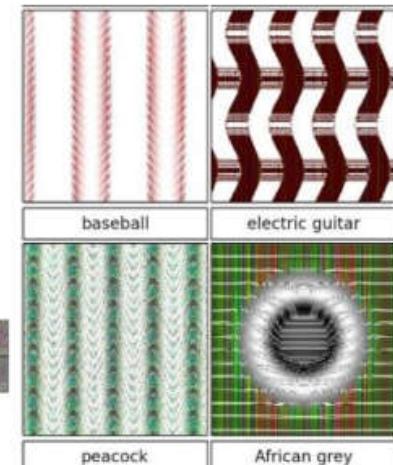
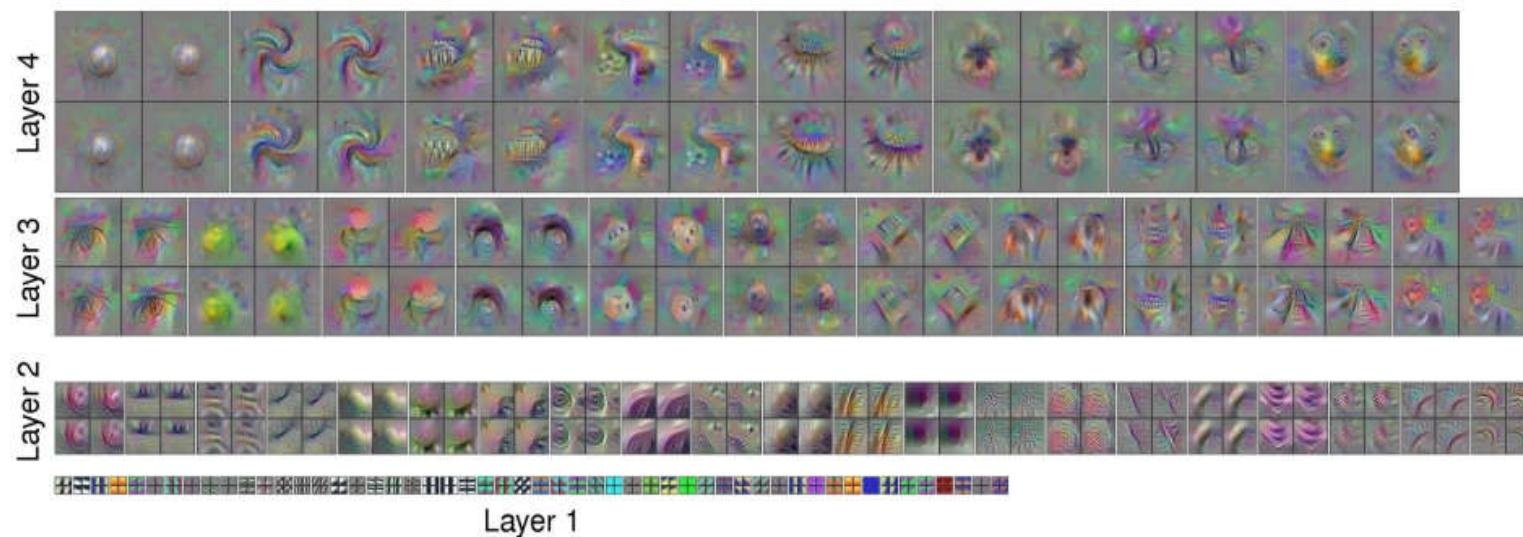
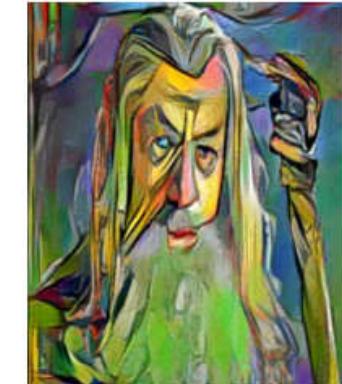
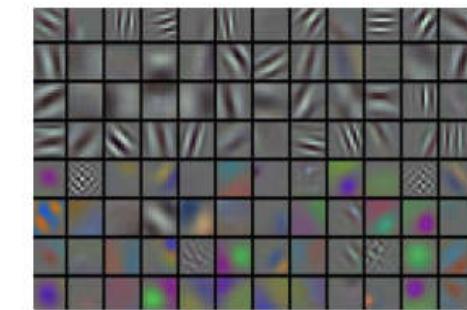


Lecture 10 – Recurrent Neural Networks

박사과정 김성빈 chengbinjin@inha.edu,
지도교수 김학일 교수 hikim@inha.ac.kr
인하대학교 컴퓨터비전 연구실

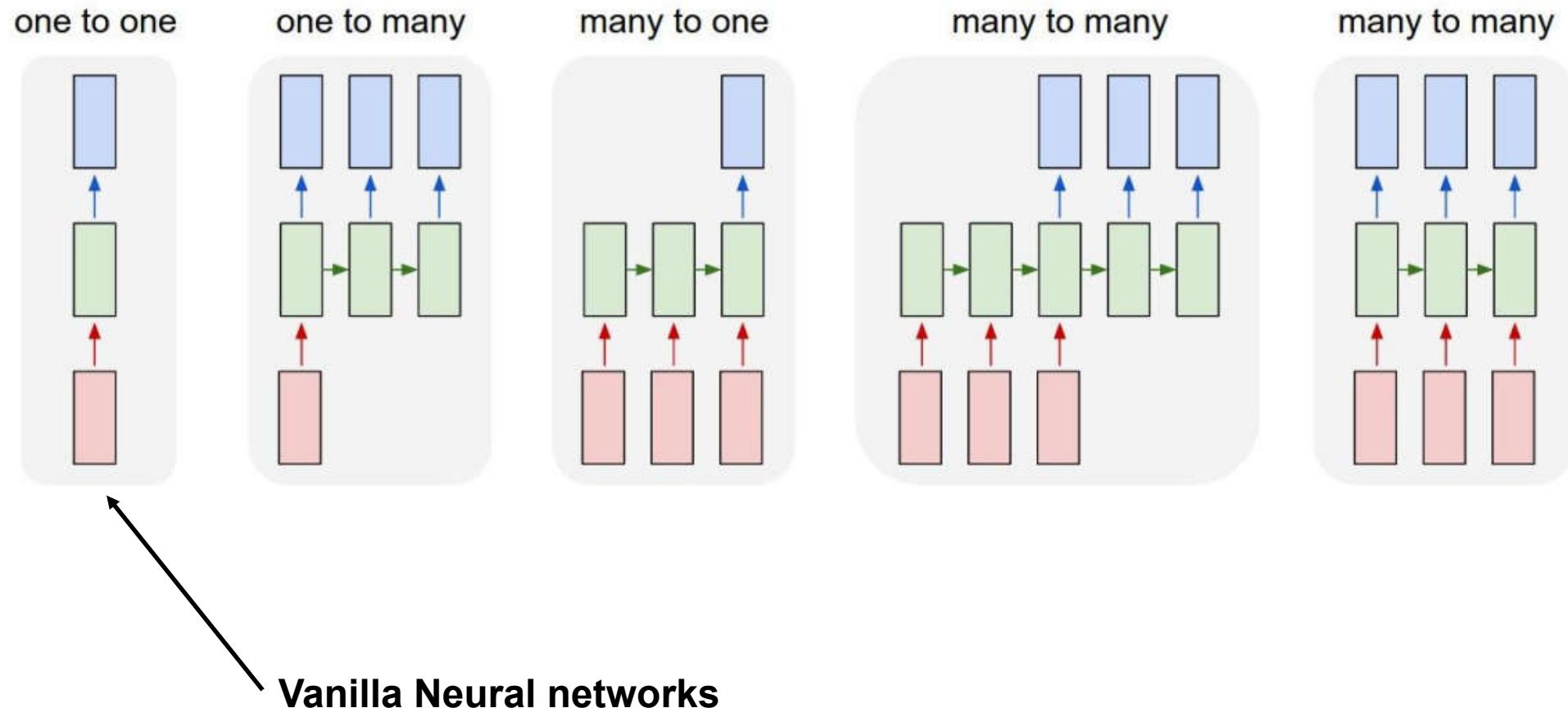


Recall from the Last Class

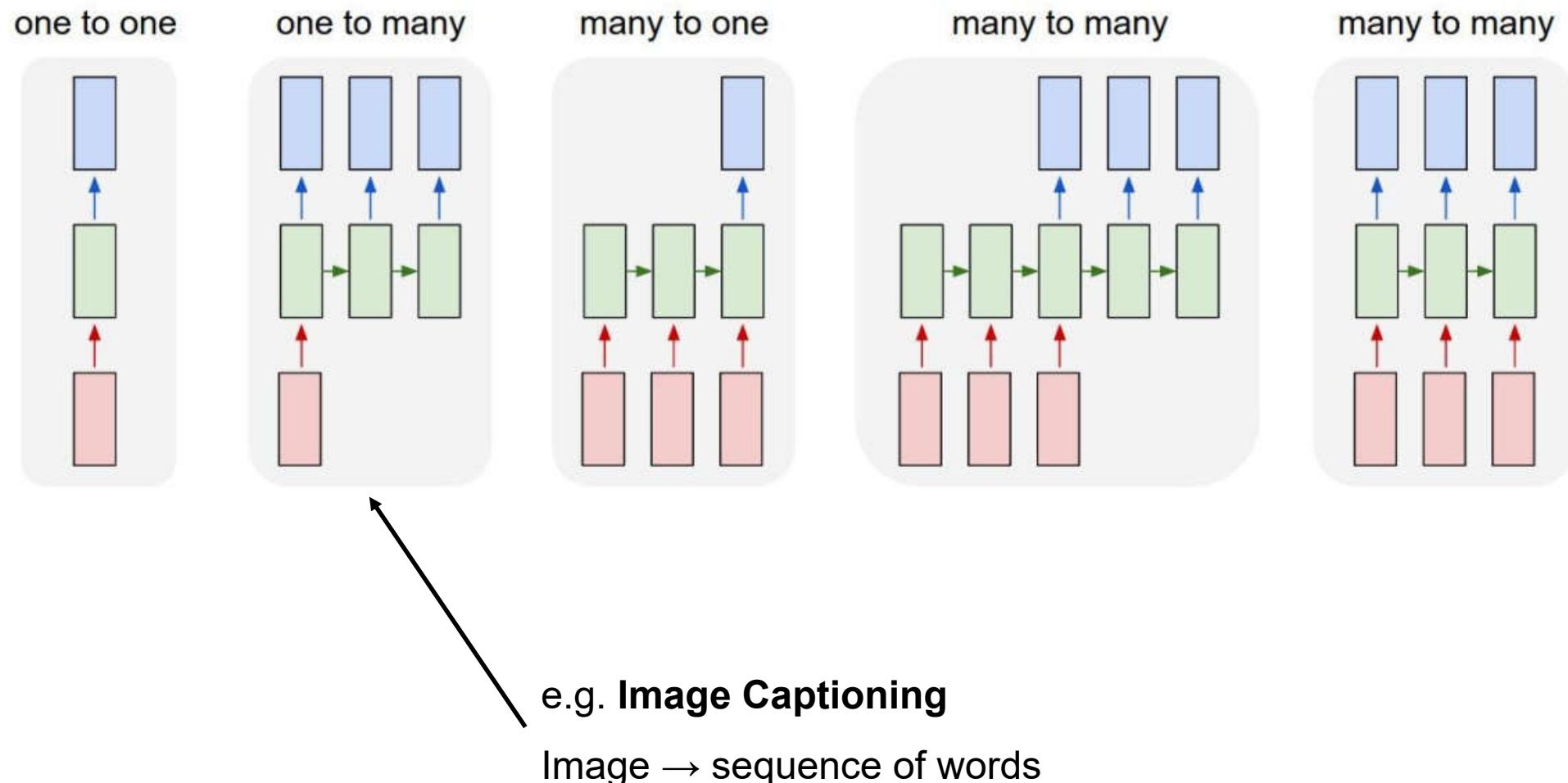


RNN & LSTM

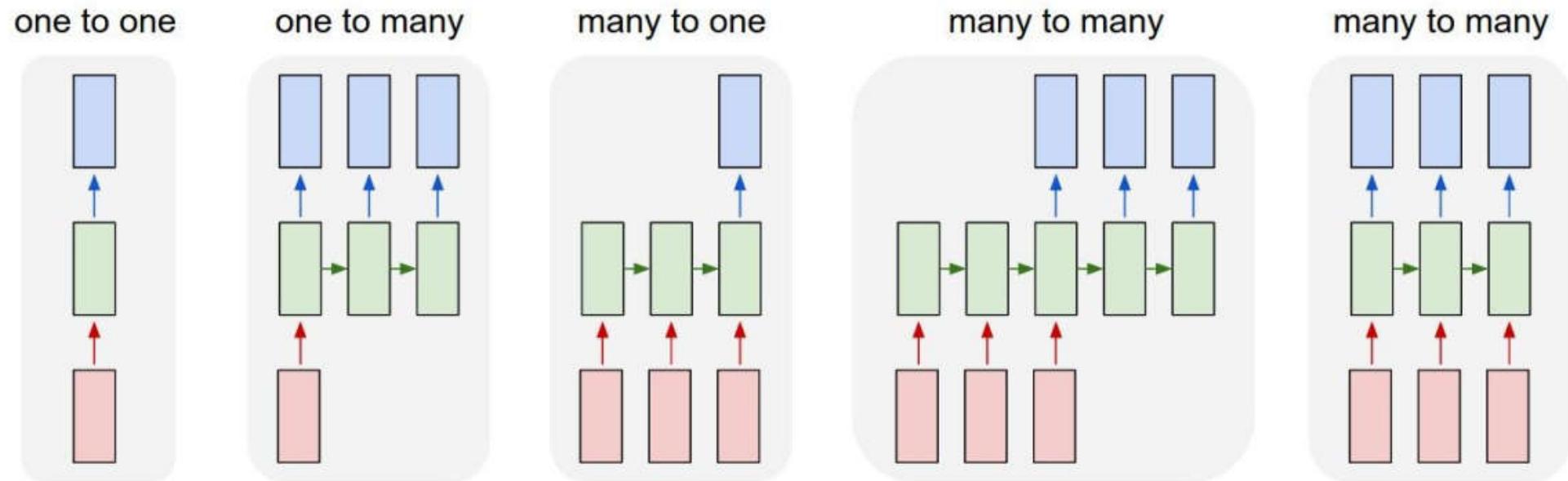
Recurrent Networks Offer a Lot of Flexibility



Recurrent Networks Offer a Lot of Flexibility

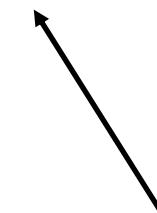
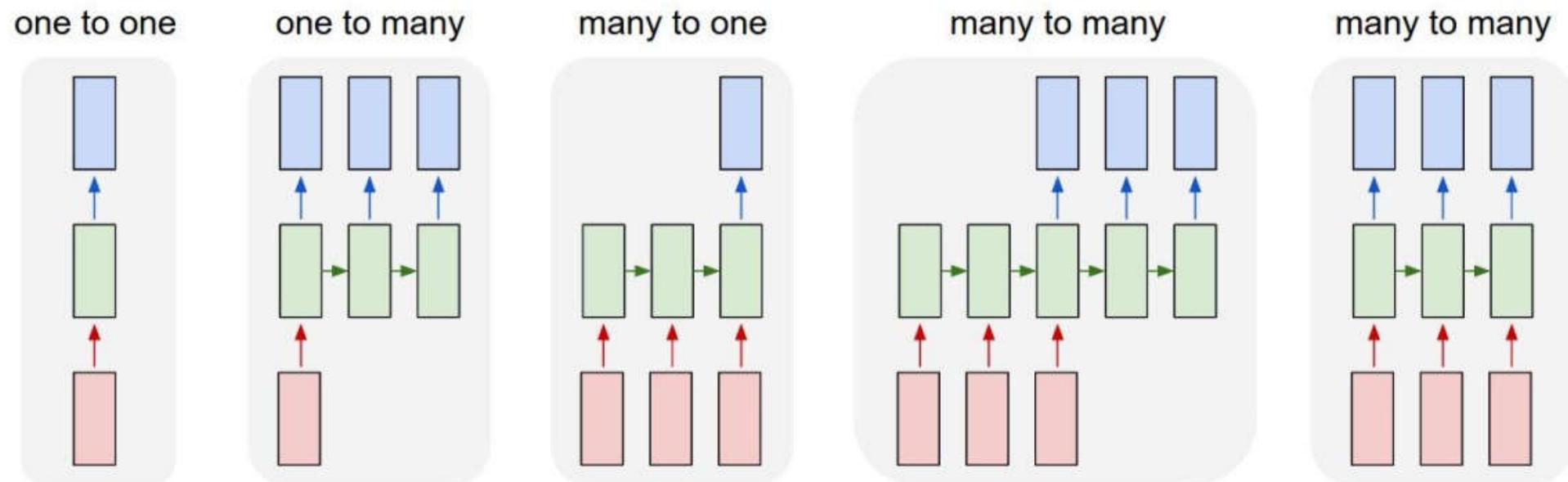


Recurrent Networks Offer a Lot of Flexibility



e.g. **Sentiment Classification**
Sequence of words → sentiment

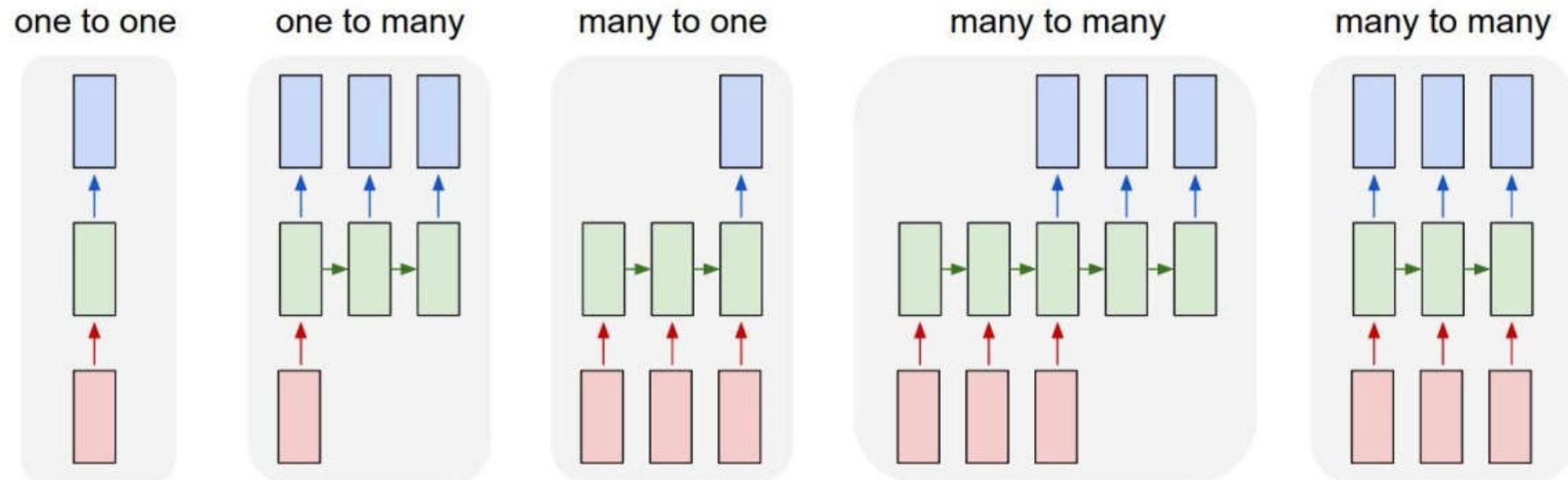
Recurrent Networks Offer a Lot of Flexibility



e.g. **Machine Translation**

Sequence of words → Sequence of words

Recurrent Networks Offer a Lot of Flexibility



e.g. Video classification on frame level

Sequential Processing of Fixed Inputs

“Multiple Object Recognition with Visual Attention,” Ba et al., **ICLR2015** (Toronto and Google DeepMind)

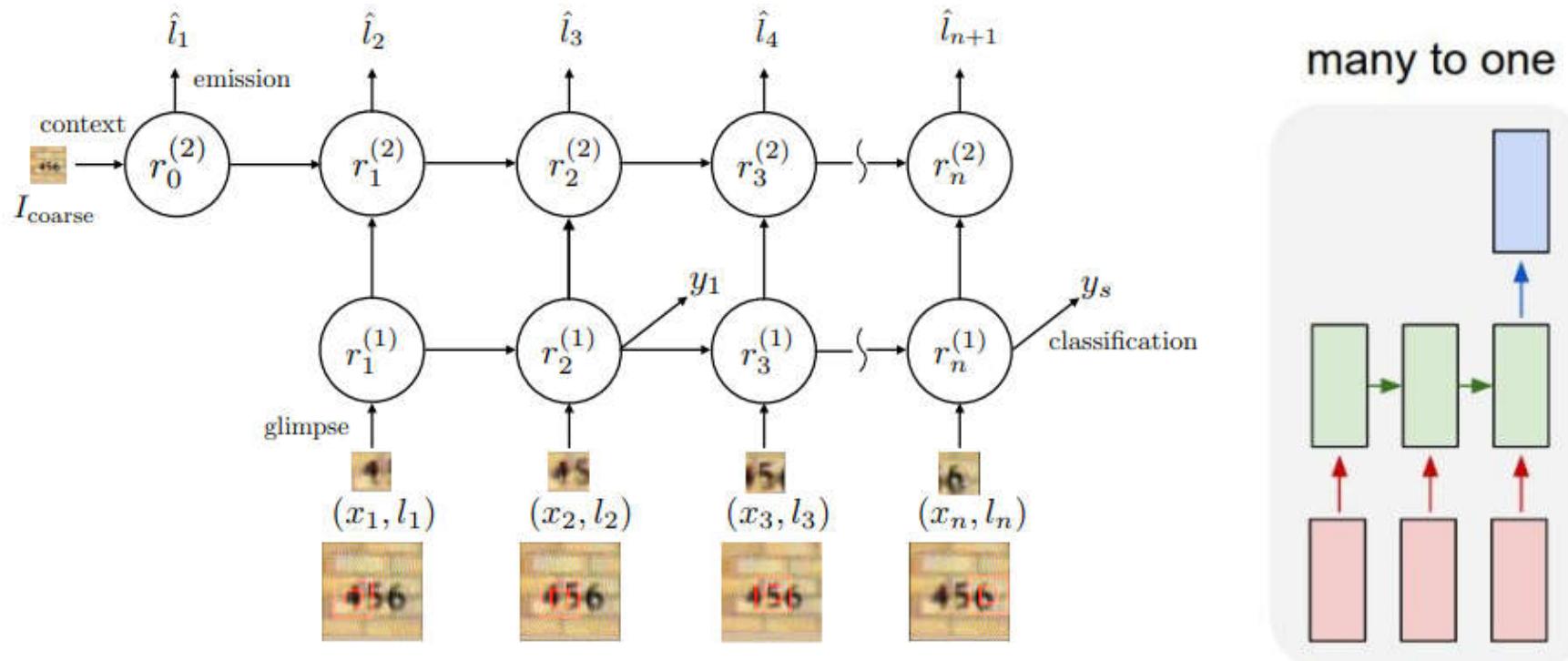


Figure: The deep recurrent attention model

Sequential Processing of Fixed Outputs

“DRAW: A Recurrent Neural Network for Image Generation,” Gregor et al.

