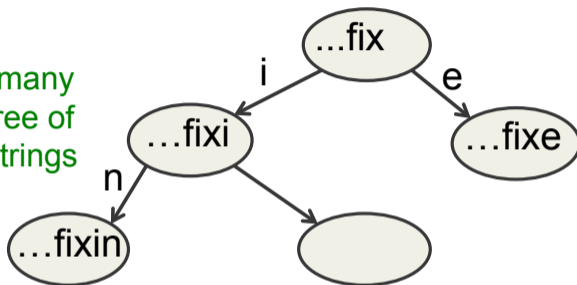
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier simple model tried to capture this but is kind of indirect

A slight tweak: Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



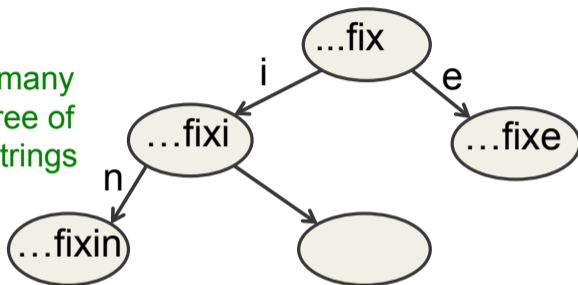
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier simple model tried to capture this but is kind of indirect

A slight tweak: Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier simple model tried to capture this but is kind of indirect

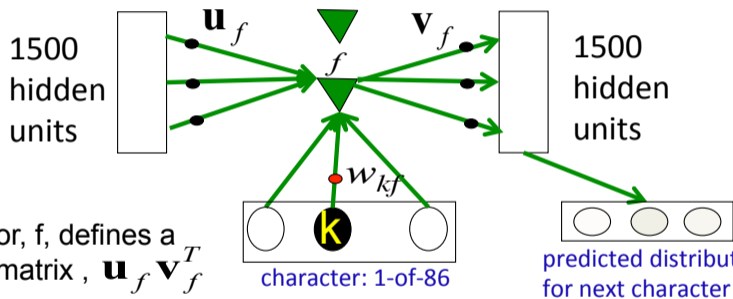
Multiplicative connections

- We may prepare a different transition matrix for each input
 - But this requires $86 \times 1500 \times 1500$ parameters (let say we have 1500 hidden variables)
 - And this could make the net overfit
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
 - We want a different transition matrix for each of the 86 characters, but we want these 86 character-specific weight matrices to share parameters (the characters 9 and 8 should have similar matrices)

Multiplicative connections

- We may prepare a different transition matrix for each input
 - But this requires $86 \times 1500 \times 1500$ parameters (let say we have 1500 hidden variables)
 - And this could make the net overfit
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
 - We want a different transition matrix for each of the 86 characters, but we want these 86 character-specific weight matrices to share parameters (the characters 9 and 8 should have similar matrices)

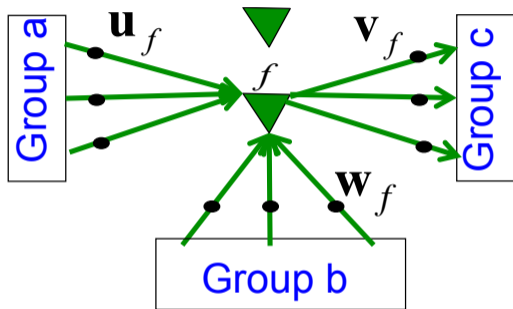
Using 3-way factors to allow a character to create a whole transition matrix



Each factor, f , defines a rank one matrix, $\mathbf{u}_f \mathbf{v}_f^T$

Each character, k , determines a gain w_{kf} for each of these matrices.