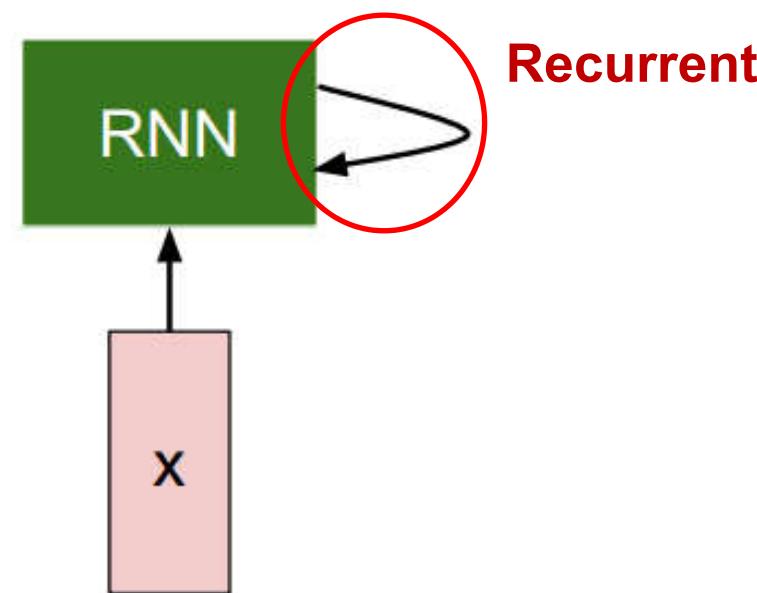
ICML2015 (Google DeepMind)

Generate House Number

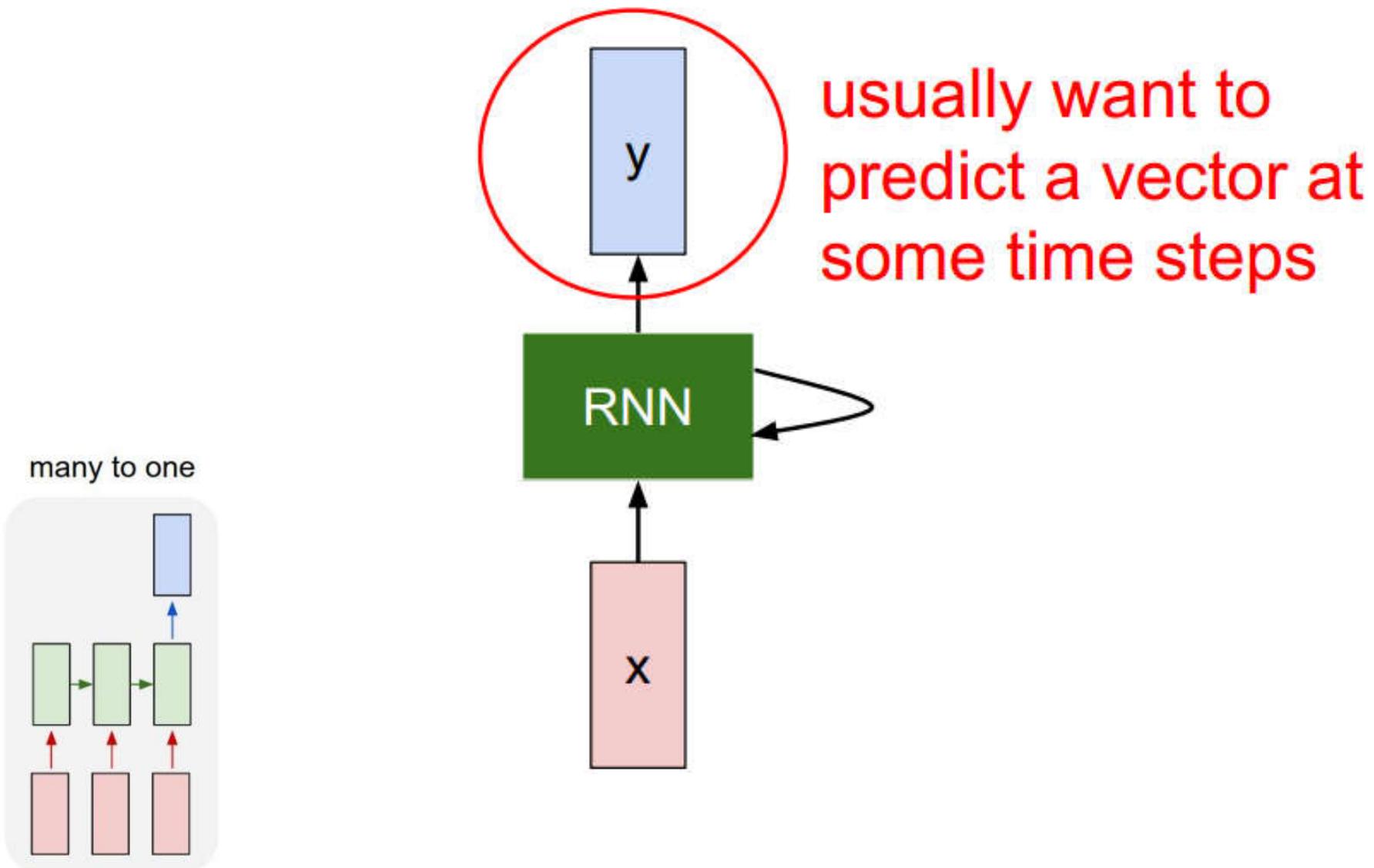


Reading MNIST

Recurrent Neural Network



Recurrent Neural Network

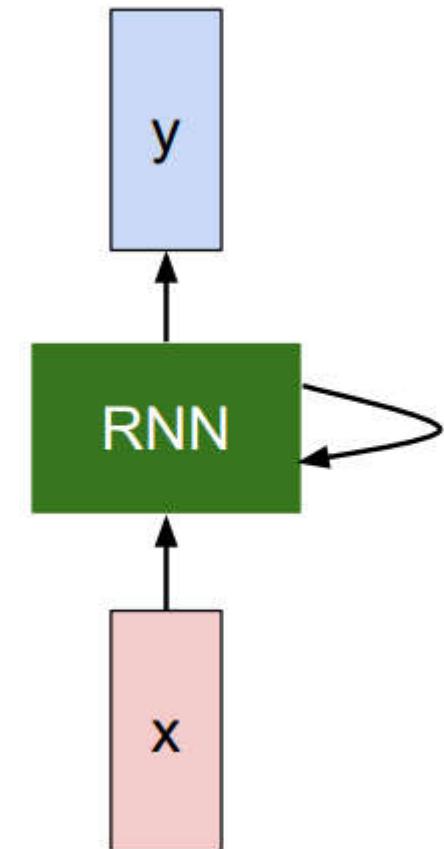


Recurrent Neural Network

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

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

new state / old state input vector at
 \ some time step
some function
with parameters W

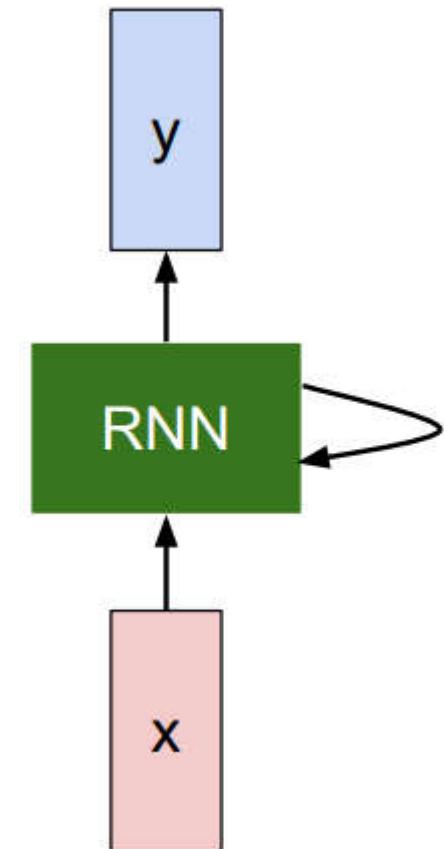


Recurrent Neural Network

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

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

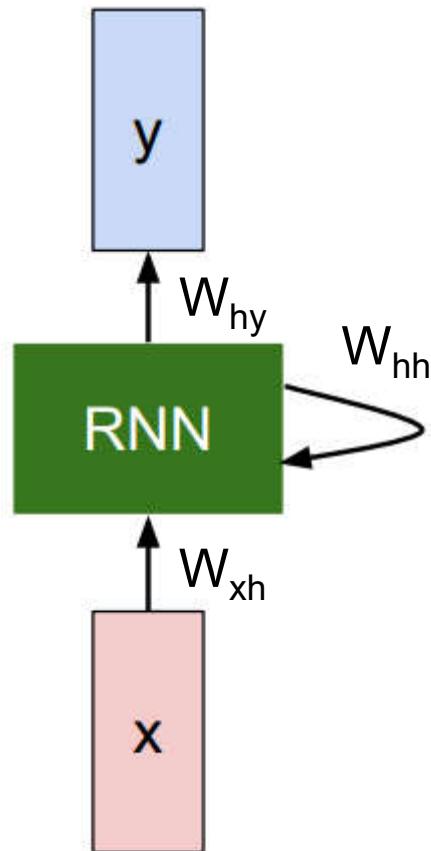
new state / old state input vector at
some function | some time step
with parameters W



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

(Vanilla) Recurrent Neural Network

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



$$\begin{aligned} h_t &= f_W(h_{t-1}, x_t) \\ & \downarrow \\ h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned}$$

Review of the movie application

- Positive
- Negative
- Neural

Example of Character-Level Language Model

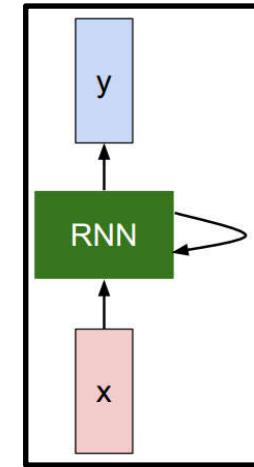
Vocabulary:

[h, e, l, o]

Example training

sequence:

“hello”



Example of Character-Level Language Model

Vocabulary:

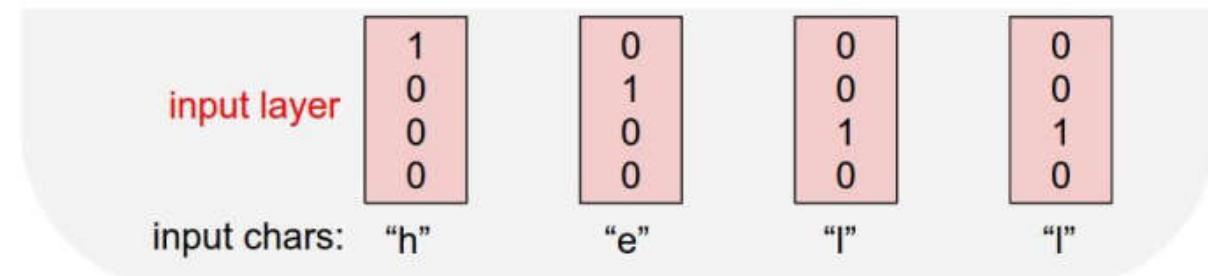
[h, e, l, o]

Example training

sequence:

“hello”

**One Hot
Representation**



Example of Character-Level Language Model

Vocabulary:

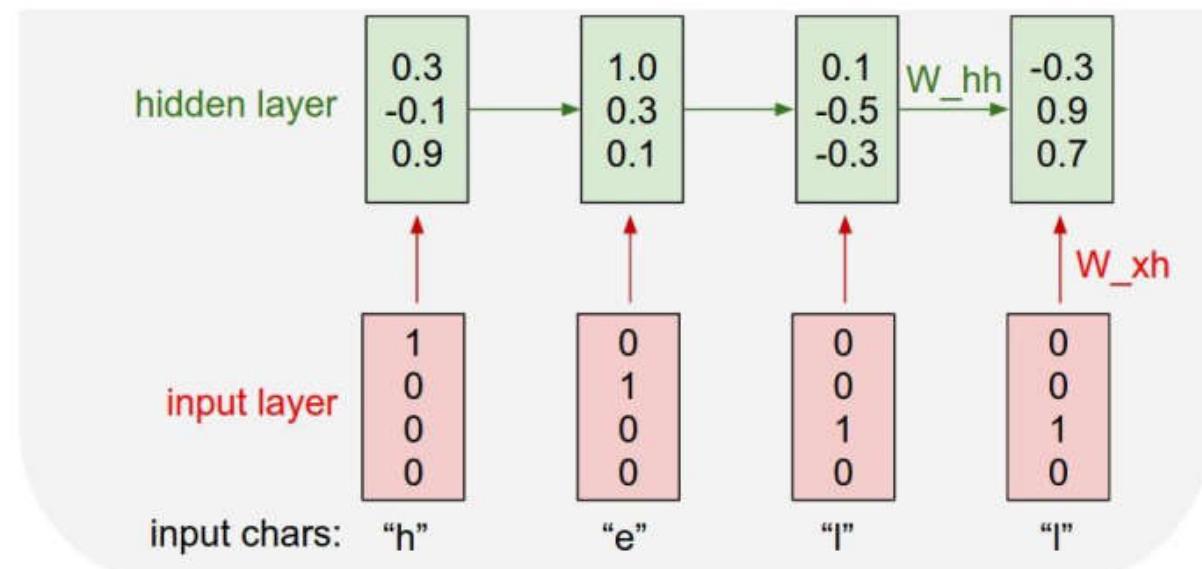
[h, e, l, o]

Example training

sequence:

“hello”

**One Hot
Representation**



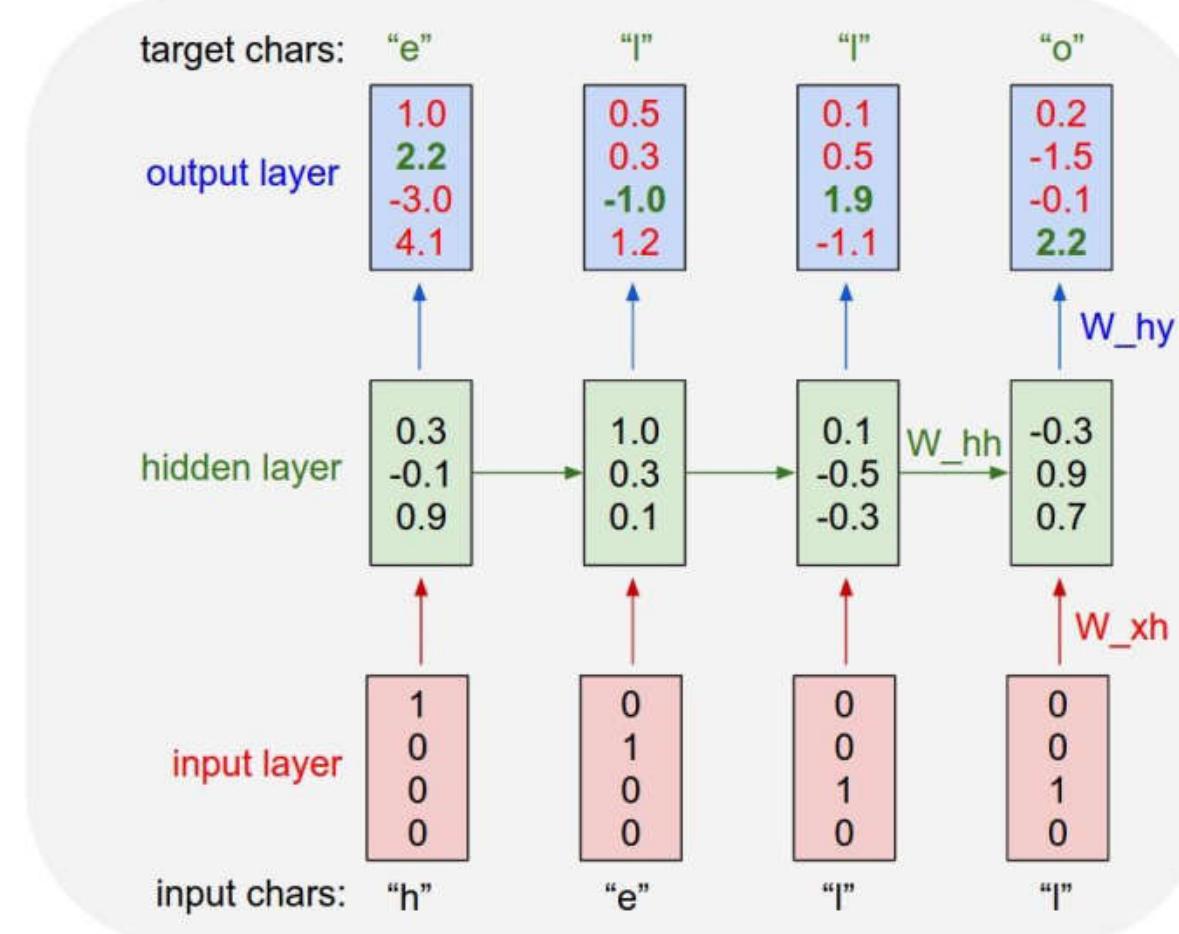
Example of Character-Level Language Model

Vocabulary:

[h, e, l, o]

Example training
sequence:
“hello”

**One Hot
Representation**



Example Code of the Vanilla RNN

```


1  #!/usr/bin/python
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() + '\n' * 1000000 # 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-3
19
20 # model parameters
21 Wxh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros(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 NxD array of initial hidden state.
31   returns the loss, gradients on model parameters, and last hidden state
32   """
33
34   xs, hs, ys, ps = ([], [], [], [])
35   hs[-1] = np.copy(hprev)
36   loss = 0
37   next_h = hs[-1]
38   for t in range(len(inputs)):
39     xs[t] = np.zeros((vocab_size, 1)) # encode as 1-hot representation
40     xs[t][inputs[t]] = 1
41     hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
42     ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next char
43     ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next char
44     loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
45   # backward pass: compute gradients using backprop
46   dWxh, dWhh, dbh, dby = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(bh), np.zeros_like(by)
47   dWxh, dWhh, dbh, dby = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(bh), np.zeros_like(by)
48   dhnext = np.zeros_like(hs[0])
49   for t in reversed(range(len(inputs))):
50     dy = np.copy(ps[t])
51     dy[targets[t]] -= 1 # backprop into y
52     dby -= np.dot(dy, hs[t].T)
53     dy -= dby
54     dh = np.dot(Why.T, dy) + dhnext # backprop into h
55     dh[hs[t].T] = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
56     dWhh += np.dot(dh, hs[t-1].T)
57     dWxh += np.dot(dy, xs[t].T)
58     dhnext = np.dot(Whh.T, dh)
59   for param in [dWxh, dWhh, dbh, dby, dhnext]:
60     np.clip(param, -5, 5, out=param) # clip to mitigate exploding gradients
61   return loss, dWxh, dWhh, dbh, dby, dhnext[inputs[1]:]