Using factors to implement multiplicative interactions

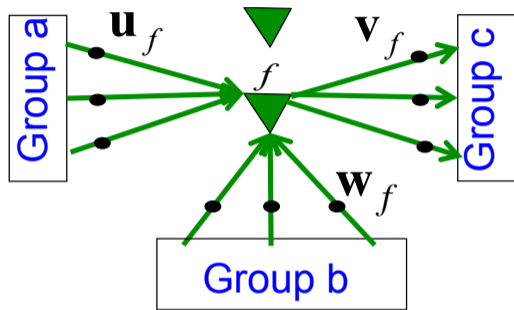


Vector input to group c:

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group b}} \underbrace{(a^T u_f)}_{\text{Scalar input from group a}} v_f$$

- We can get groups a and b to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement multiplicative interactions

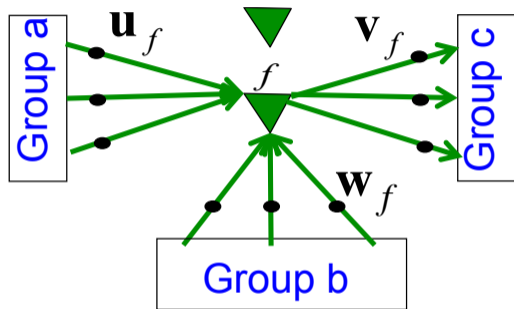


Vector input to group c:

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group b}} \underbrace{(a^T u_f)}_{\text{Scalar input from group a}} v_f$$

- We can get groups a and b to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement multiplicative interactions

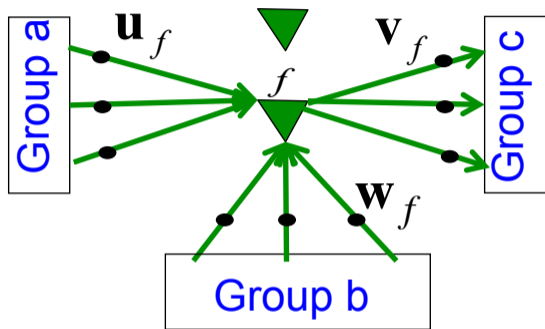


Vector input to group c:

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group b}} \underbrace{(a^T u_f)}_{\text{Scalar input from group a}} v_f$$

- We can get groups a and b to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement a set of basis matrices

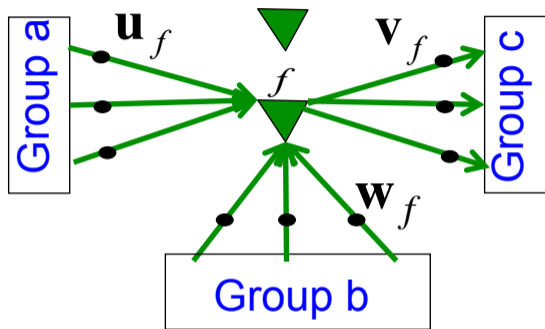


$$\begin{aligned}
 & c_f \\
 &= (\mathbf{b}^T \mathbf{w}_f) (\mathbf{a}^T \mathbf{u}_f) \mathbf{v}_f \\
 &= (\mathbf{b}^T \mathbf{w}_f) \mathbf{v}_f (\mathbf{u}_f^T \mathbf{a}) \\
 &= \underbrace{(\mathbf{b}^T \mathbf{w}_f)}_{\text{scalar coefficient}} \underbrace{(\mathbf{v}_f \mathbf{u}_f^T)}_{\text{outer product transition matrix with rank 1}} \mathbf{a}
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$\mathbf{c} = \left(\sum_f (\mathbf{b}^T \mathbf{w}_f) (\mathbf{v}_f \mathbf{u}_f^T) \right) \mathbf{a}$$