```

112 lines of Python

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

Example Code of the Vanilla RNN

[min-char-rnn.py](#) gist

```

# Minimal character-level Recurrent NN model, written by Andrej Karpathy (@karpathy)
# and LICENSE
# https://gist.github.com/karpathy/5f59ecfc187df7d1c57a99718d2bfb6

import numpy as np
from collections import deque

# inputs
data = open('input.txt', 'r').read() # should be simple plain text file
chars = set(data)
vocab_size = len(chars)
char2idx = {ch:i for i,ch in enumerate(chars)}
idx2char = {i:ch for ch,i in char2idx.items()}
text_size = len(data)
print '%d characters, %d unique.' % (text_size, vocab_size)

# model parameters
D = 256 # hidden layer size (or number of hidden units to build the layer)
H = 100 # hidden layer size
M = 200 # input layer size (or number of input units to build the layer)
learning_rate = 1e-1

# training hyperparameters
n_s = 100 # number of hidden states to use in model
n_c = 100 # number of characters to use in model
n_h = 100 # number of hidden units per character
n_i = 100 # number of input units per character
n_w = 100 # number of weight units per character

def init_weights(nin, nout, name):
    """Initialize weights for a layer, using a uniform prior distribution and
    scaling it by 1/sqrt(nin).nin is number of input units, nout is number of output units.
    name is layer name, for logging purposes.
    """
    if nin == 0:
        nin = 1
    scale = 1.0 / sqrt(nin)
    w = np.random.uniform(-scale, scale, (nout, nin))
    if name is not None:
        np.add.at(w, (0, 0), 0.01) # bias for first unit
    return w

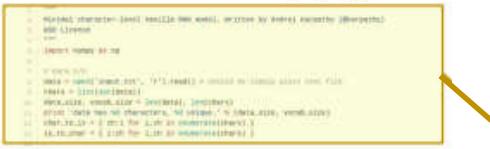
def sigmoid(x):
    """Compute sigmoid of x"""
    return 1.0 / (1 + np.exp(-x))

def softmax(x):
    """Compute softmax of x"""
    x -= x.max()
    e_x = np.exp(x)
    return e_x / e_x.sum()

def one_hot_encode(x, n_c):
    """One-hot encode x (input character) into a vector of size n_c (number of characters)"""
    x_idx = np.zeros(n_c)
    x_idx[x] = 1
    return x_idx

def forward(X, h0, Wx, Wh, b):
    """Forward pass: compute output and hidden state from input X and previous hidden state h0.
    X is a sequence of integers from 0 to n_c-1, with length n_i.
    h0 is initial hidden state, np.zeros((n_h, 1)) if none.
    Wx is weight matrix, shape (n_h, n_c).
    Wh is weight matrix, shape (n_h, n_h).
    b is bias vector, shape (n_h, 1).
    return y, h1
    """
    n_h, n_c = Wx.shape[0], Wx.shape[1]
    y, h1 = np.zeros((n_c, 1)), np.zeros((n_h, 1))
    h0 = h0.reshape((n_h, 1))
    for t in range(0, len(X)):
        x_t = one_hot_encode(X[t], n_c)
        h0 = np.tanh(Wx @ x_t + Wh @ h0 + b)
        y_t = softmax(h0)
        y[t] = y_t
        h1[t] = h0
    return y, h1

```



```

1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """

5  import numpy as np

```

```

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) }


```

Example Code of the Vanilla RNN

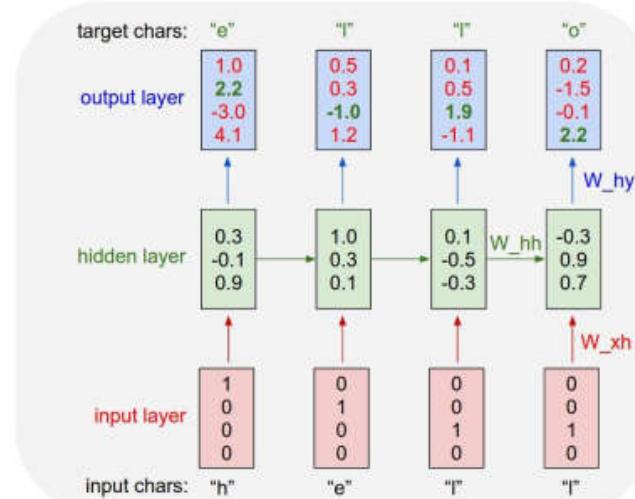
min-char-rnn.py gist

```
#
# MinCharRNN.py: character-level vanilla RNN model, written by Andrej Karpathy (AndrejX)
#
# MIT License
#
# Author: Andrej Karpathy
# Email: akarpathy@cs.toronto.edu
# GitHub: https://github.com/karpathy
# Data: https://raw.githubusercontent.com/karpathy/minnilib/master/minilm-dict.txt
# Model: https://raw.githubusercontent.com/karpathy/minnilib/master/minilm-vocab.npz
# Data URL: https://www.cs.toronto.edu/~karpathy/minnilib/minilm-dict.txt
# Model URL: https://www.cs.toronto.edu/~karpathy/minnilib/minilm-vocab.npz
# Model Size: ~6MB
# Training Time: ~1 day on GPU (NVIDIA 1080Ti)
# Accuracy: ~94% on RNN-3 (3 hidden layers, 128 units each)
# Accuracy: ~96% on RNN-4 (4 hidden layers, 128 units each)
# Accuracy: ~97% on RNN-5 (5 hidden layers, 128 units each)
# Accuracy: ~97.5% on RNN-6 (6 hidden layers, 128 units each)
# Accuracy: ~98% on RNN-7 (7 hidden layers, 128 units each)
# Accuracy: ~98.5% on RNN-8 (8 hidden layers, 128 units each)
# Accuracy: ~99% on RNN-9 (9 hidden layers, 128 units each)
# Accuracy: ~99.5% on RNN-10 (10 hidden layers, 128 units each)
# Accuracy: ~99.8% on RNN-11 (11 hidden layers, 128 units each)
# Accuracy: ~99.9% on RNN-12 (12 hidden layers, 128 units each)
# Accuracy: ~99.95% on RNN-13 (13 hidden layers, 128 units each)
# Accuracy: ~99.98% on RNN-14 (14 hidden layers, 128 units each)
# Accuracy: ~99.99% on RNN-15 (15 hidden layers, 128 units each)
# Accuracy: ~99.995% on RNN-16 (16 hidden layers, 128 units each)
# Accuracy: ~99.998% on RNN-17 (17 hidden layers, 128 units each)
# Accuracy: ~99.999% on RNN-18 (18 hidden layers, 128 units each)
# Accuracy: ~99.9995% on RNN-19 (19 hidden layers, 128 units each)
# Accuracy: ~99.9998% on RNN-20 (20 hidden layers, 128 units each)
# Accuracy: ~99.9999% on RNN-21 (21 hidden layers, 128 units each)
# Accuracy: ~99.99995% on RNN-22 (22 hidden layers, 128 units each)
# Accuracy: ~99.99998% on RNN-23 (23 hidden layers, 128 units each)
# Accuracy: ~99.99999% on RNN-24 (24 hidden layers, 128 units each)
# Accuracy: ~99.999995% on RNN-25 (25 hidden layers, 128 units each)
# Accuracy: ~99.999998% on RNN-26 (26 hidden layers, 128 units each)
# Accuracy: ~99.999999% on RNN-27 (27 hidden layers, 128 units each)
# Accuracy: ~99.9999995% on RNN-28 (28 hidden layers, 128 units each)
# Accuracy: ~99.9999998% on RNN-29 (29 hidden layers, 128 units each)
# Accuracy: ~99.9999999% on RNN-30 (30 hidden layers, 128 units each)
# Accuracy: ~99.99999995% on RNN-31 (31 hidden layers, 128 units each)
# Accuracy: ~99.99999998% on RNN-32 (32 hidden layers, 128 units each)
# Accuracy: ~99.99999999% on RNN-33 (33 hidden layers, 128 units each)
# Accuracy: ~99.999999995% on RNN-34 (34 hidden layers, 128 units each)
# Accuracy: ~99.999999998% on RNN-35 (35 hidden layers, 128 units each)
# Accuracy: ~99.999999999% on RNN-36 (36 hidden layers, 128 units each)
# Accuracy: ~99.9999999995% on RNN-37 (37 hidden layers, 128 units each)
# Accuracy: ~99.9999999998% on RNN-38 (38 hidden layers, 128 units each)
# Accuracy: ~99.9999999999% on RNN-39 (39 hidden layers, 128 units each)
# Accuracy: ~99.99999999995% on RNN-40 (40 hidden layers, 128 units each)
# Accuracy: ~99.99999999998% on RNN-41 (41 hidden layers, 128 units each)
# Accuracy: ~99.99999999999% on RNN-42 (42 hidden layers, 128 units each)
# Accuracy: ~99.999999999995% on RNN-43 (43 hidden layers, 128 units each)
# Accuracy: ~99.999999999998% on RNN-44 (44 hidden layers, 128 units each)
# Accuracy: ~99.999999999999% on RNN-45 (45 hidden layers, 128 units each)
# Accuracy: ~99.9999999999995% on RNN-46 (46 hidden layers, 128 units each)
# Accuracy: ~99.9999999999998% on RNN-47 (47 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999% on RNN-48 (48 hidden layers, 128 units each)
# Accuracy: ~99.99999999999995% on RNN-49 (49 hidden layers, 128 units each)
# Accuracy: ~99.99999999999998% on RNN-50 (50 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999% on RNN-51 (51 hidden layers, 128 units each)
# Accuracy: ~99.999999999999995% on RNN-52 (52 hidden layers, 128 units each)
# Accuracy: ~99.999999999999998% on RNN-53 (53 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999% on RNN-54 (54 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999995% on RNN-55 (55 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999998% on RNN-56 (56 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999% on RNN-57 (57 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999995% on RNN-58 (58 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999998% on RNN-59 (59 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999% on RNN-60 (60 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999995% on RNN-61 (61 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999998% on RNN-62 (62 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999% on RNN-63 (63 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999995% on RNN-64 (64 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999998% on RNN-65 (65 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999% on RNN-66 (66 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999995% on RNN-67 (67 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999998% on RNN-68 (68 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999% on RNN-69 (69 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999995% on RNN-70 (70 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999998% on RNN-71 (71 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999% on RNN-72 (72 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999995% on RNN-73 (73 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999998% on RNN-74 (74 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999% on RNN-75 (75 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999995% on RNN-76 (76 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999998% on RNN-77 (77 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999% on RNN-78 (78 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999995% on RNN-79 (79 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999998% on RNN-80 (80 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999% on RNN-81 (81 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999995% on RNN-82 (82 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999998% on RNN-83 (83 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999999% on RNN-84 (84 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999995% on RNN-85 (85 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999998% on RNN-86 (86 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999999% on RNN-87 (87 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999995% on RNN-88 (88 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999998% on RNN-89 (89 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999999% on RNN-90 (90 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999999995% on RNN-91 (91 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999999998% on RNN-92 (92 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999999999% on RNN-93 (93 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999999995% on RNN-94 (94 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999999998% on RNN-95 (95 hidden layers, 128 units each)
# Accuracy: ~99.99999999999999999999999999999% on RNN-96 (96 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999999995% on RNN-97 (97 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999999998% on RNN-98 (98 hidden layers, 128 units each)
# Accuracy: ~99.999999999999999999999999999999% on RNN-99 (99 hidden layers, 128 units each)
# Accuracy: ~99.9999999999999999999999999999995% on RNN-100 (100 hidden layers, 128 units each)
#
# The code below defines the RNN model architecture and training loop. It uses numpy
# for all linear operations and scipy for softmax. The training loop iterates over
# a sequence of characters, unrolls the RNN for seq_length steps, and performs
# backpropagation through time (BPTT) to update the parameters. The code also includes
# a helper function to calculate perplexity and a progress bar for monitoring training
# progress.
```

Initializations

```
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = 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 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
```

recall:



Example Code of the Vanilla RNN

min-char-rnn.py gist

```
1  #!/usr/bin/python
2  # modified from http://www.jamesmcauley.com/research/rnn.html
3  # MIT License
4
5  import numpy as np
6
7  # imports for cvlab
8  from cvlab import *
9
10
11  # imports for tensorflow
12  import tensorflow as tf
13  from tensorflow.python.platform import gfile
14
15  # imports for os
16  import os
17
18  # imports for time
19  import time
20
21  # imports for sys
22  import sys
23
24  # imports for random
25  import random
26
27  # imports for math
28  import math
29
30  # imports for struct
31  import struct
32
33  # imports for time
34  import time
35
36  # imports for cPickle
37  import cPickle
38
39  # imports for numpy
40  import numpy
41
42  # imports for tensorflow
43  import tensorflow
44
45  # imports for cvlab
46  from cvlab import *
47
48
49  # imports for time
50  import time
51
52
53  # imports for os
54  import os
55
56
57  # imports for struct
58  import struct
59
60
61  # imports for time
62  import time
63
64
65  # imports for cPickle
66  import cPickle
67
68
69  # imports for numpy
70  import numpy
71
72
73  # imports for tensorflow
74  import tensorflow
75
76
77  # imports for cvlab
78  from cvlab import *
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0

while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length+1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size,1)) # reset RNN memory
        p = 0 # go from start of data
    inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
    targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '----\n%s\n----' % (txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                   [dWxh, dWhh, dWhy, dbh, dby],
                                   [mWxh, mWhh, mWhy, mbh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

    p += seq_length # move data pointer
    n += 1 # iteration counter
```

Main loop

Example Code of the Vanilla RNN

min-char-rnn.py gist

```

1  #!/usr/bin/python
2  # modified character-level Vanilla RNN model; written by Andrej Karpathy (http://cs.stanford.edu/~karpathy)
3  # and LICENSE
4
5  # MIT License
6
7  # HISTORY
8  # min-char-rnn.py should be able to train class files
9  # chars = listified
10 # print(chars) > len(chars) / vocab_size == vocab_size
11 # character = C where len(C) == len(chars)/vocab_size
12 # character[0] == character[vocab_size]
13 # character[vocab_size] == character[vocab_size]
14
15 # Hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN
18 learning_rate = 1e-1
19
20 # Parameters
21 x_to_ix = {c: i for i, c in enumerate(chars)} # map characters to integers
22 ix_to_char = {i: c for i, c in enumerate(chars)} # map integers to characters
23 Wxh = np.random.randn(hidden_size, len(chars)) * np.sqrt(0.01)
24 Whh = np.random.randn(hidden_size, hidden_size) * np.sqrt(0.01)
25 Why = np.random.randn(len(chars), hidden_size) * np.sqrt(0.01)
26 bh = np.zeros((hidden_size, 1))
27 by = np.zeros((len(chars), 1))
28 hprev = np.zeros((hidden_size, 1))
29
30 # Model parameters
31 n, p = 0, 0
32 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
33 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
34 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
35
36 while True:
37
38     # prepare inputs (we're sweeping from left to right in steps seq_length long)
39     if p+seq_length+1 >= len(data) or n == 0:
40         hprev = np.zeros((hidden_size,1)) # reset RNN memory
41         p = 0 # go from start of data
42
43     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
44     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
45
46
47     # sample from the model now and then
48     if n % 100 == 0:
49         sample_ix = sample(hprev, inputs[0], 200)
50         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
51         print '----\n %s \n----' % (txt, )
52
53     # forward seq_length characters through the net and fetch gradient
54     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
55     smooth_loss = smooth_loss * 0.999 + loss * 0.001
56
57     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
58
59
60     # perform parameter update with Adagrad
61     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
62                                 [dWxh, dWhh, dWhy, dbh, dby],
63                                 [mWxh, mWhh, mWhy, mbh, mby]):
64
65         mem += dparam * dparam
66         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
67
68     p += seq_length # move data pointer
69     n += 1 # iteration counter
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

```

Main loop

```

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length+1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size,1)) # reset RNN memory
        p = 0 # go from start of data
    inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
    targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '----\n %s \n----' % (txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                  [dWxh, dWhh, dWhy, dbh, dby],
                                  [mWxh, mWhh, mWhy, mbh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

    p += seq_length # move data pointer
    n += 1 # iteration counter

```

Example Code of the Vanilla RNN

min-char-rnn.py gist

```

1  #!/usr/bin/python
2  # modified character-level Vanilla RNN model; written by Andrej Karpathy (http://cs.stanford.edu/~karpathy)
3  # and LICENSE
4
5  # imports
6  import numpy as np
7  from collections import Counter
8  import math
9  import random
10 from time import time
11
12 # config options
13 hidden_size = 100 # size of hidden layer of neurons
14 seq_length = 25 # number of steps to unroll the RNN for
15 learning_rate = 1.0
16
17 # data I/O
18 data = np.loadtxt('data/char rnn.txt', 'str') # should be plain text file
19 chars = list(set(data))
20 n_vocab = len(chars)
21 char_to_ix = {ch:i for i, ch in enumerate(chars)}
22 ix_to_char = {i:ch for i, ch in enumerate(chars)}
23
24 # hyperparameters
25 hidden_size = 100 # size of hidden layer of neurons
26 seq_length = 25 # number of steps to unroll the RNN for
27 learning_rate = 1.0
28
29 # training parameters
30 n_iters = 60000000 # number of iterations to train for
31 n_hidden = 100 # number of hidden units in RNN
32 n_in = 100 # number of input units in RNN
33 n_out = 100 # number of output units in RNN
34 n_layers = 1 # number of layers in RNN
35 batch_size = 100 # number of training examples per mini-batch
36
37 # for softmax
38 softmax_size = 100 # size of softmax layer
39 softmax_size = n_out # size of softmax layer
40 softmax_size = n_in # size of softmax layer
41 softmax_size = hidden_size # size of softmax layer
42 softmax_size = n_out # size of softmax layer
43 softmax_size = n_in # size of softmax layer
44 softmax_size = hidden_size # size of softmax layer
45 softmax_size = n_out # size of softmax layer
46 softmax_size = n_in # size of softmax layer
47 softmax_size = hidden_size # size of softmax layer
48 softmax_size = n_out # size of softmax layer
49 softmax_size = n_in # size of softmax layer
50 softmax_size = hidden_size # size of softmax layer
51 softmax_size = n_out # size of softmax layer
52 softmax_size = n_in # size of softmax layer
53 softmax_size = hidden_size # size of softmax layer
54 softmax_size = n_out # size of softmax layer
55 softmax_size = n_in # size of softmax layer
56 softmax_size = hidden_size # size of softmax layer
57 softmax_size = n_out # size of softmax layer
58 softmax_size = n_in # size of softmax layer
59 softmax_size = hidden_size # size of softmax layer
60 softmax_size = n_out # size of softmax layer
61 softmax_size = n_in # size of softmax layer
62 softmax_size = hidden_size # size of softmax layer
63 softmax_size = n_out # size of softmax layer
64 softmax_size = n_in # size of softmax layer
65 softmax_size = hidden_size # size of softmax layer
66 softmax_size = n_out # size of softmax layer
67 softmax_size = n_in # size of softmax layer
68 softmax_size = hidden_size # size of softmax layer
69 softmax_size = n_out # size of softmax layer
70 softmax_size = n_in # size of softmax layer
71 softmax_size = hidden_size # size of softmax layer
72 softmax_size = n_out # size of softmax layer
73 softmax_size = n_in # size of softmax layer
74 softmax_size = hidden_size # size of softmax layer
75 softmax_size = n_out # size of softmax layer
76 softmax_size = n_in # size of softmax layer
77 softmax_size = hidden_size # size of softmax layer
78 softmax_size = n_out # size of softmax layer
79 softmax_size = n_in # size of softmax layer
80 softmax_size = hidden_size # size of softmax layer
81 softmax_size = n_out # size of softmax layer
82 softmax_size = n_in # size of softmax layer
83 softmax_size = hidden_size # size of softmax layer
84 softmax_size = n_out # size of softmax layer
85 softmax_size = n_in # size of softmax layer
86 softmax_size = hidden_size # size of softmax layer
87 softmax_size = n_out # size of softmax layer
88 softmax_size = n_in # size of softmax layer
89 softmax_size = hidden_size # size of softmax layer
90 softmax_size = n_out # size of softmax layer
91 softmax_size = n_in # size of softmax layer
92 softmax_size = hidden_size # size of softmax layer
93 softmax_size = n_out # size of softmax layer
94 softmax_size = n_in # size of softmax layer
95 softmax_size = hidden_size # size of softmax layer
96 softmax_size = n_out # size of softmax layer
97 softmax_size = n_in # size of softmax layer
98 softmax_size = hidden_size # size of softmax layer
99 softmax_size = n_out # size of softmax layer
100 softmax_size = n_in # size of softmax layer
101 softmax_size = hidden_size # size of softmax layer
102 softmax_size = n_out # size of softmax layer
103 softmax_size = n_in # size of softmax layer
104 softmax_size = hidden_size # size of softmax layer
105 softmax_size = n_out # size of softmax layer
106 softmax_size = n_in # size of softmax layer
107 softmax_size = hidden_size # size of softmax layer
108 softmax_size = n_out # size of softmax layer
109 softmax_size = n_in # size of softmax layer
110 softmax_size = hidden_size # size of softmax layer
111 softmax_size = n_out # size of softmax layer
112 softmax_size = n_in # size of softmax layer

```

```

81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n%s\n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100 loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101 smooth_loss = smooth_loss * 0.999 + loss * 0.001
102 if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104 # perform parameter update with Adagrad
105 for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                             [dWxh, dWhh, dWhy, dbh, dby],
107                             [mWxh, mWhh, mWhy, mbh, mby]):
108     mem += dparam * dparam
109     param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111 p += seq_length # move data pointer
112 n += 1 # iteration counter

```

Main loop

Example Code of the Vanilla RNN

min-char-rnn.py gist

```
1  #!/usr/bin/python
2  # modified character-level Vanilla RNN model; written by Andrej Karpathy (http://cs.stanford.edu/~karpathy)
3  # and LICENSE
4
5  # imports
6  import numpy as np
7  from collections import Counter
8  import math
9  import random
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
```

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0

while True:
    # prepare inputs (we're sweeping from left to right in steps seq_length long)
    if p+seq_length+1 >= len(data) or n == 0:
        hprev = np.zeros((hidden_size,1)) # reset RNN memory
        p = 0 # go from start of data
    inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
    targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

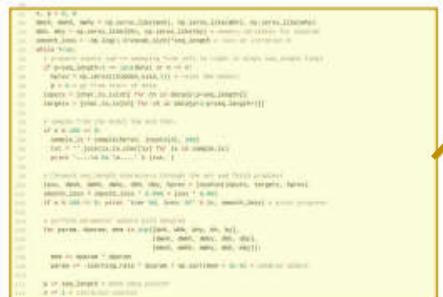
    # sample from the model now and then
    if n % 100 == 0:
        sample_ix = sample(hprev, inputs[0], 200)
        txt = ''.join(ix_to_char[ix] for ix in sample_ix)
        print '----\n%s\n----' % (txt, )