Using factors to implement a set of basis matrices

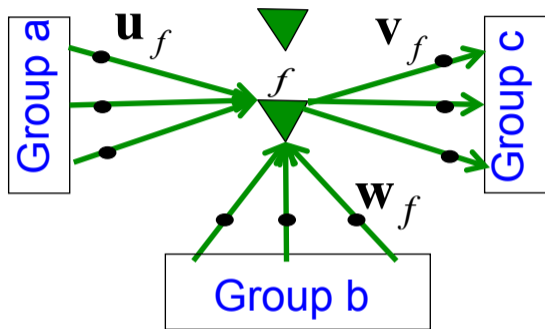


$$\begin{aligned}
 & c_f \\
 &= (\mathbf{b}^T \mathbf{w}_f) (\mathbf{a}^T \mathbf{u}_f) \mathbf{v}_f \\
 &= (\mathbf{b}^T \mathbf{w}_f) \mathbf{v}_f (\mathbf{u}_f^T \mathbf{a}) \\
 &= \underbrace{(\mathbf{b}^T \mathbf{w}_f)}_{\text{scalar coefficient}} \underbrace{(\mathbf{v}_f \mathbf{u}_f^T)}_{\text{outer product transition matrix with rank 1}} \mathbf{a}
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$\mathbf{c} = \left(\sum_f (\mathbf{b}^T \mathbf{w}_f) (\mathbf{v}_f \mathbf{u}_f^T) \right) \mathbf{a}$$

Using factors to implement a set of basis matrices

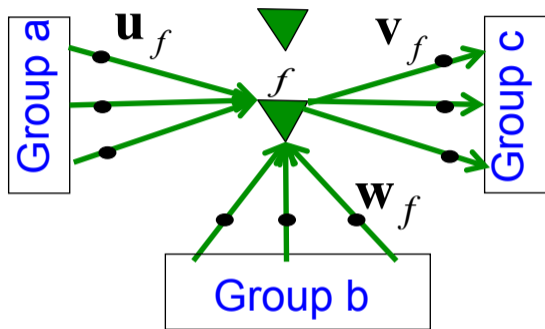


$$\begin{aligned}
 & c_f \\
 &= (\mathbf{b}^T \mathbf{w}_f)(\mathbf{a}^T \mathbf{u}_f) \mathbf{v}_f \\
 &= (\mathbf{b}^T \mathbf{w}_f) \mathbf{v}_f (\mathbf{u}_f^T \mathbf{a}) \\
 &= \underbrace{(\mathbf{b}^T \mathbf{w}_f)}_{\text{scalar coefficient}} \underbrace{(\mathbf{v}_f \mathbf{u}_f^T)}_{\text{outer product transition matrix with rank 1}} \mathbf{a}
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$\mathbf{c} = \left(\sum_f (\mathbf{b}^T \mathbf{w}_f) (\mathbf{v}_f \mathbf{u}_f^T) \right) \mathbf{a}$$

Using factors to implement a set of basis matrices



$$\begin{aligned}
 & c_f \\
 &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$c = \left(\sum_f (b^T w_f)(v_f u_f^T) \right) a$$

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

Conjugate gradient

- There is an alternative to going to the minimum in one step by multiplying by the inverse of the curvature matrix
- Use a sequence of steps each of which finds the minimum along one direction
- Make sure that each new direction is “conjugate” to the previous directions so you do not mess up the minimization you already did.
 - “conjugate” means that as you go in the new direction, you do not change the gradients in the previous directions

Conjugate gradient

- There is an alternative to going to the minimum in one step by multiplying by the inverse of the curvature matrix
- Use a sequence of steps each of which finds the minimum along one direction
- Make sure that each new direction is “conjugate” to the previous directions so you do not mess up the minimization you already did.
 - “conjugate” means that as you go in the new direction, you do not change the gradients in the previous directions

Conjugate gradient

- There is an alternative to going to the minimum in one step by multiplying by the inverse of the curvature matrix
- Use a sequence of steps each of which finds the minimum along one direction
- Make sure that each new direction is “conjugate” to the previous directions so you do not mess up the minimization you already did.
 - “conjugate” means that as you go in the new direction, you do not change the gradients in the previous directions

Training the model

- Ilya Sutskever used 5 million strings of 100 characters taken from wikipedia. For each string he starts predicting at the 11th character
- Using the HF optimizer, it took a month on a GPU board to get a really good model (back in 2011) text

Result

He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters' sisters in lower coil trains were always operated on the line of the **ephemeral** street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade."

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

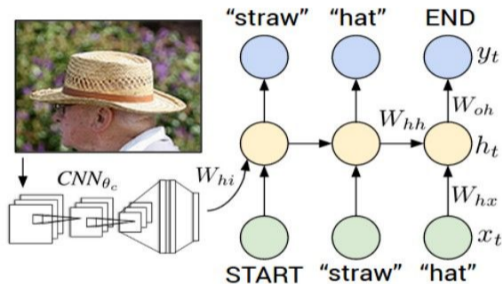
Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

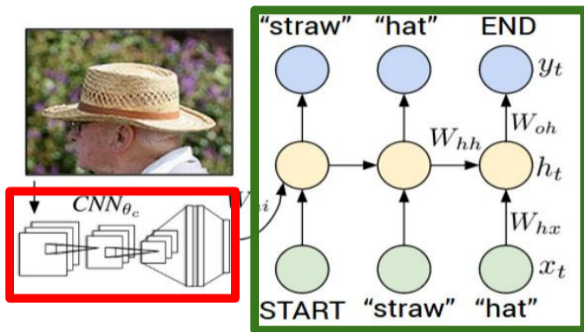
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network

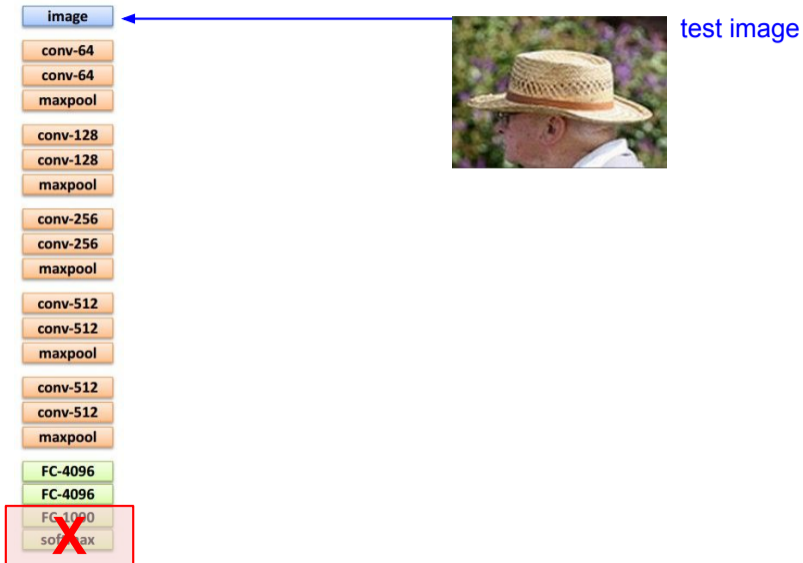


Convolutional Neural Network



test image





test image



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

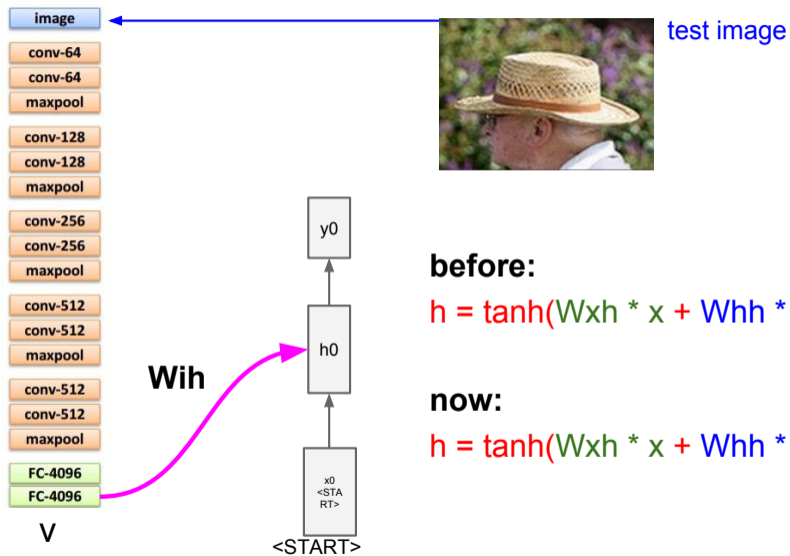
maxpool

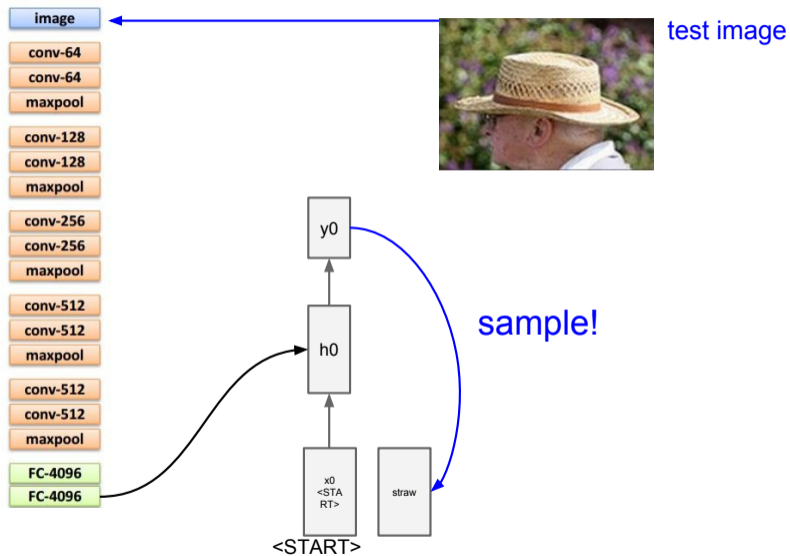
FC-4096

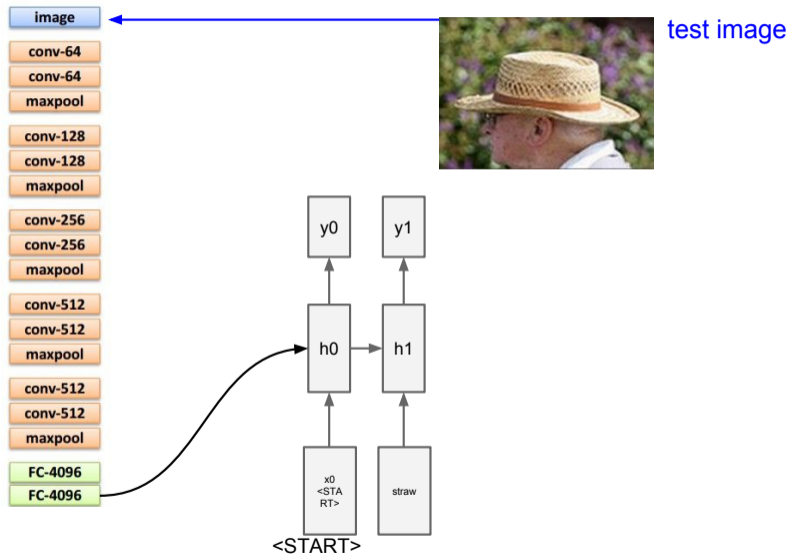
FC-4096

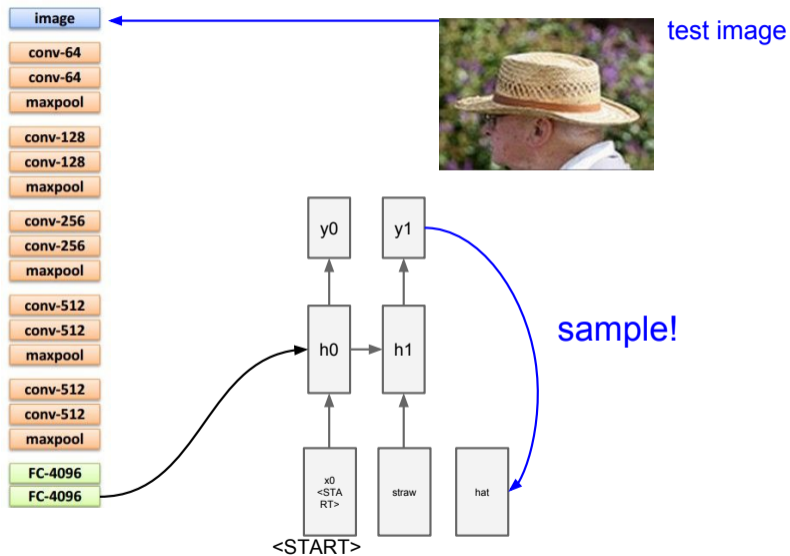
x0
<STA
RT>

<START>