    # forward seq_length characters through the net and fetch gradient
    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
    smooth_loss = smooth_loss * 0.999 + loss * 0.001
    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

    # perform parameter update with Adagrad
    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                  [dWxh, dWhh, dWhy, dbh, dby],
                                  [mWxh, mWhh, mWhy, mbh, mby]):
        mem += dparam * dparam
        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

    p += seq_length # move data pointer
    n += 1 # iteration counter
```

Main loop



Example Code of the

min-char-rnn.py gist

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
 81 n, p = 0, 0
 82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
 83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
 84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
 85 while True:
 86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
 87     if p+seq_length+1 >= len(data) or n == 0:
 88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
 89         p = 0 # go from start of data
 90     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
 91     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
 92
 93     # sample from the model now and then
 94     if n % 100 == 0:
 95         sample_ix = sample(hprev, inputs[0], 200)
 96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
 97         print '----\n%s\n----' % (txt, )
 98
 99     # forward seq_length characters through the net and fetch gradient
100     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                 [dWxh, dWhh, dWhy, dbh, dby],
107                                 [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
```

Main loop

Example Code of the Vanilla RNN

min-char-rnn.py gist

```

# coding: utf-8
# Author: fanrenjie (fanrenjie@msn.com), written by wouter vanmerwijk (wouter)
# See LICENSE
# See LICENSE

import numpy as np
from __future__ import print_function
from six.moves import range,zip
from six import iteritems
from collections import defaultdict, OrderedDict
print("Data has %d characters, %d unique! (%d total_size, %d vocab_size)" % (len(data), len(unique), len(data)/len(unique), len(unique)))
DELTA_CLIP = -0.25 # For L2 norm constraint
DELTA_NORM = 1.0 # For L2 norm constraint

# Parameters
BATCH_SIZE = 100 # Size of batches of neurons
UNK_THRESH = 20 # Number of times to sample from the same char
LEARNING_RATE = 0.05

# Model parameters
W_XH = np.random.rand(vocab_size, hidden_size) * np.sqrt(0.01)
W_HH = np.random.rand(hidden_size, hidden_size) * np.sqrt(0.01)
W_BY = np.random.rand(hidden_size, vocab_size) * np.sqrt(0.01)
B_H = np.zeros((vocab_size, 1)) # hidden state bias
B_Y = np.zeros((vocab_size, 1)) # output bias

def unnormalize(vocab, target):
    """Converts a list of integers to one-hot vectors.
    inputs, targets are both list of integers
    returns the loss, gradients on model parameters, and last hidden state
    """
    if len(vocab) == 0 or len(target) == 0:
        raise ValueError('vocab and target must be non-empty')
    if len(vocab) != len(target):
        raise ValueError('vocab and target must have same length')
    if len(vocab) > vocab_size:
        raise ValueError('vocab size must be less than or equal to %d' % vocab_size)
    if len(target) > vocab_size:
        raise ValueError('target size must be less than or equal to %d' % vocab_size)

    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    loss = 0

    for t in range(len(target)):
        # forward pass
        xs[t] = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
        ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
        loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)

    # backward pass: compute gradients going backwards
    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1 # backprop into y
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dbh += ddraw
        dWxh += np.dot(ddraw, xs[t].T)
        dWhh += np.dot(ddraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, ddraw)

    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

```

Loss function

- forward pass (compute loss)
- backward pass (compute param gradient)

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
44     # backward pass: compute gradients going backwards
45     dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dWhy += np.dot(dy, hs[t].T)
52         dby += dy
53         dh = np.dot(Why.T, dy) + dhnext # backprop into h
54         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55         dbh += ddraw
56         dWxh += np.dot(ddraw, xs[t].T)
57         dWhh += np.dot(ddraw, hs[t-1].T)
58         dhnext = np.dot(Whh.T, ddraw)
59     for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61     return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

```

Example Code of the Vanilla RNN

min-char-rnn.py gist

```

# coding: utf-8
# Python character-level Vanilla RNN model, written by Andrej Karpathy (@karpathy)
# MIT License

# inputs: 1D array of integers, targets: 1D array of integers same size as input
# states: 1D array of floats, shape (n_h, 1)
# n_h: int, number of hidden neurons, n_vocab: n unique characters, vocab_size: vocabulary size
# n_in: int, size of each input vector, n_out: size of each output vector
# n_in == n_out == n_h for this simple example
# learning_rate = 3e-2

# various constants
# hidden layer size (number of hidden neurons)
n_h = 200
# number of hidden layers
n_lyr = 2
# number of characters, n_vocab = len(chars), vocab_size = n_h
n_vocab = len(chars)
# learning rate
learning_rate = 3e-2
# softmax neuron bias
b = np.zeros(n_vocab).T
# hidden state initial state
h0 = np.zeros((n_h, 1))

# forward pass
def forward(inputs, targets):
    inputs, targets = np.asarray(inputs), np.asarray(targets)
    inputs = inputs.reshape(-1, 1) # reshape to 1-of-k representation
    targets = targets.reshape(-1, 1) # reshape to 1-of-n representation
    loss = 0
    hprev = h0
    for t in xrange(len(inputs)):
        x = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
        x[inputs[t]] = 1
        hcur = np.tanh(np.dot(Wxh, x) + np.dot(Whh, hprev) + bh) # hidden state
        ycur = np.dot(Why, hcur) + b # unnormalized log probabilities for next chars
        ps = np.exp(ycur) / np.sum(np.exp(ycur)) # probabilities for next chars
        loss += -np.log(ps[targets[t], 0]) # softmax (cross-entropy loss)
        hprev = hcur
    return loss, ps
    
```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(Why, hs[t]) + b # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
    
```

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

$$y_t = W_{hy}h_t$$

Softmax classifier

Example Code of the Vanilla RNN

[min-char-rnn.py](#) gist

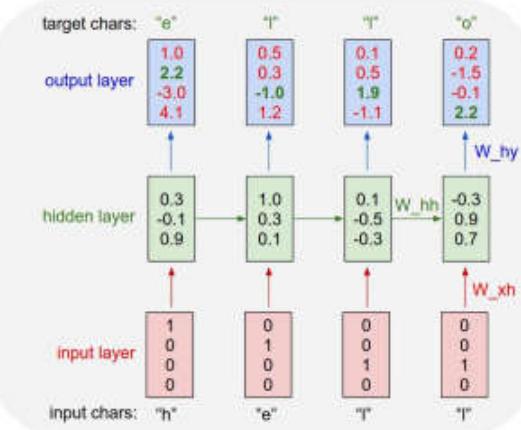
```

1 #!/usr/bin/python
2 # Python character-level RNN, written by Andrej Karpathy (@karpathy)
3 # http://karpathy.github.io/rnn.html
4
5 import numpy as np
6
7 from __future__ import print_function
8
9 # hyperparameters
10 HIDDEN_SIZE = 100 # units in hidden layer; we will have 2 layers
11 INP_VOCAB = 27 # number of input characters
12 OUT_VOCAB = 27 # number of output characters
13 LR = 0.001 # learning rate
14 MAX_EPOCHS = 100 # number of epochs to train for
15 SEED = 12345 # random seed
16
17 # data prep
18 with open('data/tinyshakespeare.txt') as f:
19     data = f.read()
20     data = data[:1000]
21
22 chars = sorted(list(set(data)))
23 char_to_ix = {ch: ix for ix, ch in enumerate(chars)}
24 ix_to_char = {ix: ch for ch, ix in enumerate(chars)}
25 n_chars = len(chars)
26
27 # hyperparameters: weight decay
28 DROPOUT = 0.2 # drop some of hidden layer neurons
29 DROPOUT_TARGET = 0.2 # drop some of input neurons
30 DROPOUT_H = 0.2 # drop some of hidden neurons
31 DROPOUT_X = 0.2 # drop some of input neurons
32
33 # model parameters
34 Wxh = np.random.randn(HIDDEN_SIZE, INP_VOCAB) * 0.01 # input-to-hidden
35 Whh = np.random.randn(HIDDEN_SIZE, HIDDEN_SIZE) * 0.01 # hidden-to-hidden
36 Why = np.random.randn(OUT_VOCAB, HIDDEN_SIZE) * 0.01 # hidden-to-output
37 bhh = np.zeros(HIDDEN_SIZE) # bias for hidden neurons
38 by = np.zeros(OUT_VOCAB) # bias for output neurons
39
40 np.random.seed(SEED)
41
42 def softmax(x):
43     """Inputs x must be 2d array of shape (nbatch, nclass), logits for each class"""
44     exps = np.exp(x - np.max(x, axis=1)[:, np.newaxis])
45     return exps / np.sum(exps, axis=1)[:, np.newaxis]
46
47 def forward(x, h0, Wxh, Whh, Why, bhh, by):
48     """Compute forward pass: input x, hidden state h0, weight matrices Wxh, Whh, Why, bias vectors bhh, by, produce next hidden state h1 and output y"""
49
50     # forward pass
51     h1 = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h0) + bhh) # hidden state
52     y = softmax(np.dot(Why, h1) + by) # output
53
54     return y, h1
55
56 def backward(y, targets, hs, xs, Wxh, Whh, Why, bhh, by, dhnext):
57     """Compute backward pass: input y, target characters, hidden states hs, inputs xs, weight matrices Wxh, Whh, Why, bias vectors bhh, by, previous hidden state dhnext, produce gradients dWxh, dWhh, dWhy, dbhh, dy, dhnext"""
58
59     # backward pass
60     dy = np.copy(ps[t]) # backprop into y
61     dy[targets[t]] -= 1 # backprop into y
62     dWhy += np.dot(dy, hs[t].T)
63     dy += dy
64     dh = np.dot(Why.T, dy) + dhnext # backprop into h
65     ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
66     dbhh += ddraw
67     dWxh += np.dot(ddraw, xs[t].T)
68     dWhh += np.dot(ddraw, hs[t-1].T)
69     dhnext = np.dot(Whh.T, ddraw)
70
71     for dparam in [dWxh, dWhh, dWhy, dbhh, dy]:
72         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
73
74     return loss, dWxh, dWhh, dWhy, dbhh, dy, hs[len(inputs)-1]
    
```

```

44     # backward pass: compute gradients going backwards
45     dwxh, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
46     dbh, dby = np.zeros_like(bhh), np.zeros_like(by)
47     dhnext = np.zeros_like(hs[0])
48     for t in reversed(xrange(len(inputs))):
49         dy = np.copy(ps[t])
50         dy[targets[t]] -= 1 # backprop into y
51         dwhy += np.dot(dy, hs[t].T)
52         dy += dy
53         dh = np.dot(Why.T, dy) + dhnext # backprop into h
54         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55         dbh += ddraw
56         dwxh += np.dot(ddraw, xs[t].T)
57         dwhh += np.dot(ddraw, hs[t-1].T)
58         dhnext = np.dot(Whh.T, ddraw)
59
60         for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
61             np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
62
63     return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
    
```

recall:



Example Code of the Vanilla RNN

min-char-rnn.py gist

```

1 #!/usr/bin/python
2
3 # This file is part of min-char-rnn, written by Andrew Kurszynski (dkursy)
4
5 # Readme:
6
7 #-----#
8 # This code is just a simple toy implementation of an unrolled Vanilla RNN model.
9 # It is not meant for production, or anything other than testing. It is slow, very slow.
10 # The code is meant to show how an RNN works, what its state is at each time step,
11 # and how it generates sequences of characters.
12
13 #-----#
14 # Hyperparameters
15 vocab_size = 256 # size of hidden layer of neurons
16 n_in = 256 # number of inputs we will see per step
17 learning_rate = 1e-4
18
19 #-----#
20 # Model parameters
21 Wxh = np.random.uniform(-0.1, 0.1, (vocab_size, n_in)) # weight from hidden layer to input
22 Whh = np.random.uniform(-0.1, 0.1, (vocab_size, vocab_size)) # weight from hidden layer to hidden layer
23 Why = np.random.uniform(-0.1, 0.1, (vocab_size, vocab_size)) # weight from hidden layer to output
24 bh = np.zeros((vocab_size, 1)) # bias for hidden layer
25 by = np.zeros((vocab_size, 1)) # bias for output layer
26
27 #-----#
28 # Constants: targets, labels
29
30 # Input targets are just list of integers.
31 # Labels is the initial hidden state.
32 # Returns the loss, gradients on word parameters, and last hidden state.
33
34 #-----#
35 # Initialize hidden state
36 # h0, m0, b0 = np.zeros((D, 1), np.float) # initialize hidden state
37 # m0[0] = np.zeros(D) # initialize cell state
38 # b0 = np.zeros(1) # initialize bias
39
40 #-----#
41 # Compute hidden state
42 # zt1 = np.tanh(np.dot(Wxh, xt) + np.dot(Whh, ht) + bh) # hidden state
43 # st1 = softmax(np.dot(Why, ht) + by) # softmax of outputs
44 # p1 = st1 * np.exp(st1) / np.sum(st1) # probability for next chars
45 # zm1 = np.argmax(p1) # next hidden state
46 # xt1 = np.zeros((D, 1), np.float) # next input
47 # xt1[zm1] = 1.0 # set the right input to 1.0
48 # xt1 -= np.random.uniform(0.0, 1.0) # scale input
49 # b1 = np.tanh(b0 + np.sum(st1))
50 # dh1 = (dt1 * st1) * np.tanh(b1).T # dh through tanh
51 # dm1 = np.tanh(b1) * dt1.T # dm through tanh
52 # db1 = np.sum(dt1) # db
53 # st1[zm1] = 0.0 # set the right output to 0.0
54 # xt1[zm1] = 1.0 # set the right input to 1.0
55 # xt1 -= np.random.uniform(0.0, 1.0) # scale input
56 # b2 = np.tanh(b1 + dm1)
57 # dh2 = (dt1 * dm1) * np.tanh(b2).T # dh through tanh
58 # dm2 = np.tanh(b2) * dt1.T # dm through tanh
59 # db2 = np.sum(dt1) # db
60 # st2[zm1] = 0.0 # set the right output to 0.0
61 # xt2[zm1] = 1.0 # set the right input to 1.0
62 # xt2 -= np.random.uniform(0.0, 1.0) # scale input
63
64 #-----#
65 #-----#
66 #-----#
67 #-----#
68
69
70
71
72
73
74
75
76
77
78
79

```



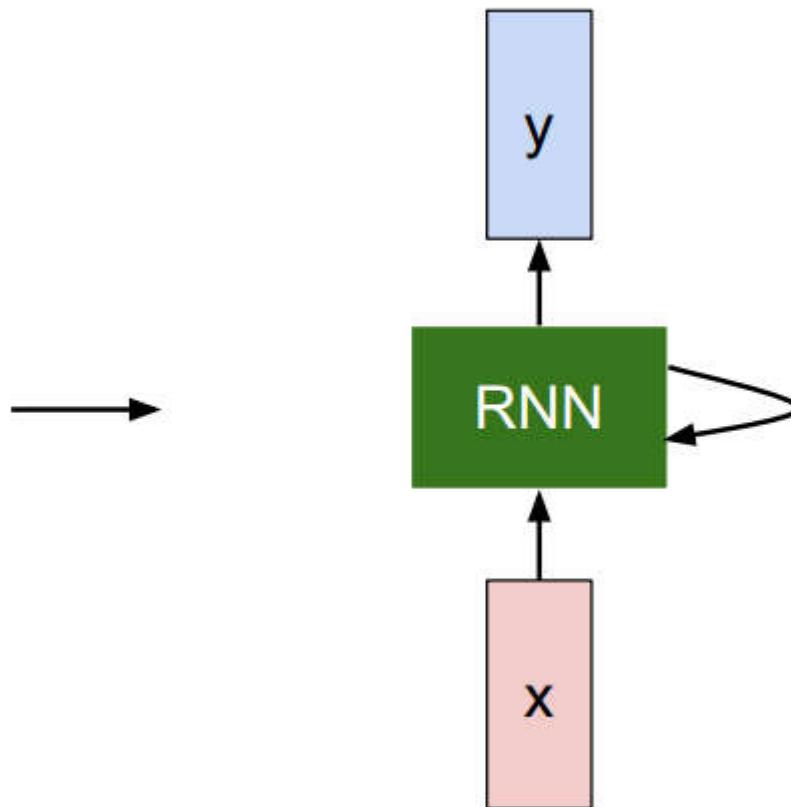
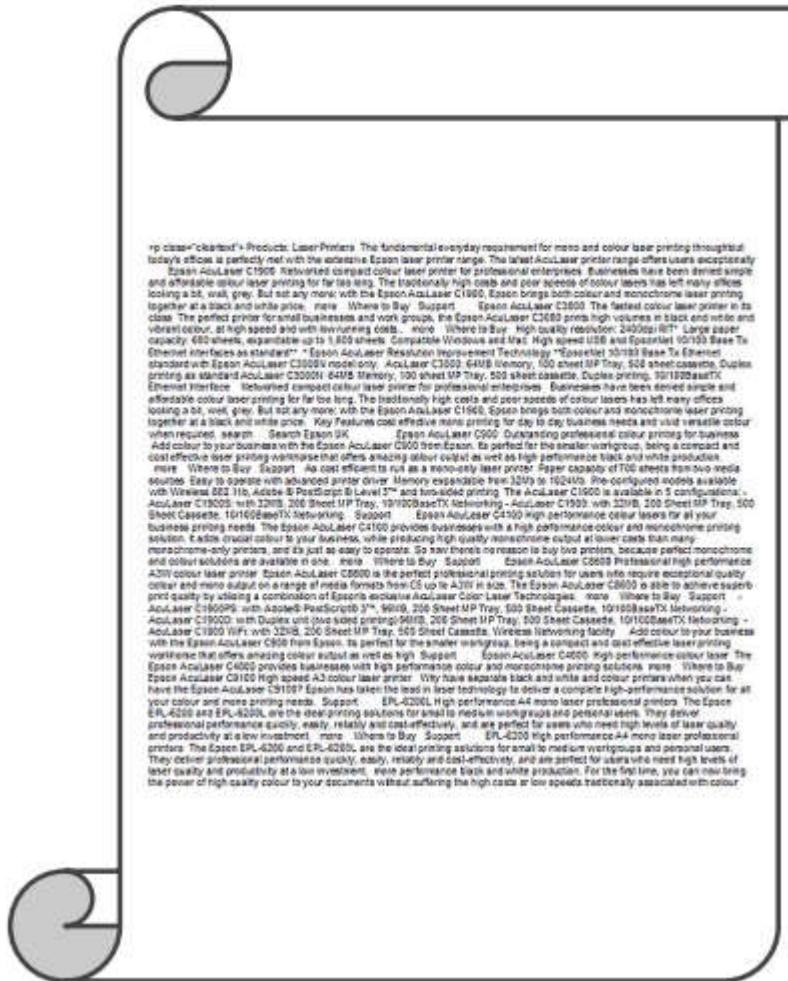
```

def sample(h, seed_ix, n):
    """
    sample a sequence of integers from the model
    h is memory state, seed_ix is seed letter for first time step
    """
    x = np.zeros((vocab_size, 1))
    x[seed_ix] = 1
    ixes = []
    for t in xrange(n):
        h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
        y = np.dot(Why, h) + by
        p = np.exp(y) / np.sum(np.exp(y))
        ix = np.random.choice(range(vocab_size), p=p.ravel())
        x = np.zeros((vocab_size, 1))
        x[ix] = 1
        ixes.append(ix)
    return ixes

```

RNN in Other Works

RNN in Shakespeare Works



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.

RNN in Shakespeare Works

At first:

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

↓ train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwv 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 ofter.

↓ 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.

RNN in Shakespeare Works

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 aps, 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.

RNN in Algebraic Geometry



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

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

Latex source

RNN in Algebraic Geometry

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m_n} = 0$, 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 comparicoly 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 $x, x' \in S'$ 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 X' , and \mathcal{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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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 $X_{\text{spaces},\text{étale}}$ which gives an open subspace of X and T equal to S_{Zar} , 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 $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$ 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 restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

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

Lemma 0.3. In Situation ???. Hence we may assume $q' = 0$.

Proof. We will use the property we see that 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

RNN in Algebraic Geometry

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 $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

$$\begin{array}{ccccc}
 S & \xrightarrow{\quad} & & & \\
 \downarrow & & & & \\
 \xi & \xrightarrow{\quad} & \mathcal{O}_{X'} & \xrightarrow{\quad} & \\
 \text{gor}_s & & \uparrow & \searrow & \\
 & & = \alpha' & \longrightarrow & \\
 & & \downarrow & & \\
 & & = \alpha' & \longrightarrow & \\
 & & & & \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & X \\
 & & & & \downarrow \\
 & & & & d(\mathcal{O}_{X_{X/k}}, \mathcal{G})
 \end{array}$$

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”

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\overline{x}} \dashv (\mathcal{O}_{X_{\text{étale}}}) \rightarrow \mathcal{O}_{X_{\text{ét}}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_{\eta}}^{\text{vir}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . 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 $\mathcal{O}_{X_{\lambda}}$ is a closed immersion, see Lemma ???. This is a sequence of \mathcal{F} is a similar morphism.