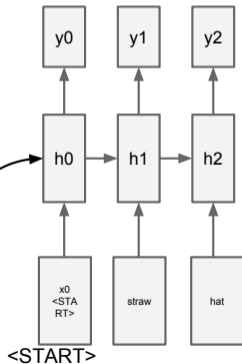








test image



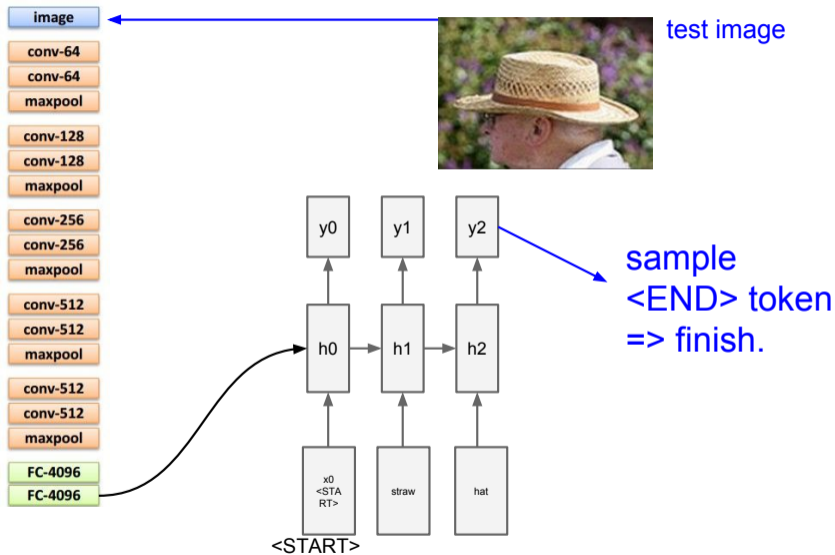


Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.



Microsoft COCO

[*Tsung-Yi Lin et al. 2014*]

mscoco.org

currently:

~120K images

~5 sentences each



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



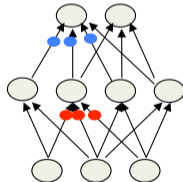
"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

The key idea of echo state networks (perceptrons again?)

- A very simple way to learn a feedforward network is to make the early layers random and fixed.
- Then we just learn the last layer which is a linear model that uses the transformed inputs to predict the target outputs.