RNN Generating C Code



torvalds / linux

Watch 6,584 Star 66,264 Fork 24,042

Code Pull requests 237 Projects 0 Insights

Linux kernel source tree

797,479 commits 1 branch 582 releases contributors View license

Branch: master New pull request Create new file Upload files Find file Clone or download

MatiasBjorling and torvalds ia64: export node_distance function	Latest commit ef78e5e 12 hours ago
Documentation	Merge tag 'xarray-4.20-rc4' of git://git.infradead.org/users/willy/li...
LICENSES	Merge tag 'docs-4.20' of git://git.lwn.net/linux
arch	ia64: export node_distance function
block	SCSI: fix queue cleanup race before queue initialization is done
certs	export.h: remove VMLINUX_SYMBOL() and VMLINUX_SYMBOL_STR()
crypto	crypto: user - Zeroize whole structure given to user space
drivers	Merge tag 'hwmon-for-v4.20-rc5' of git://git.kernel.org/pub/scm/linux...
firmware	kbuild: remove all dummy assignments to obj-
fs	Merge tag 'nfs-for-4.20-4' of git://git.linux-nfs.org/projects/trondm...
include	Merge tag 'dma-mapping-4.20-3' of git://git.infradead.org/users/hch/d...
init	memblock: stop using implicit alignment to SMP_CACHE_BYTES

RNN Generating C Code

```

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 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x000000f) << 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

RNN Generating 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/seteaw.h>
#include <asm/pgproto.h>
```

RNN Generating C Code

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.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;
}

```

Searching for Interpretable Cells

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
    */
```

“Visualizing and Understanding Recurrent Networks,” Andrej Karpathy, Justin Johnson, Li Fei-Fei, **arXiv2015 v2**

Searching for Interpretable Cells



"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Quote detection cell

Searching for Interpretable Cells



Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Line length tracking cell

Searching for Interpretable Cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

If statement cell

Searching for Interpretable Cells

```

/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                    (void **)&df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM \\'%s\\' is invalid\n",
               df->lsm_str);
        ret = 0;
    }
    return ret;
}

```

quote/comment cell

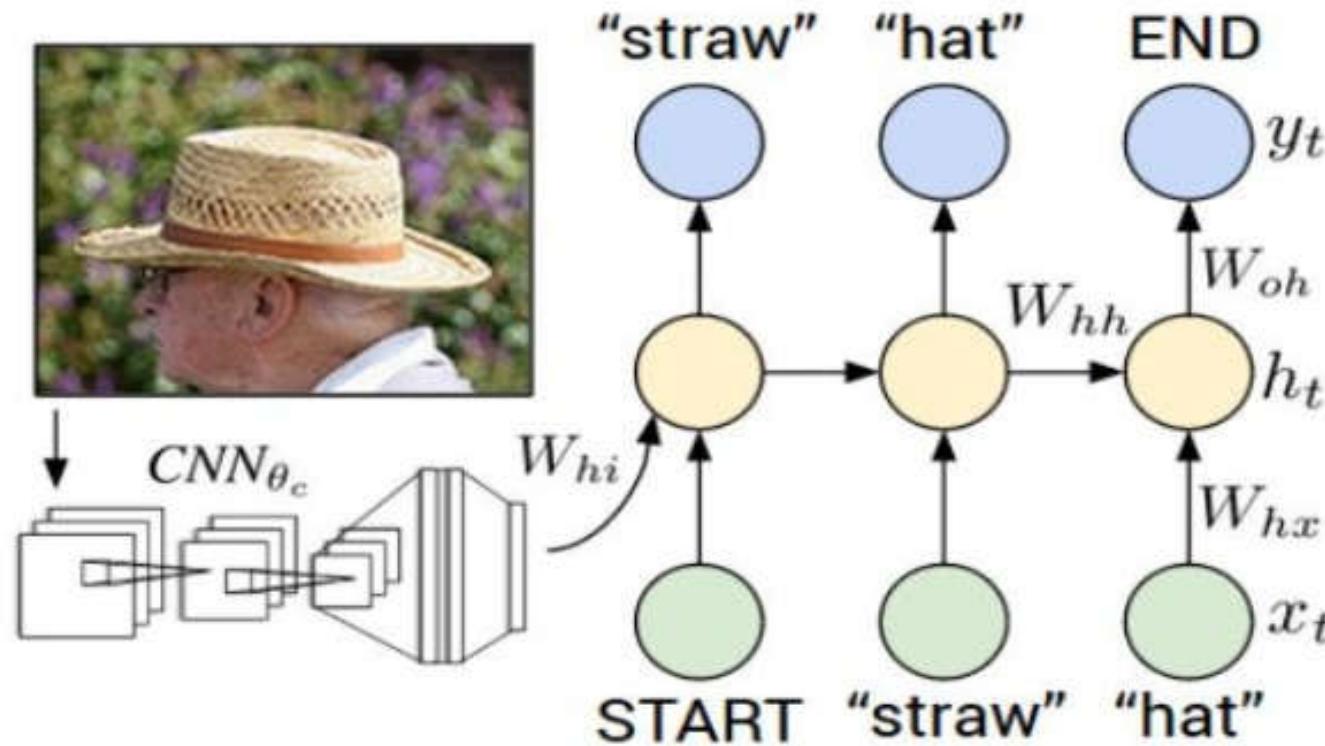
Searching for Interpretable Cells

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

Code depth cell

Image Captioning

Image Captioning



"Explain images with multimodal Recurrent Neural Networks," Mao et al., arXiv2014

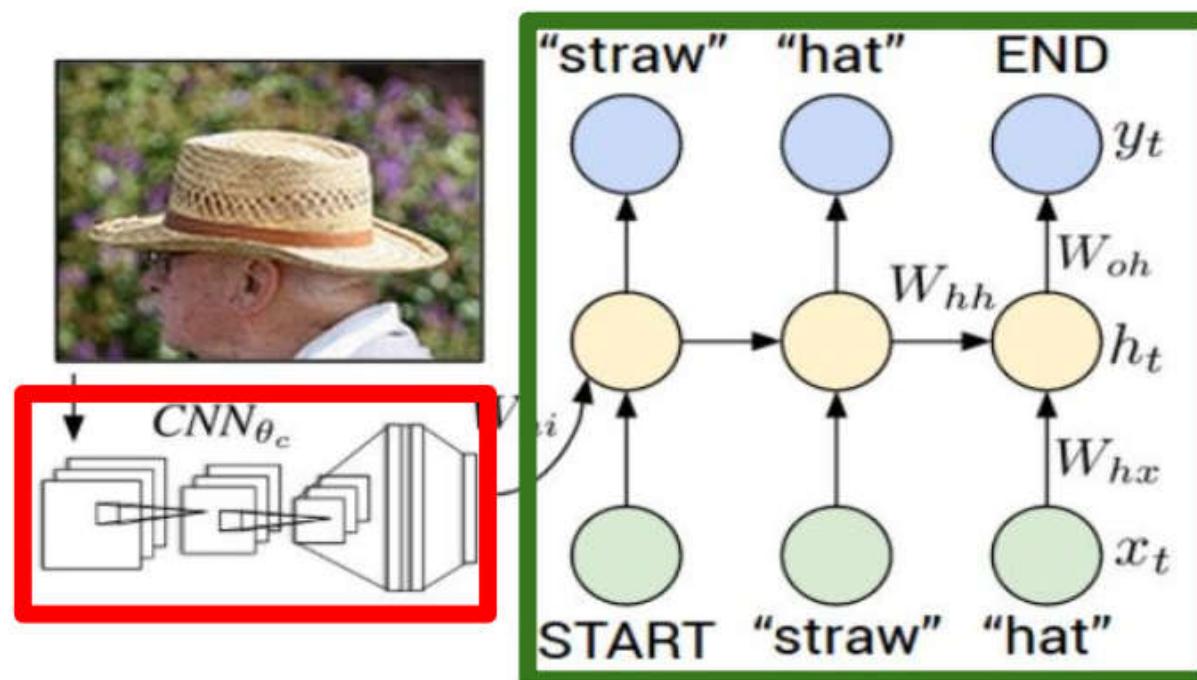
"Deep Visual-semantic alignments for generating image descriptions," Karpathy and Fei-Fei, CVPR2015

"Show and Tell: A neural image caption generator," Vinyals et al., CVPR2015

"Long-term Recurrent Convolutional Networks for visual recognition and description," Donahue et al., CVPR2015

"Learning a recurrent visual representation for image caption generation," Chen and Zitnick, ICML2015

Recurrent Neural Network



Convolutional Neural Network

Image Captioning



test image

Image Captioning



Image Captioning

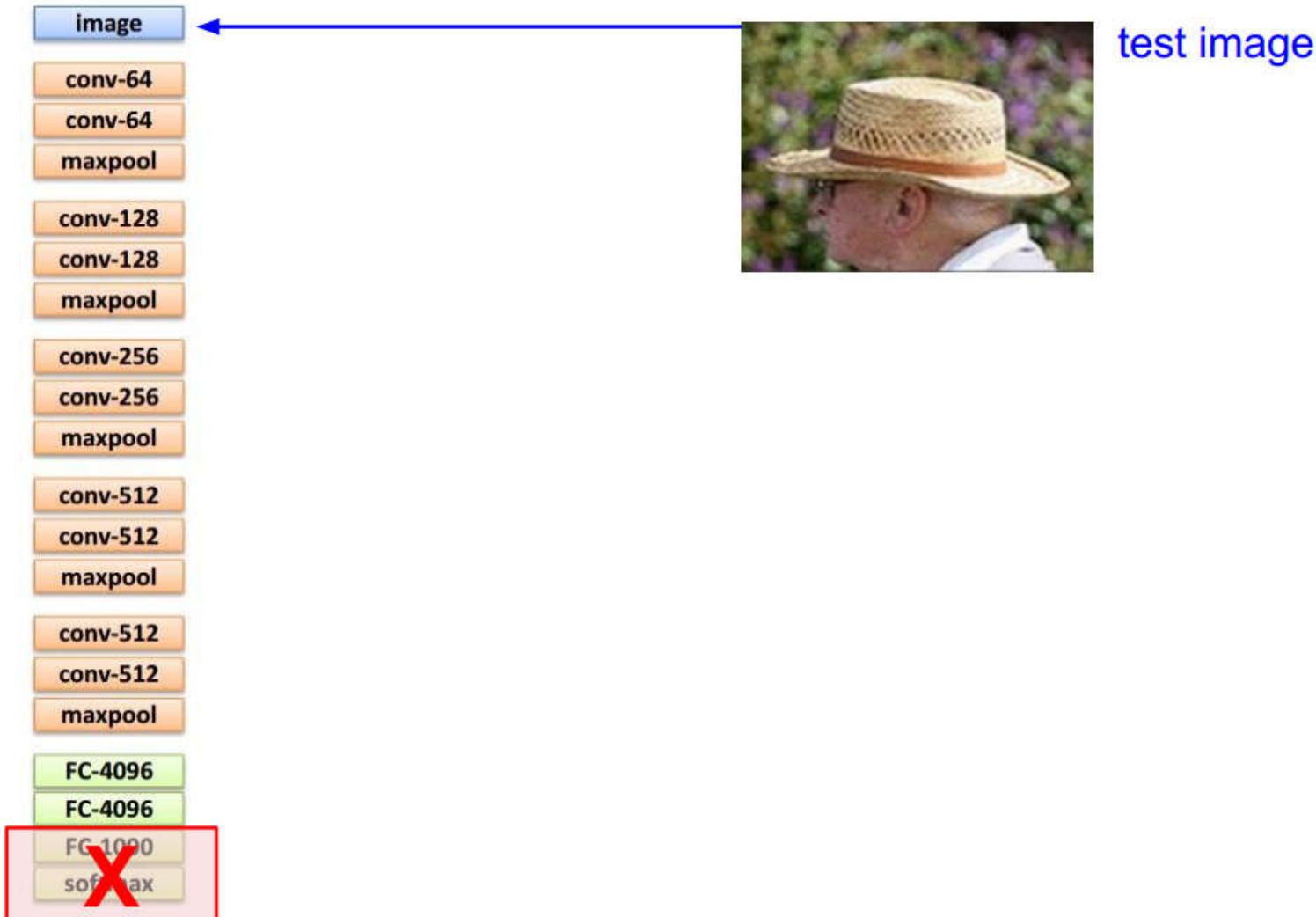


Image Captioning

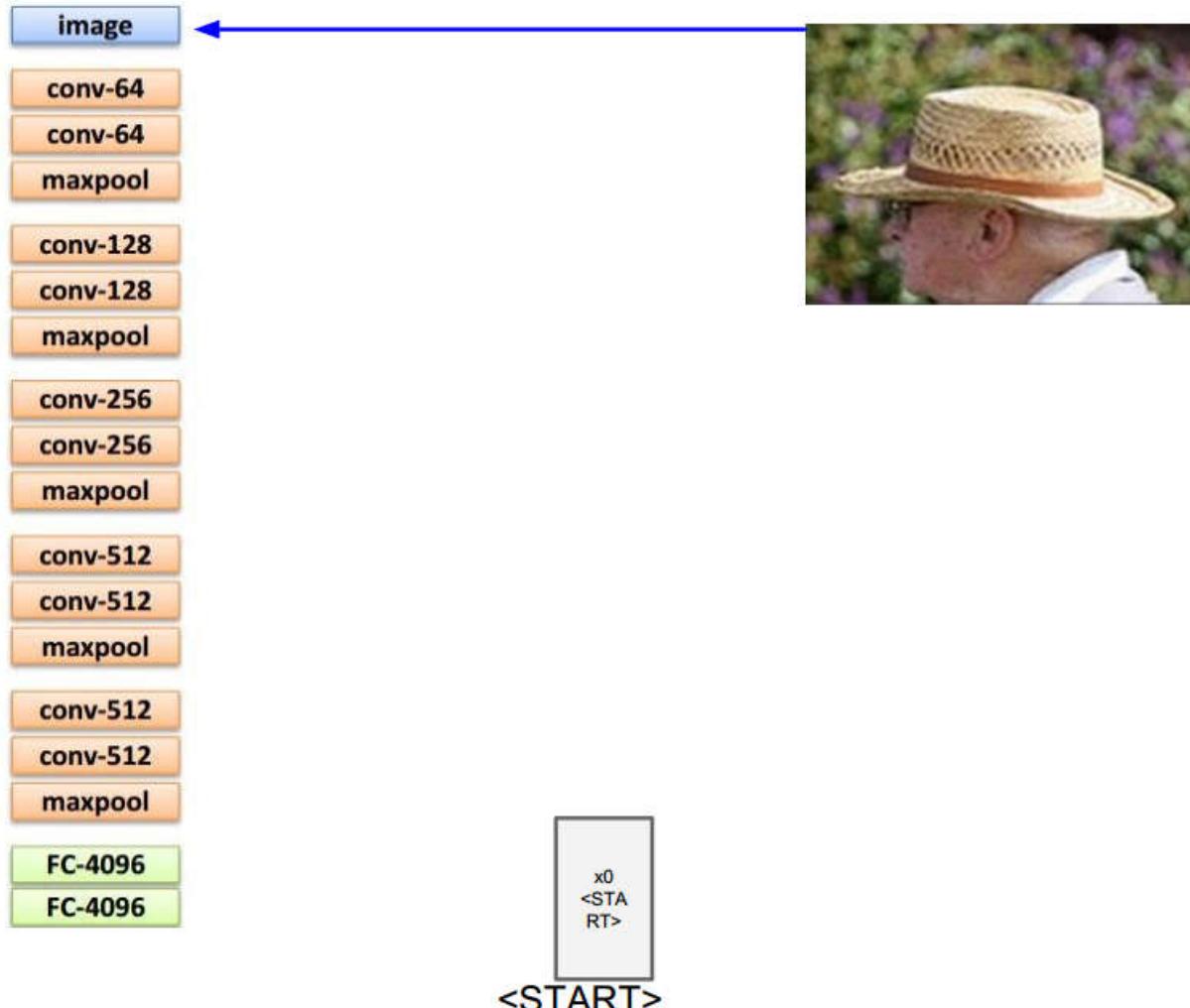
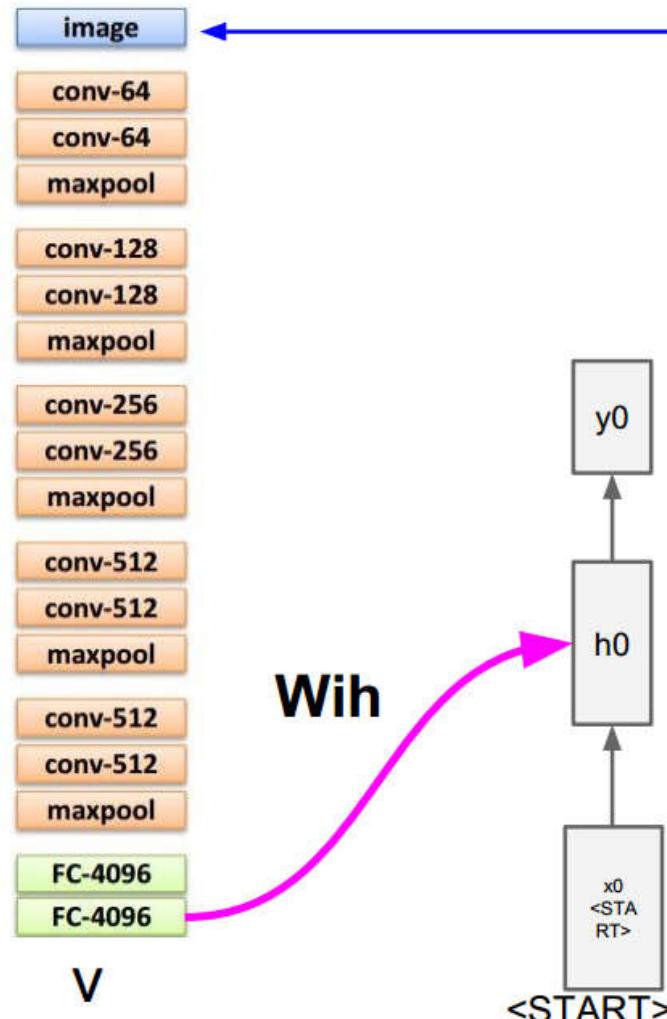