 - A big random expansion of the input vector can help.



- The equivalent idea for RNNs is to fix the **input→hidden** connections and the **hidden→hidden** connections at random values and only learn the **hidden→output** connections.
 - The learning is then very simple (assuming linear output units).
 - Its important to set the random connections very carefully so the RNN does not explode or die.

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the input→hidden weights and sparsenesses
 - This is often necessary

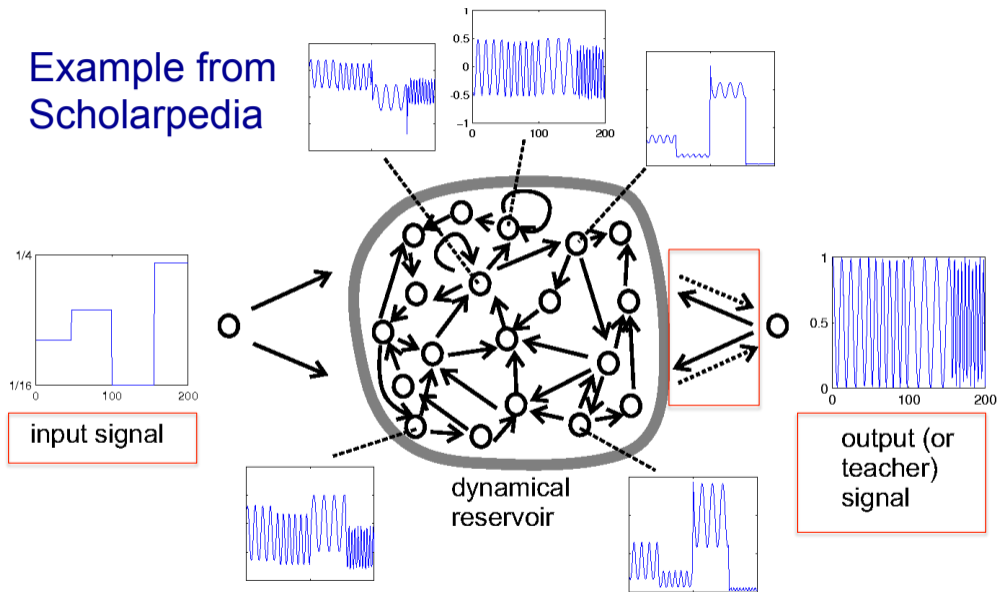
A simple example of an echo state network