Image Captioning



before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{vh} * v)$$

Image Captioning

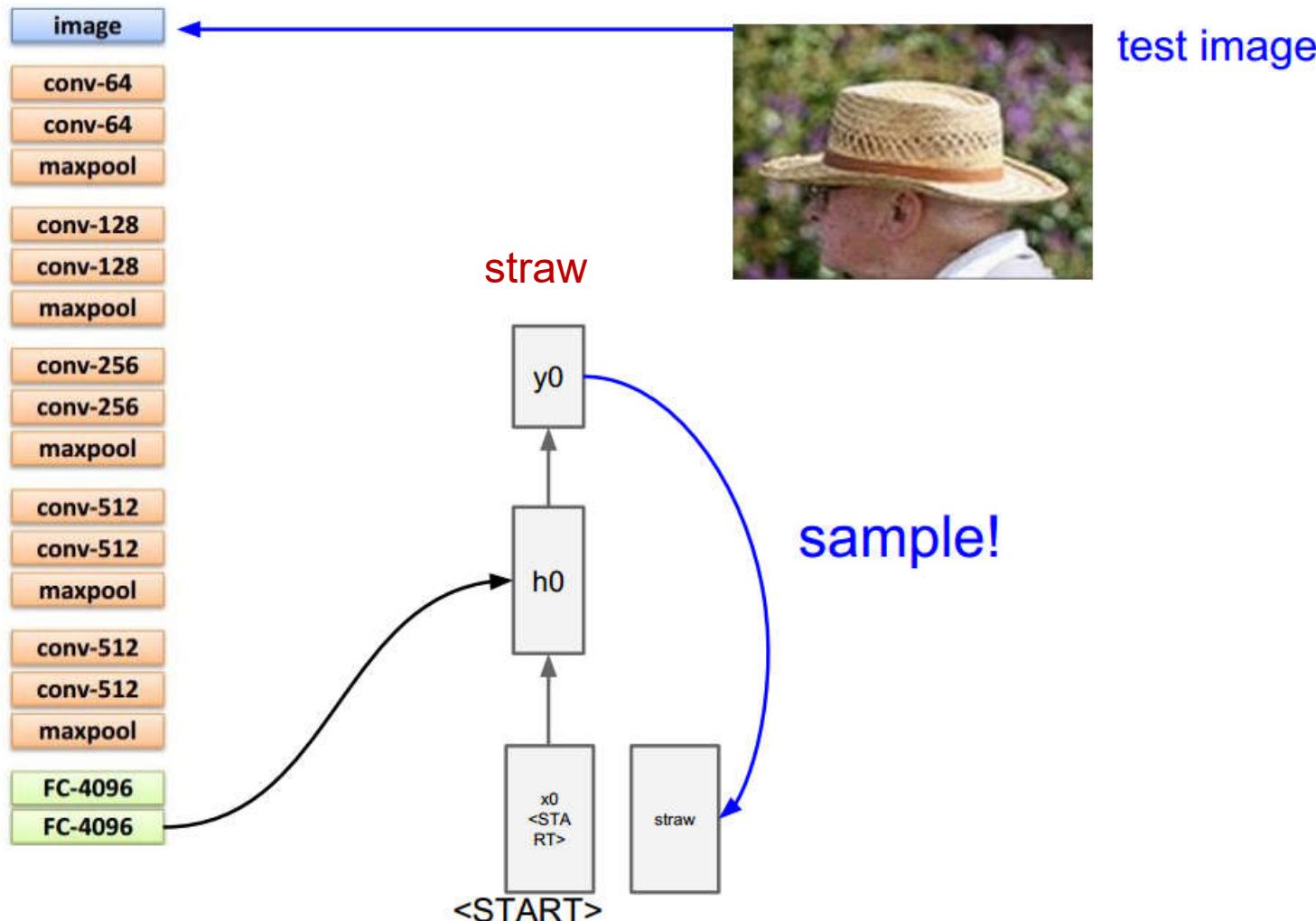


Image Captioning

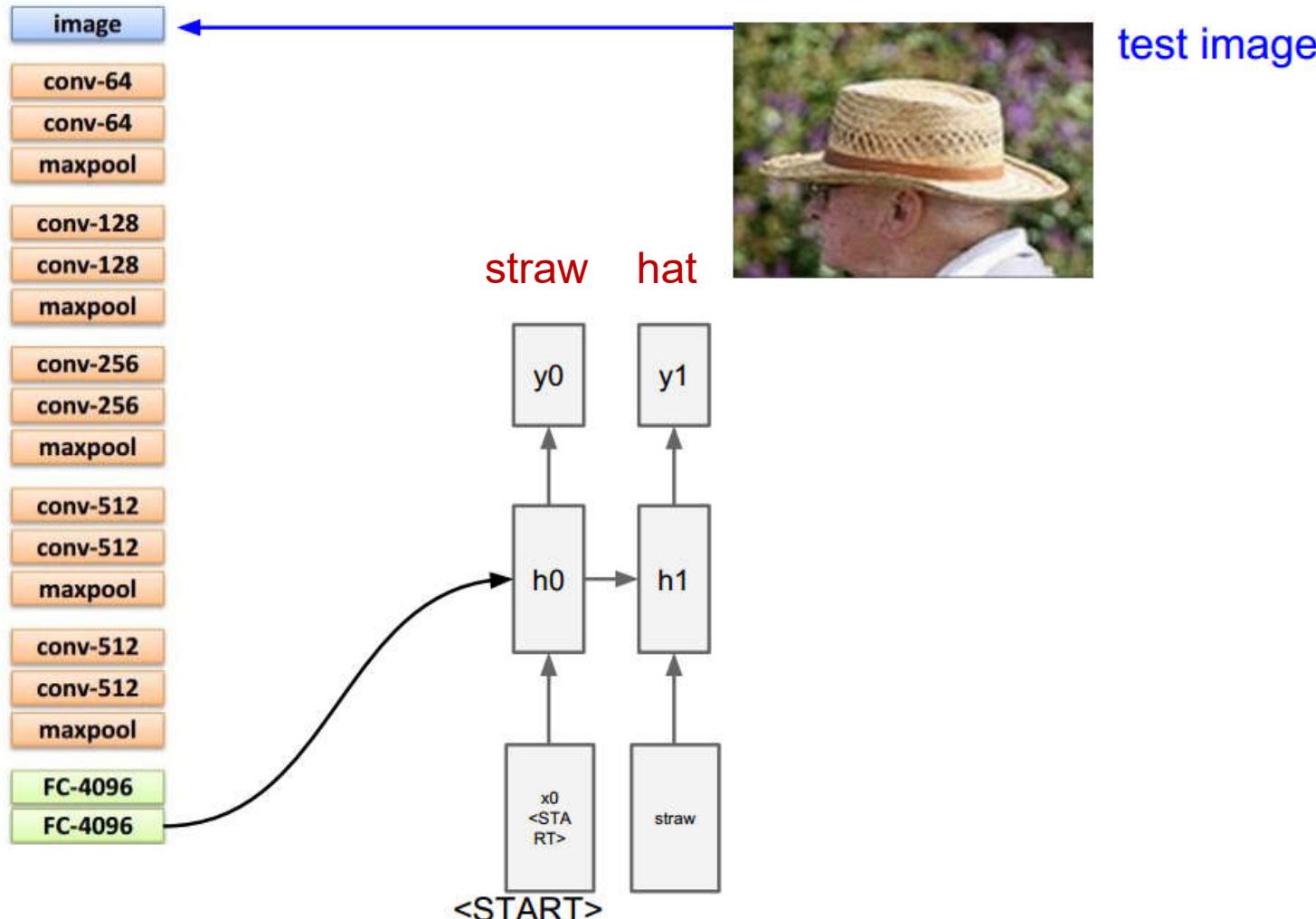


Image Captioning

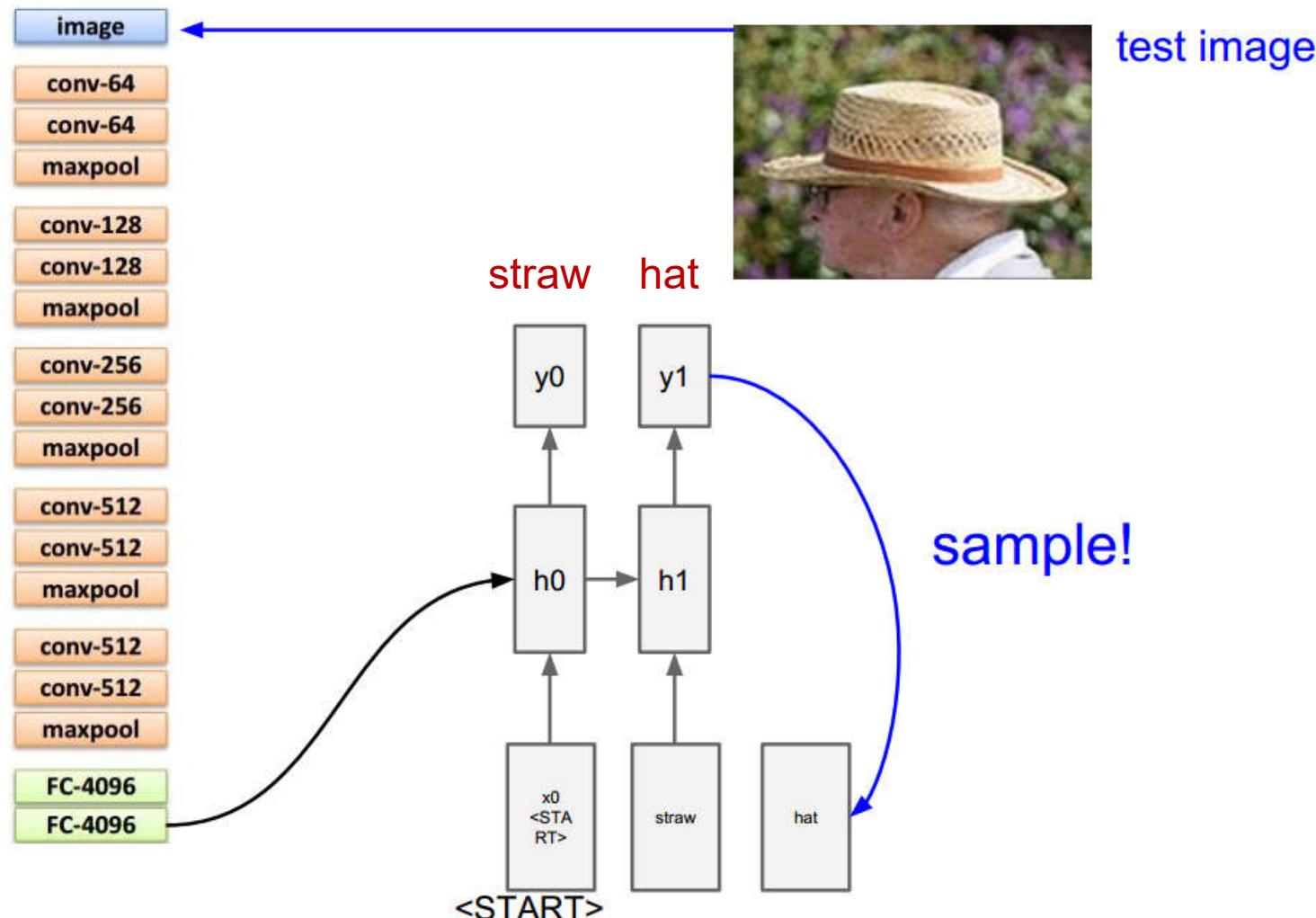


Image Captioning

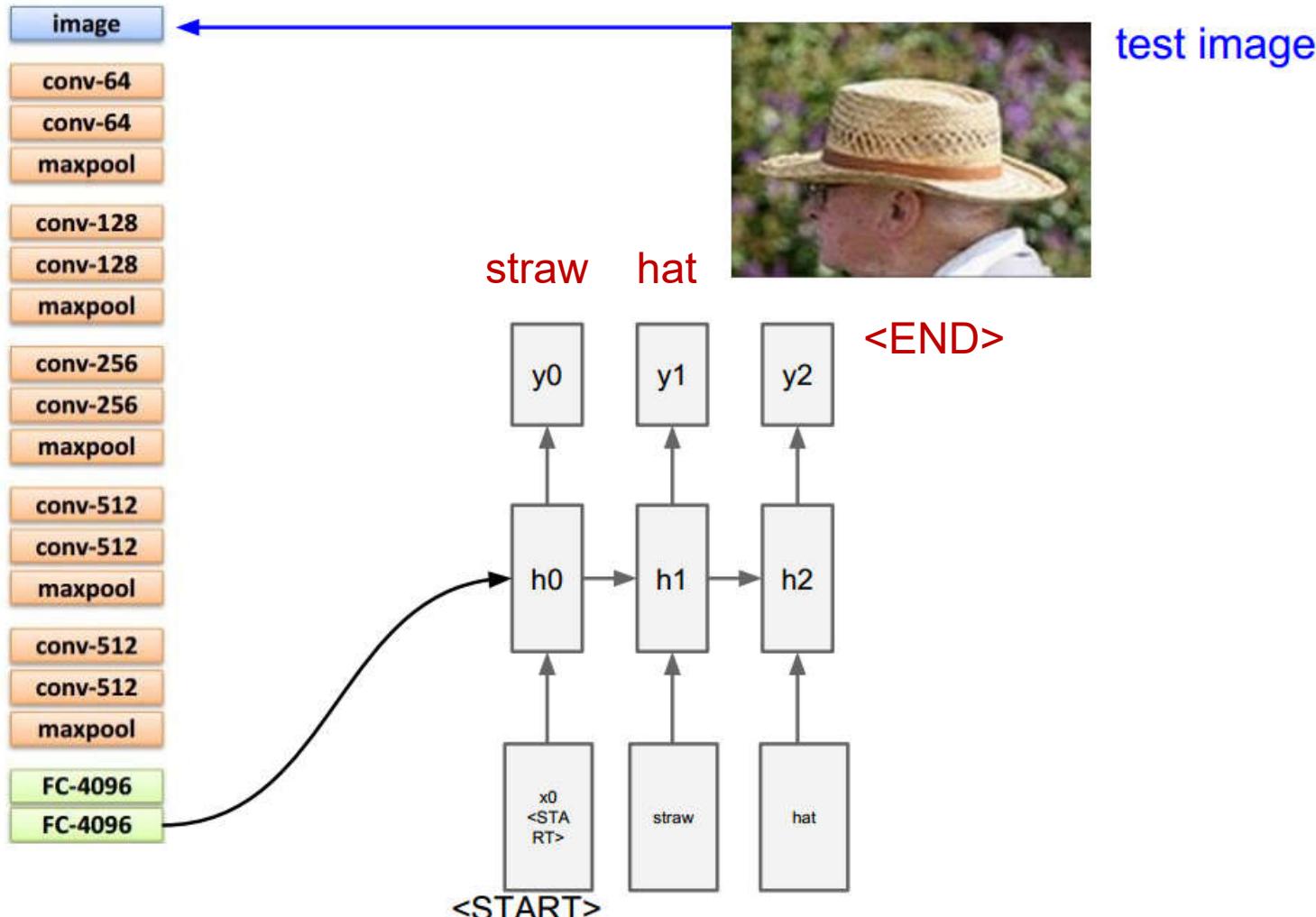


Image Captioning

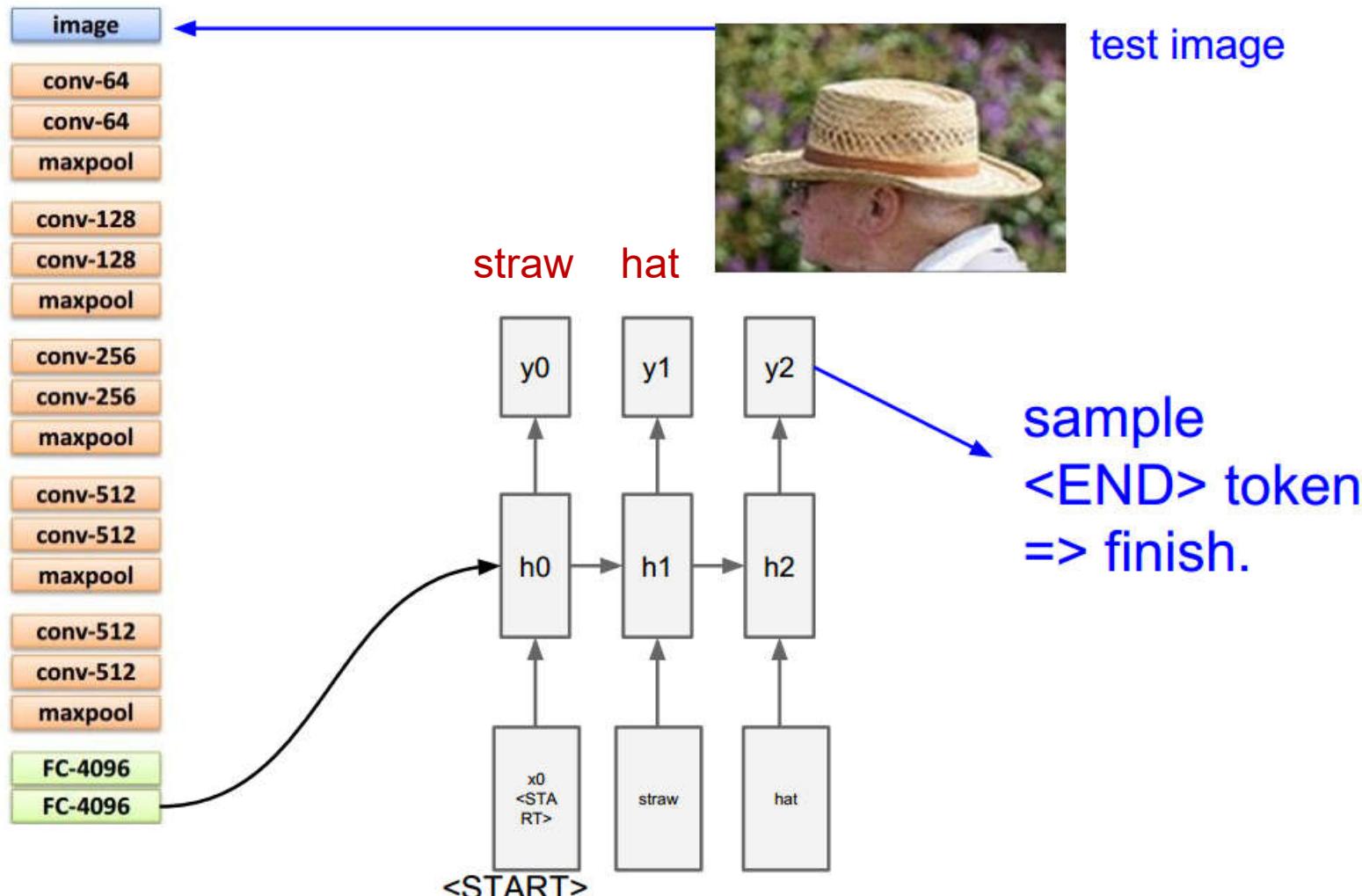


Image Sentence Datasets

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]
<http://mscoco.org/>

AMT: Amazon
Mechanical Turk

Currently:
~120K images
~5 sentences each

Image Captioning Examples



"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."

Image Captioning Examples



"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."

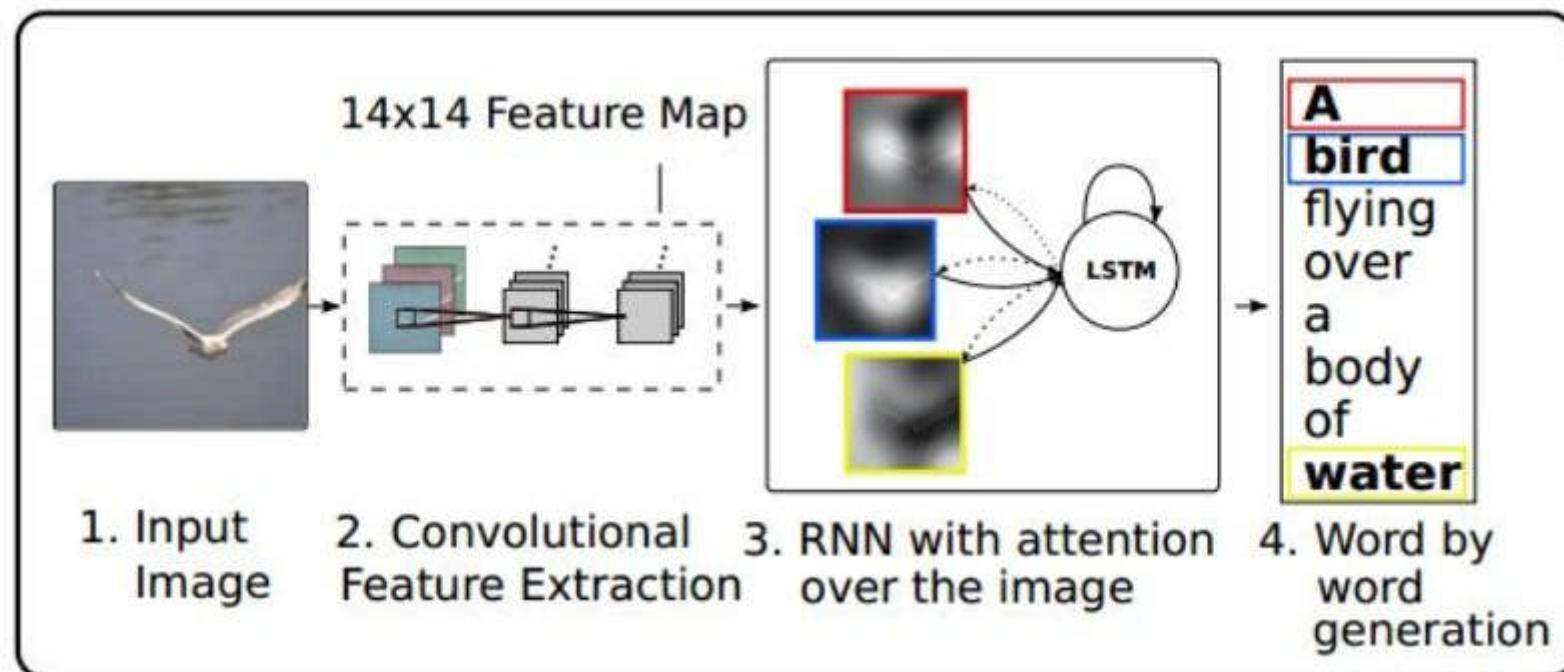


"a horse is standing in the middle of a road."

Preview of Fancier Architectures

RNN attends spatially to different parts of images while generating each word of the sentence:

Soft attention model



Show Attend and Tell: Neural Image Caption Generation with Visual Attention, Xu et al., 2015

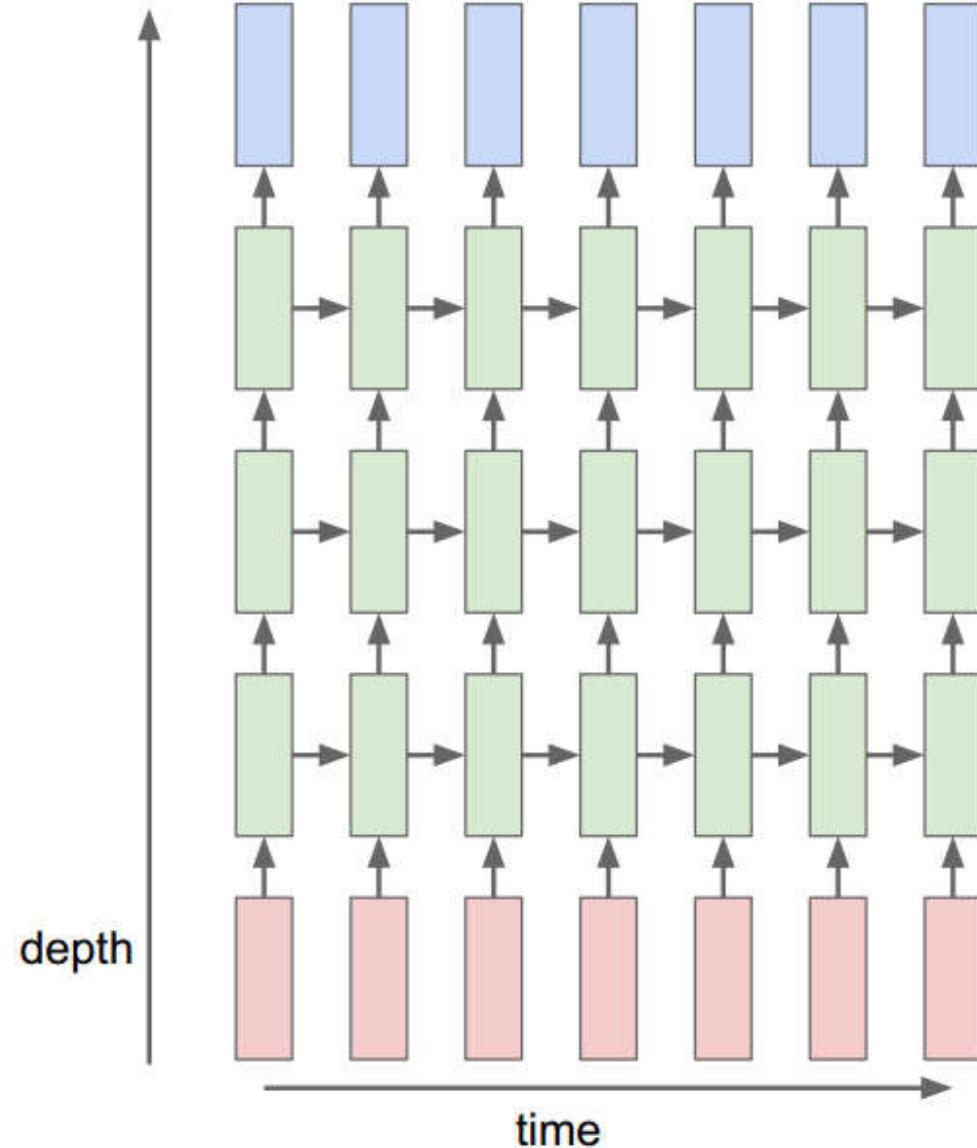
LSTM

Vanilla RNN

RNN:

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

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



Vanilla RNN

RNN:

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

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

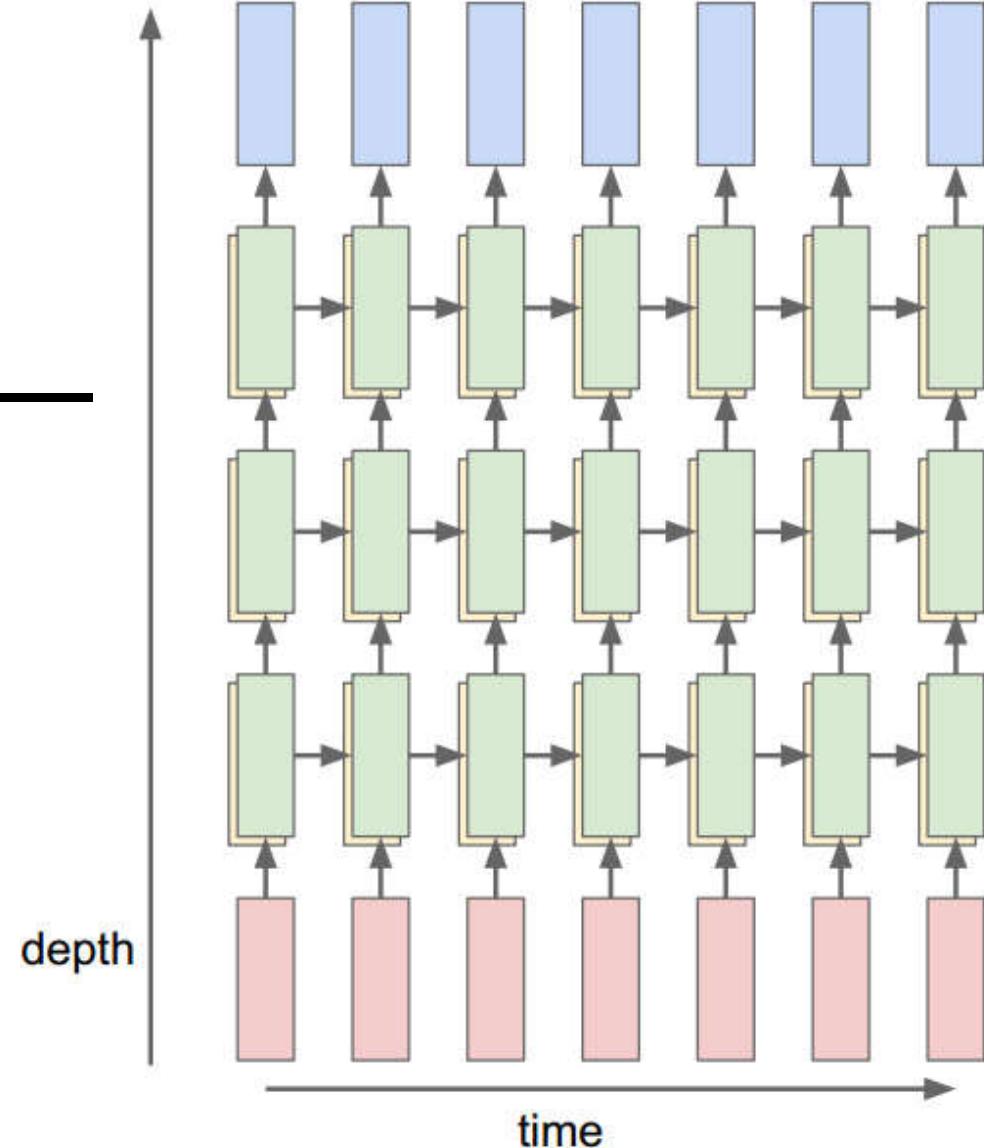
LSTM:

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

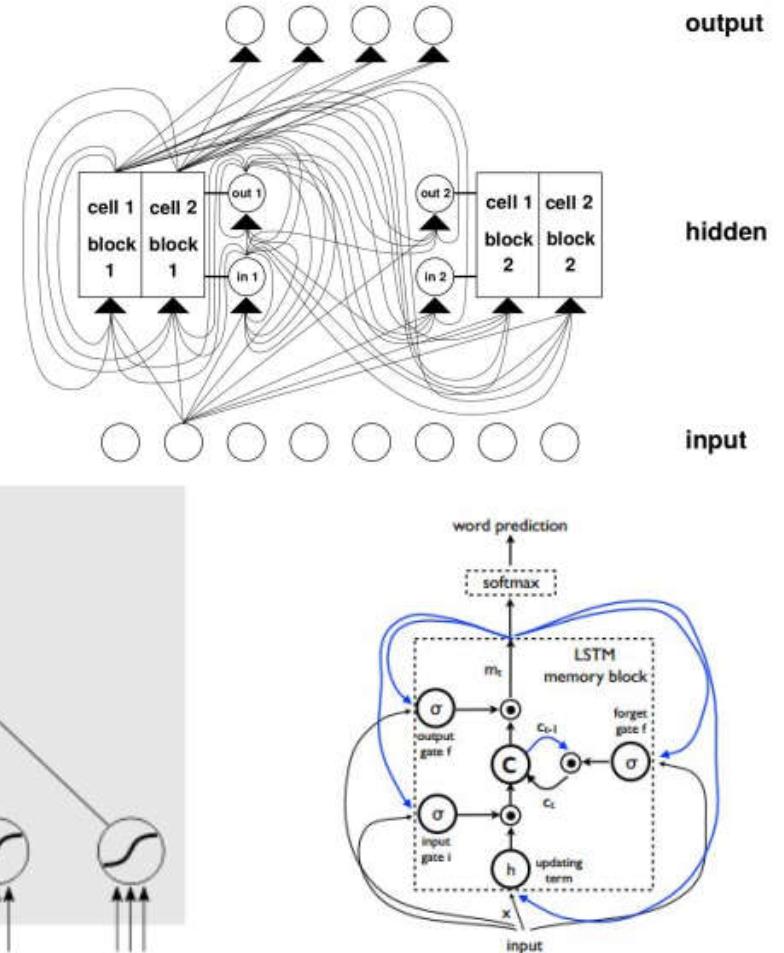
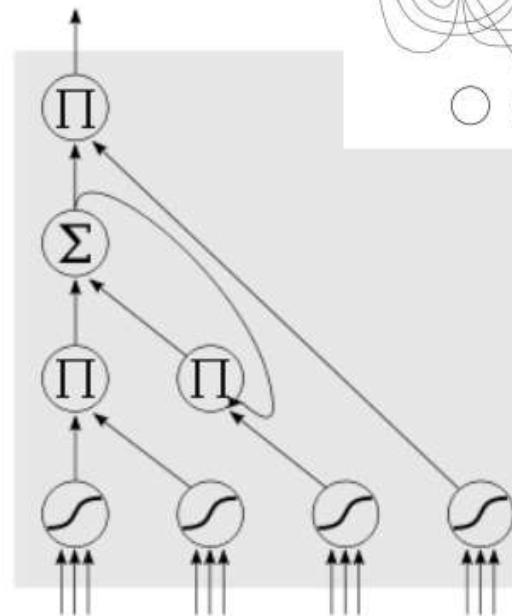
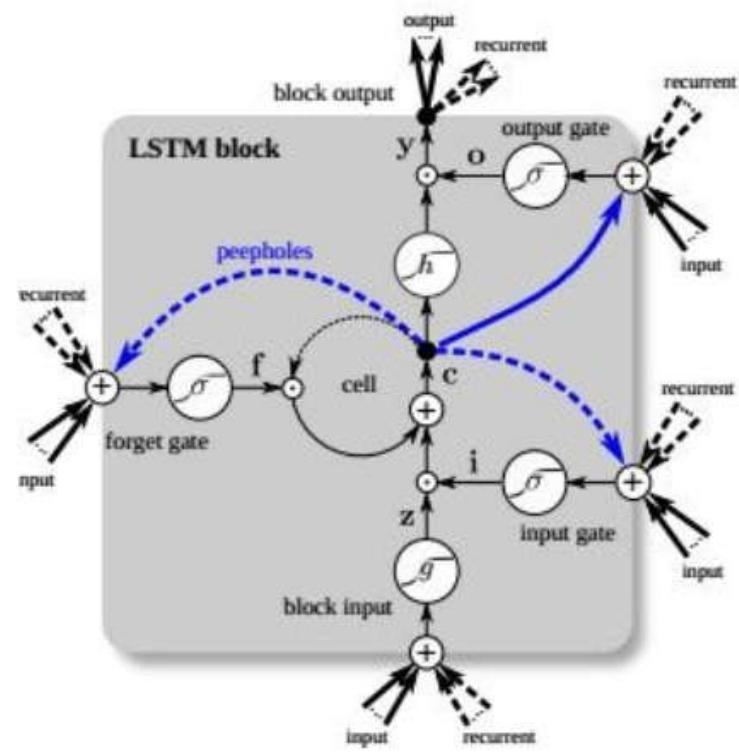
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

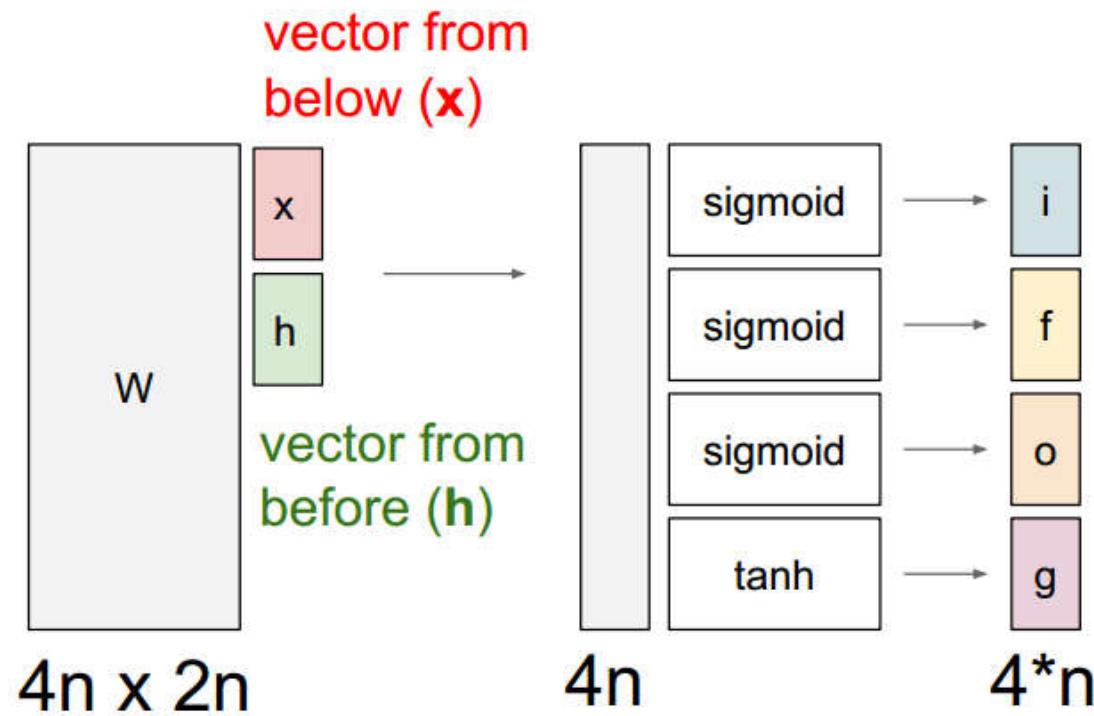


Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



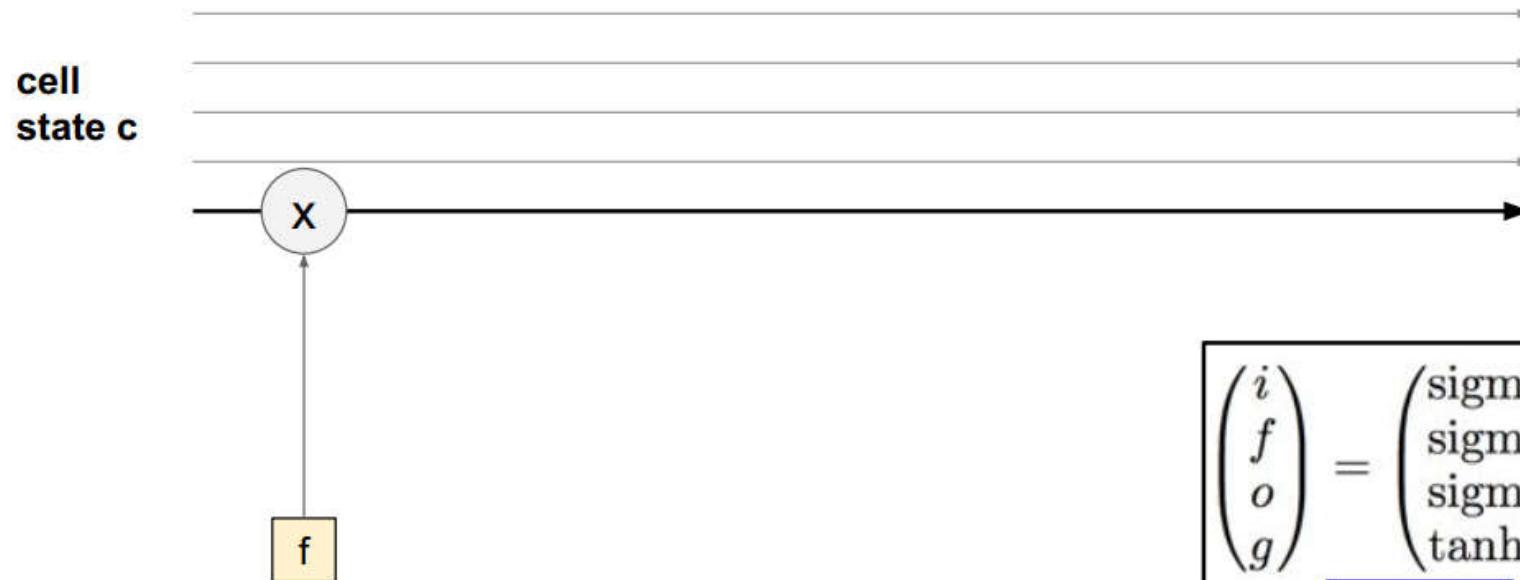
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



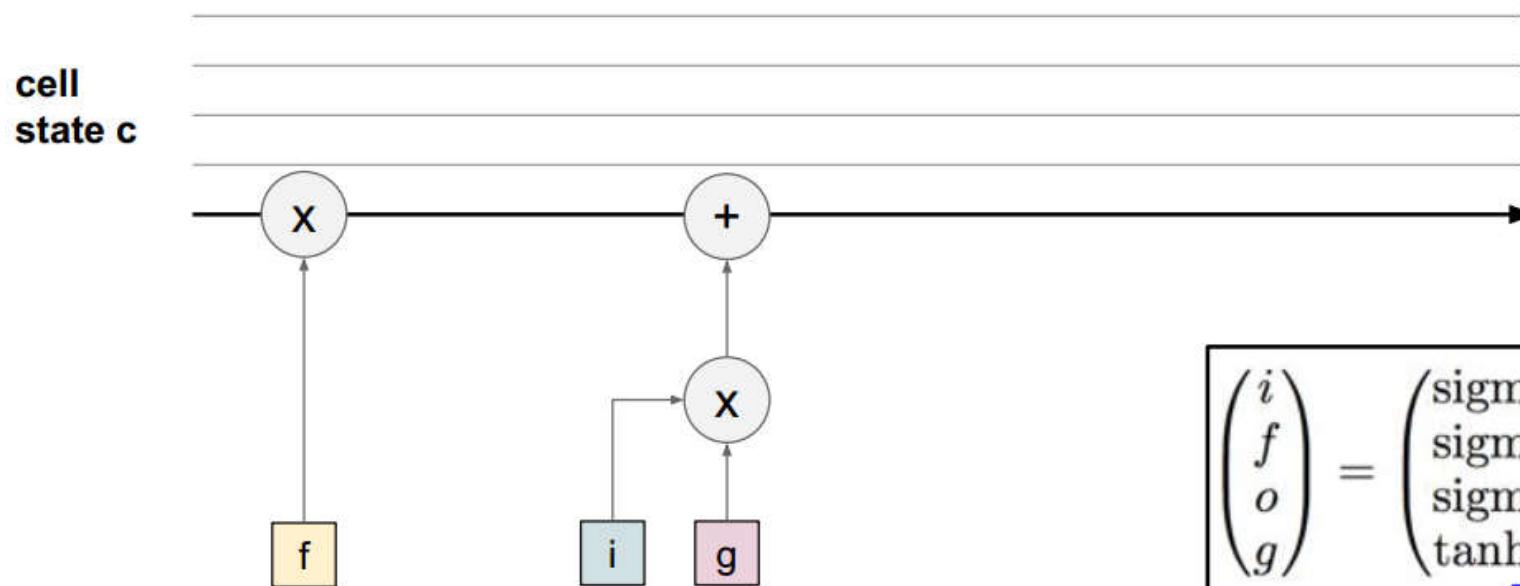
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



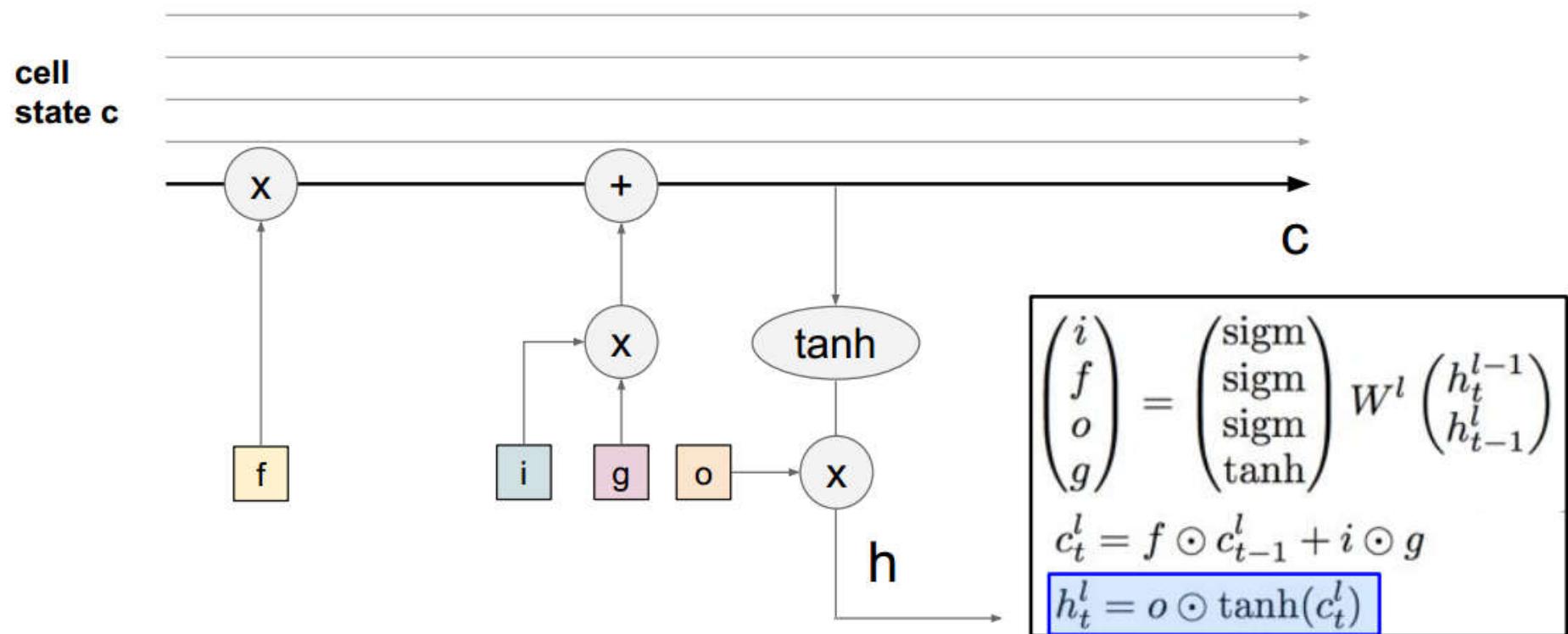
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

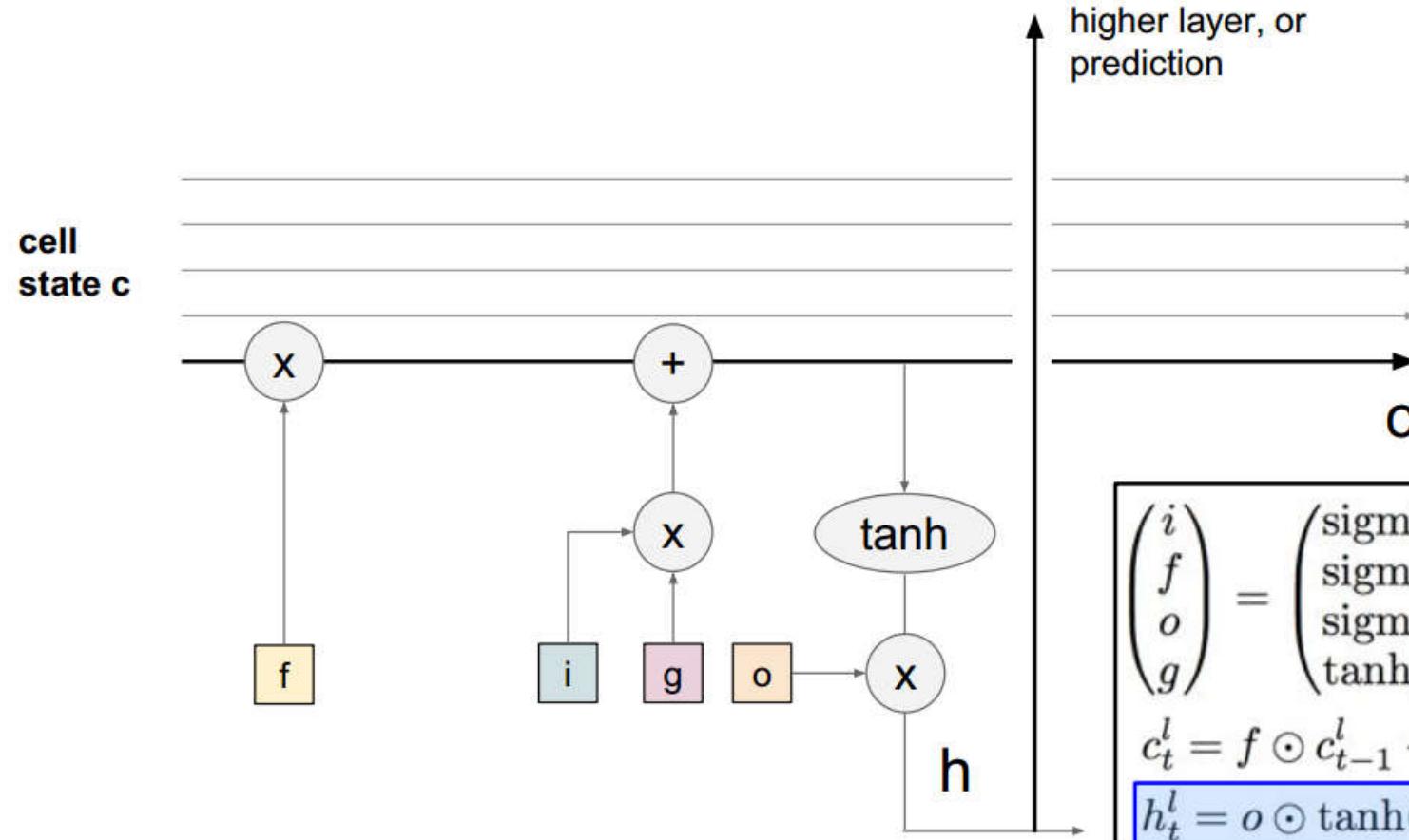
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



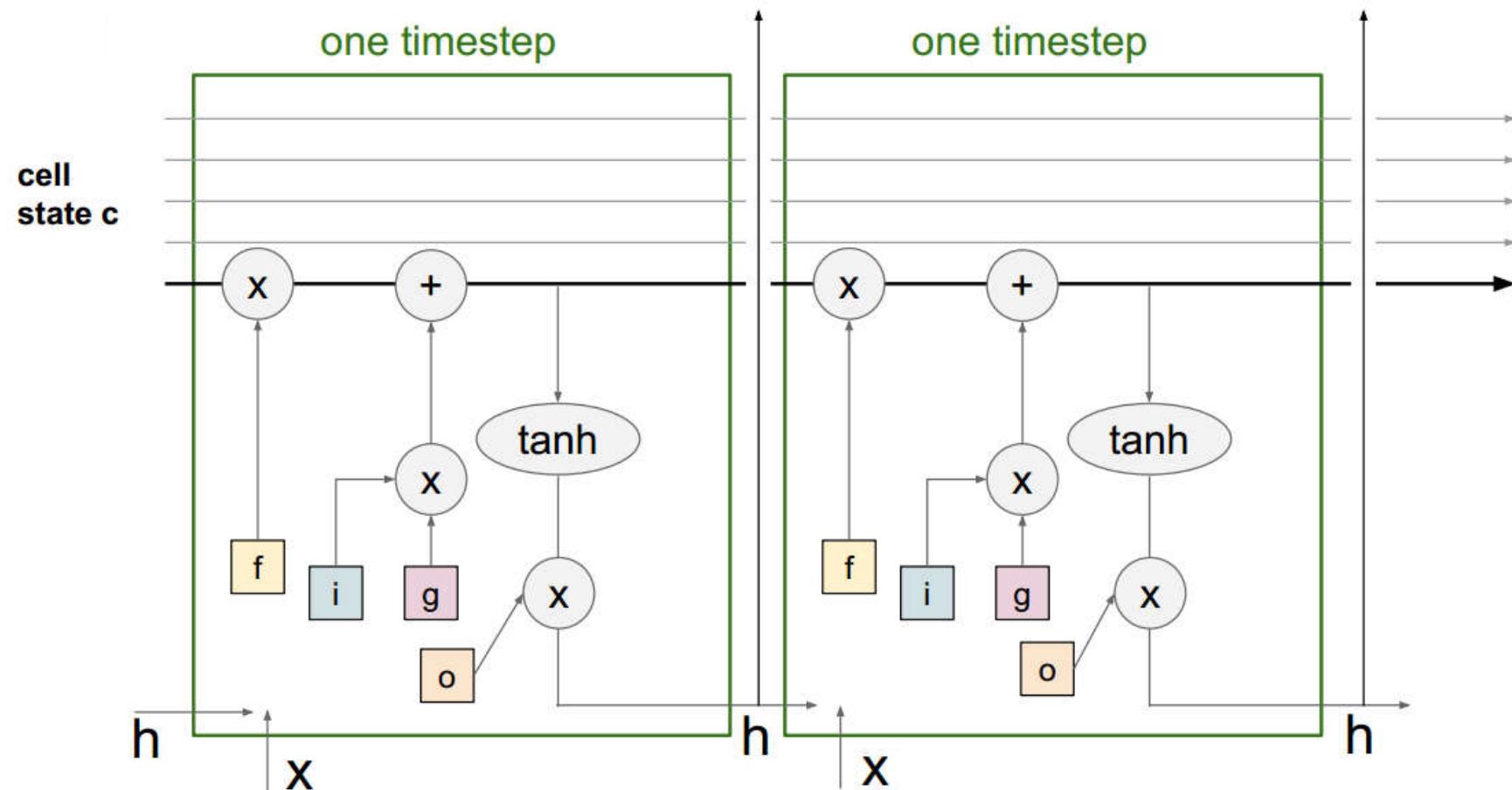
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

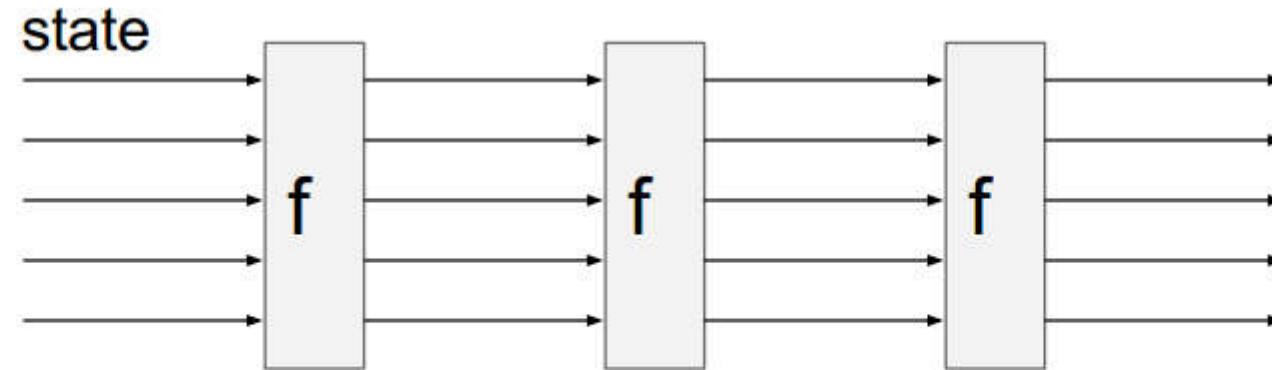
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

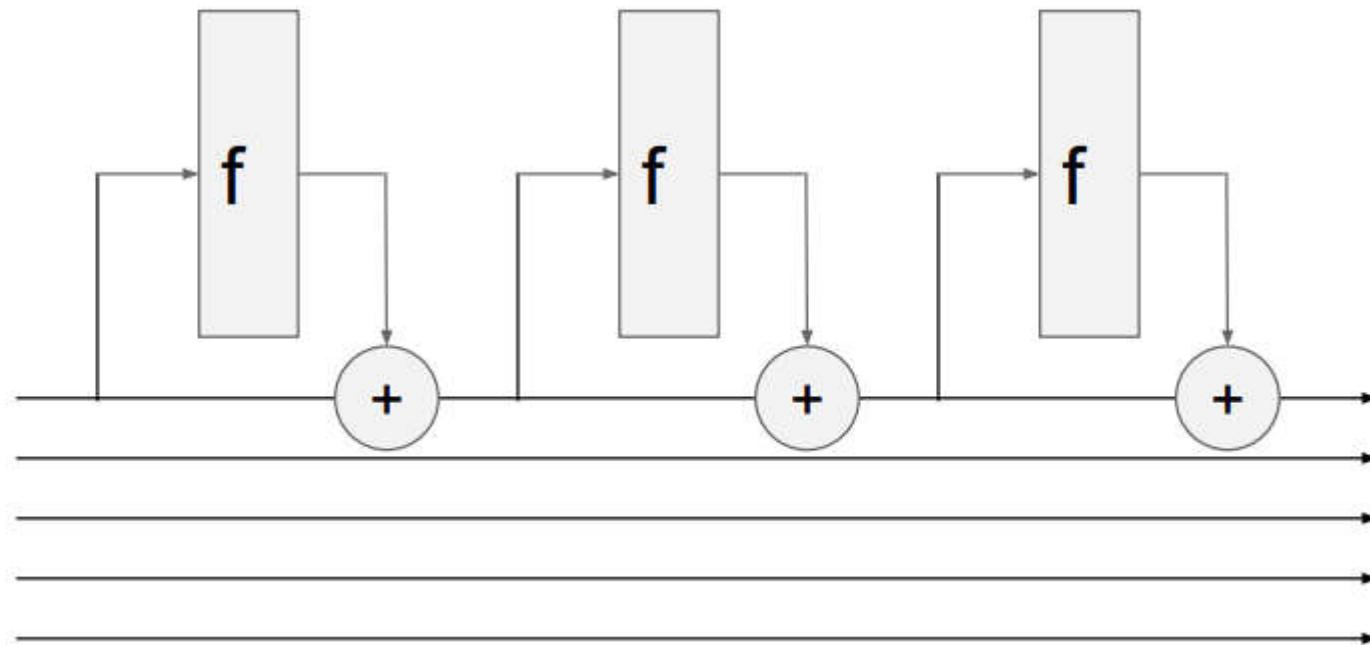


RNN vs. LSTM

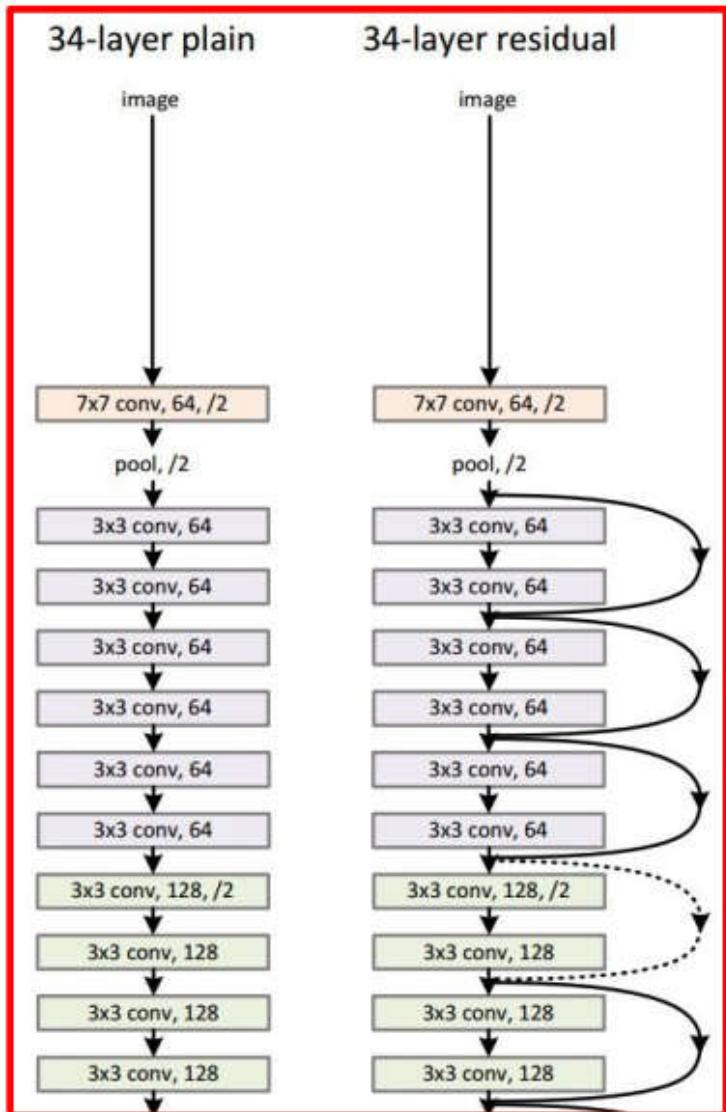
RNN



LSTM
(ignoring
forget gates)



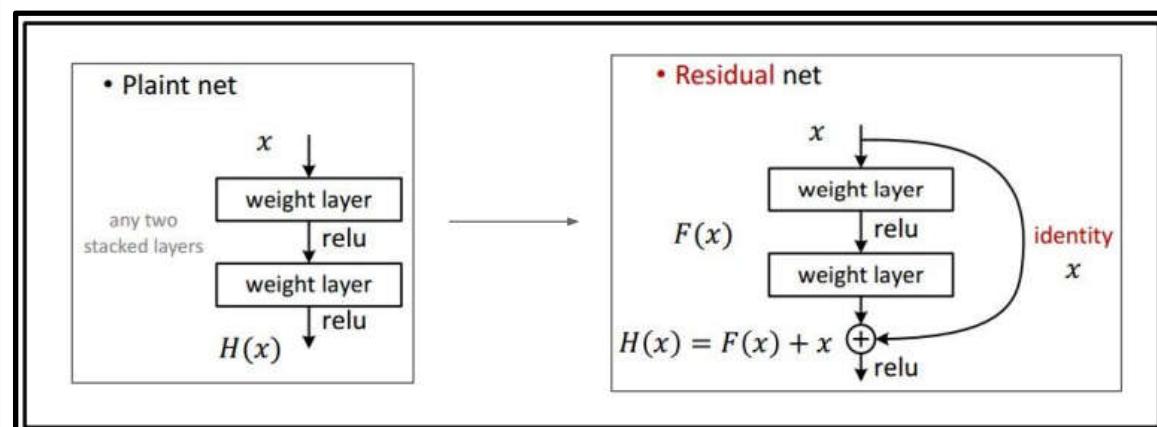
RNN vs. LSTM



Recall:

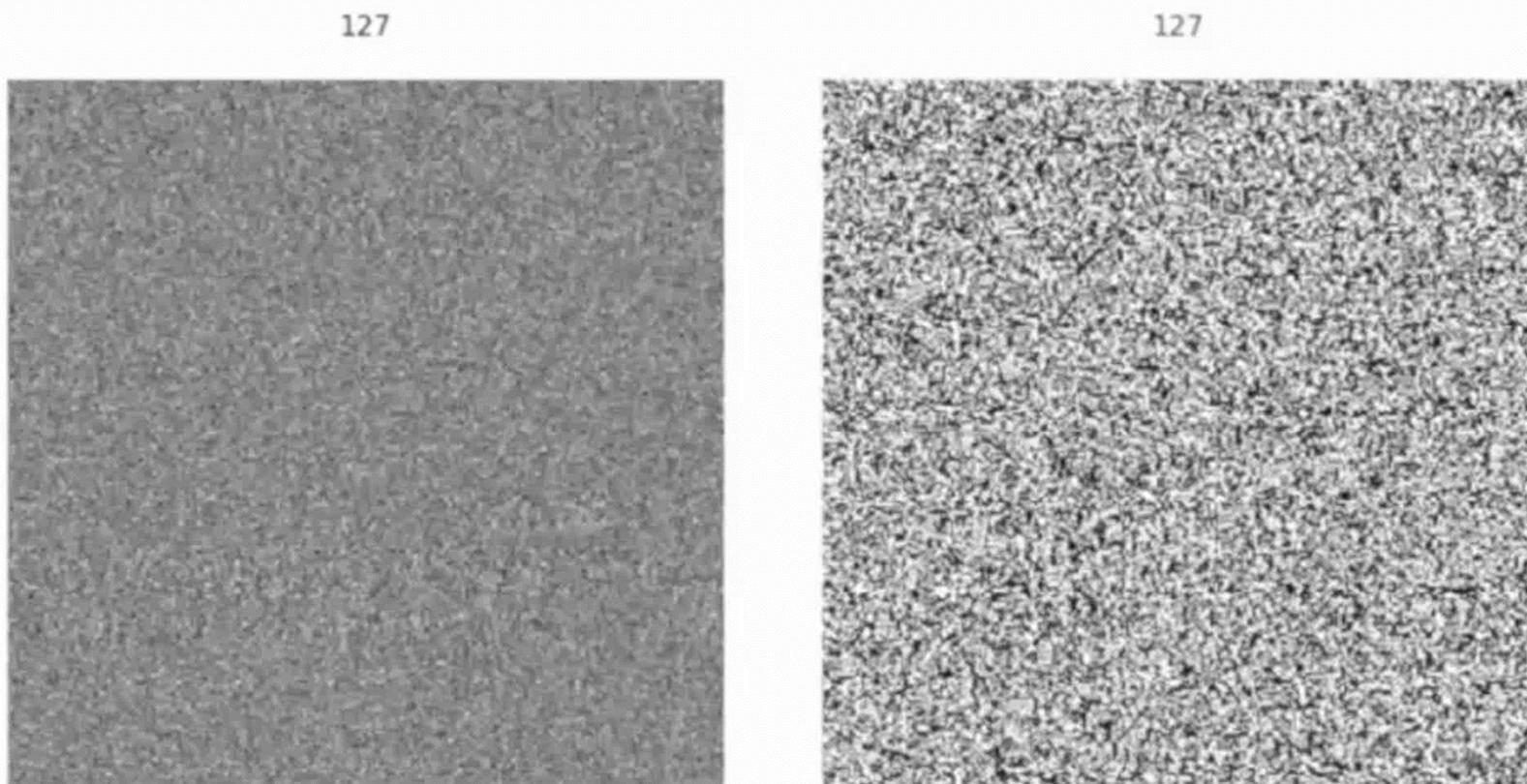
“PlainNets” vs. ResNets

ResNet is to PlainNet what LSTM is to RNN,
kind of.



Understanding Gradient Flow Dynamics

Cute backprop signal video: <http://imgur.com/gallery/vaNahKE>



Understanding Gradient Flow Dynamics



```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

- There are two possible ways

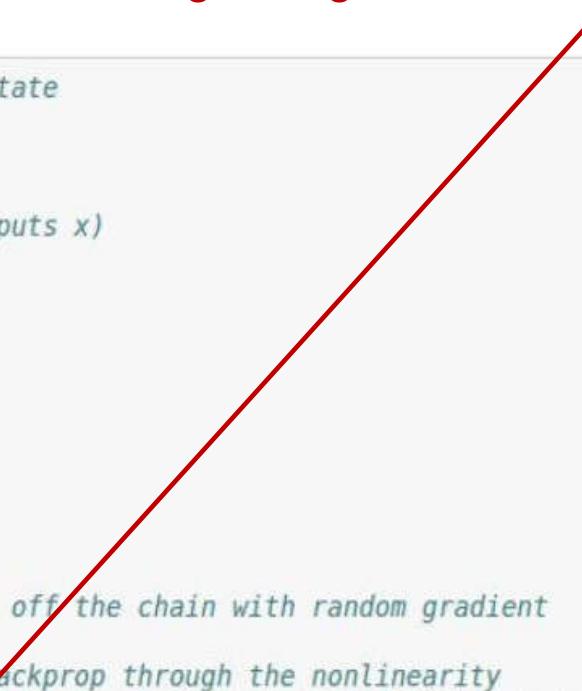
Understanding Gradient Flow Dynamics

if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish

```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```



- Vanilla RNN is very unstable and it just dies or explodes

Understanding Gradient Flow Dynamics

if the largest eigenvalue is > 1 , gradient will explode
if the largest eigenvalue is < 1 , gradient will vanish

```
H = 5      # dimensionality of hidden state
T = 50     # number of time steps
Whh = np.random.randn(H,H)

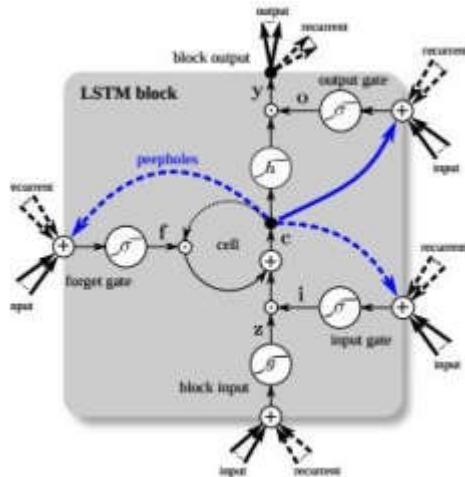
# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

can control exploding with gradient clipping
can control vanishing with LSTM

- Vanilla RNN is very unstable and it just dies or explodes
- **We always use LSTM and we do gradient clipping usually**

LSTM Variants and Friends



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

GRU (Gated Recurrent Unit)

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

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

It's a shorter smaller formula and it only has a single H vector

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

MUT1:

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

MUT2:

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

MUT3:

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

They didn't find anything that works substantially better than basic LSTM

Summary

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

Thank you for your attention!