INPUT SEQUENCE A real-valued time-varying value that specifies the frequency of a sine wave

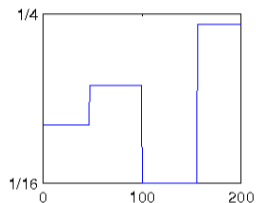
TARGET OUTPUT SEQUENCE A sine wave with the currently specified frequency

LEARNING METHOD Fit a linear model that takes the states of the hidden units as input and produces a single scalar output

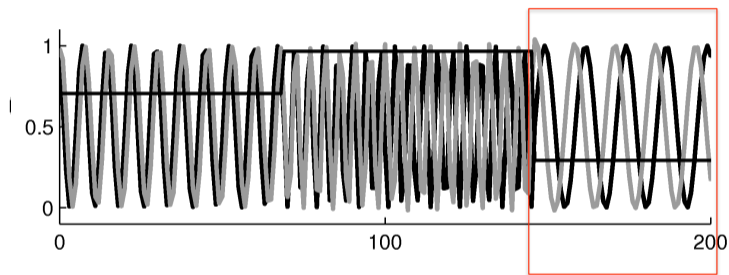
Example from Scholarpedia



The target and predicted outputs after learning



input signal



Beyond echo state networks

- Good aspects of ESNs: Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs: They need many more hidden units for a given task than an RNN that learns the hidden \rightarrow hidden weights
- Ilya Sutskever (2012) has illustrated that if the weights are initialized using the ESN methods, RNNs could be trained very effectively
 - He uses rmsprop with momentum

Beyond echo state networks

- Good aspects of ESNs: Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs: They need many more hidden units for a given task than an RNN that learns the hidden \rightarrow hidden weights
- Ilya Sutskever (2012) has illustrated that if the weights are initialized using the ESN methods, RNNs could be trained very effectively
 - He uses rmsprop with momentum

Beyond echo state networks

- Good aspects of ESNs: Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs: They need many more hidden units for a given task than an RNN that learns the hidden \rightarrow hidden weights
- Ilya Sutskever (2012) has illustrated that if the weights are initialized using the ESN methods, RNNs could be trained very effectively
 - He uses rmsprop with momentum

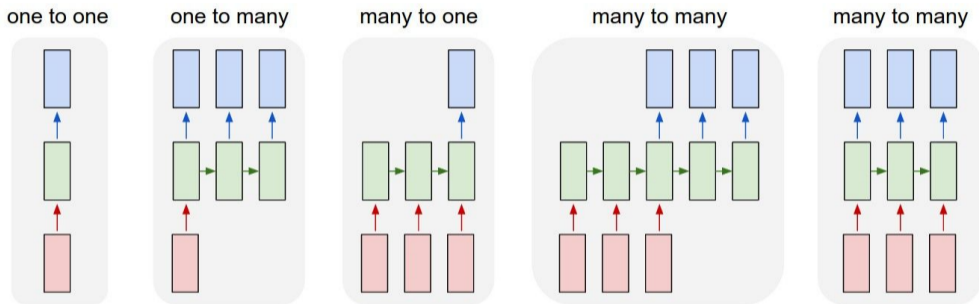
Beyond echo state networks

- Good aspects of ESNs: Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs: They need many more hidden units for a given task than an RNN that learns the hidden \rightarrow hidden weights
- Ilya Sutskever (2012) has illustrated that if the weights are initialized using the ESN methods, RNNs could be trained very effectively
 - He uses rmsprop with momentum

Conclusions

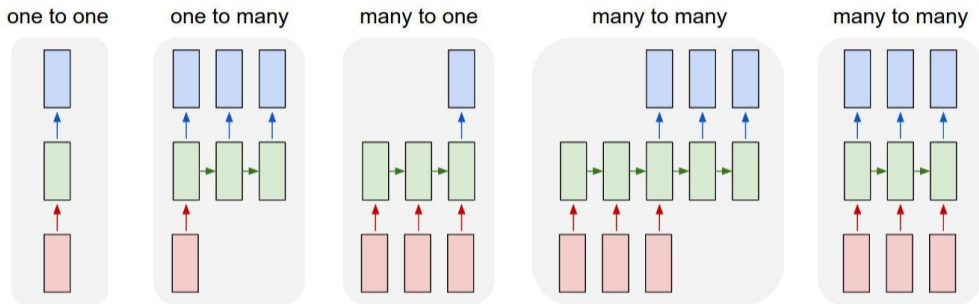
- RNNs allow a lot of flexibility in architecture design and have many applications

Recurrent Networks offer a lot of flexibility:



Vanilla Neural Networks

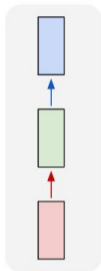
Recurrent Networks offer a lot of flexibility:



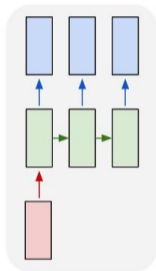
↖ e.g. **Image Captioning**
image → sequence of words

Recurrent Networks offer a lot of flexibility:

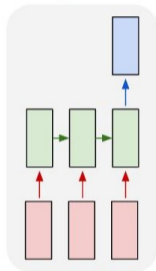
one to one



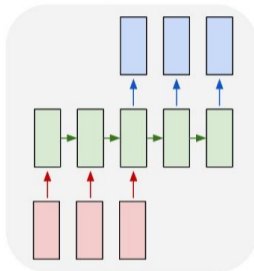
one to many



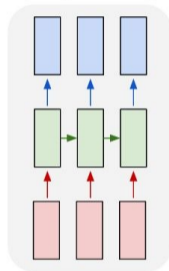
many to one



many to many



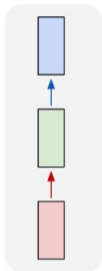
many to many



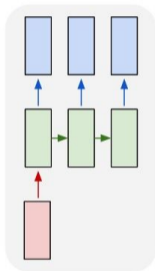
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

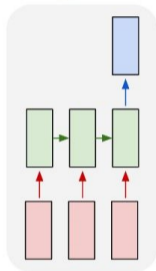
one to one



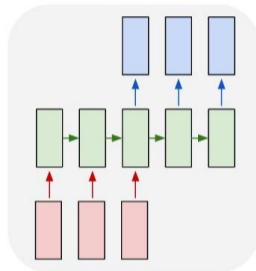
one to many



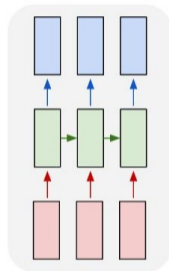
many to one



many to many

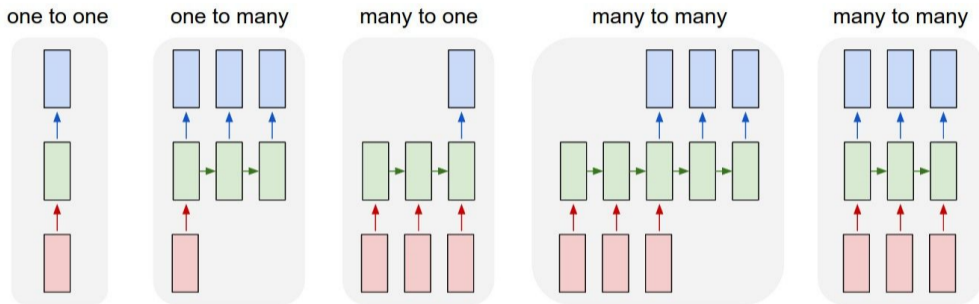


many to many



e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:



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

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- 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 optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- 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 optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- 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 optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- 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 optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- 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 optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs