Lecture 23: Recurrent Neural Networks Again

COMP 411, Fall 2021
Victoria Manfredi

WESLEYAN
UNIVERSITY

These slides are based on figures and info from Andrej Karpathy's blog, slides created by Justin Johnson (U of Michigan) and Geoffrey Hinton (U of Toronto), and the book Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

Today's Topics

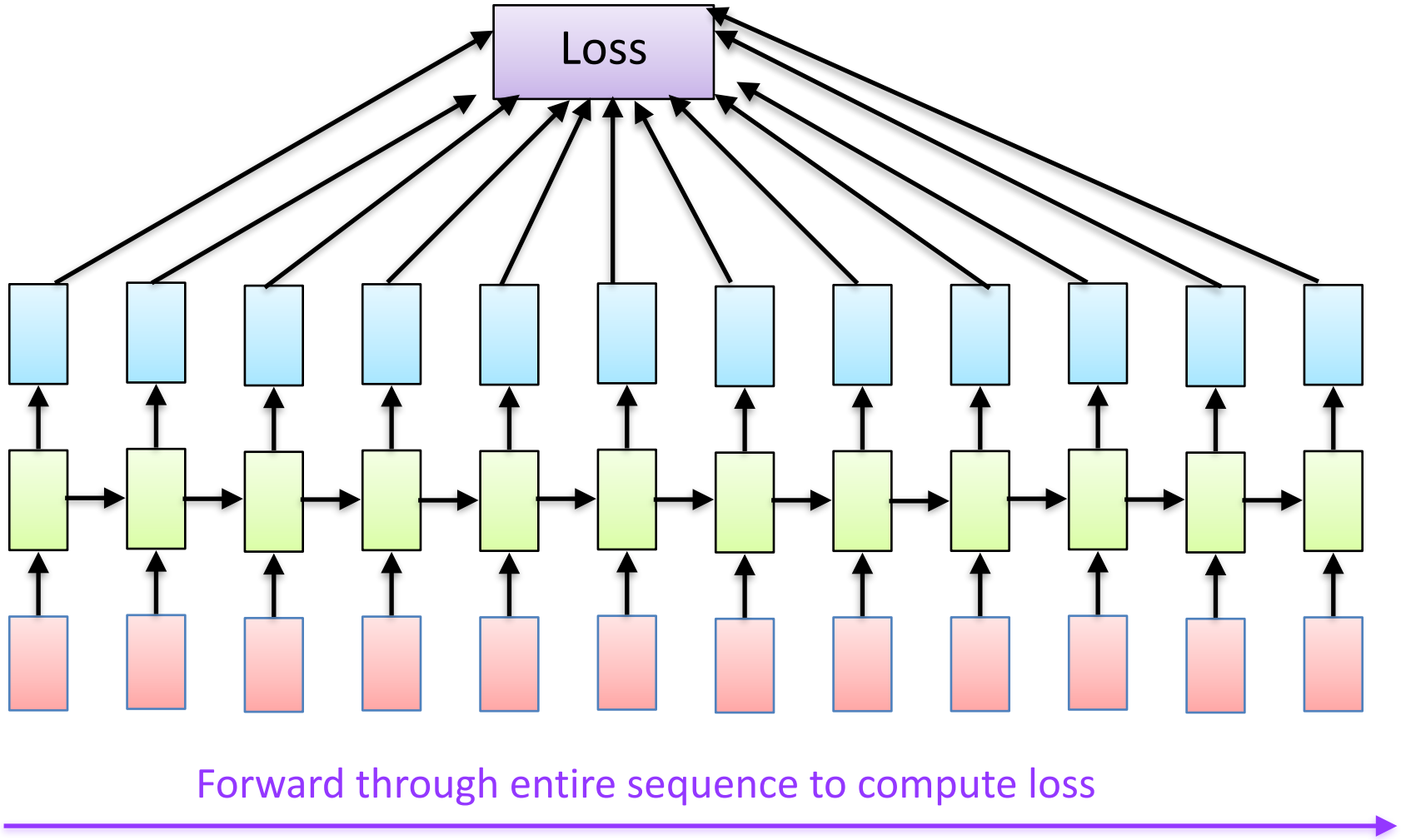
Recurrent Neural Networks

- Backpropagation through time
- Vanilla RNN gradient flow
- Issues
- Some examples

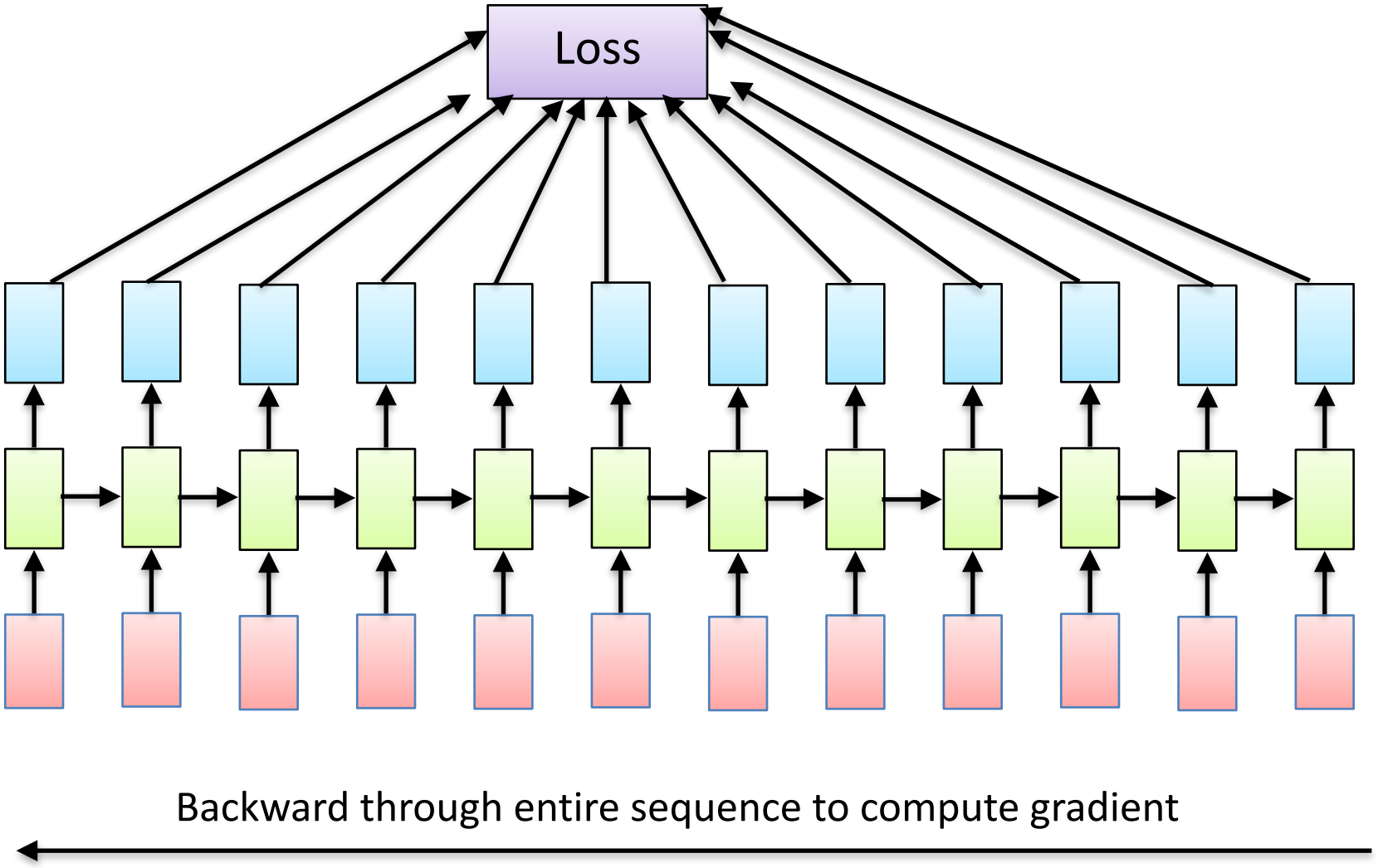
Recurrent Neural Networks

BACKPROPAGATION THROUGH TIME

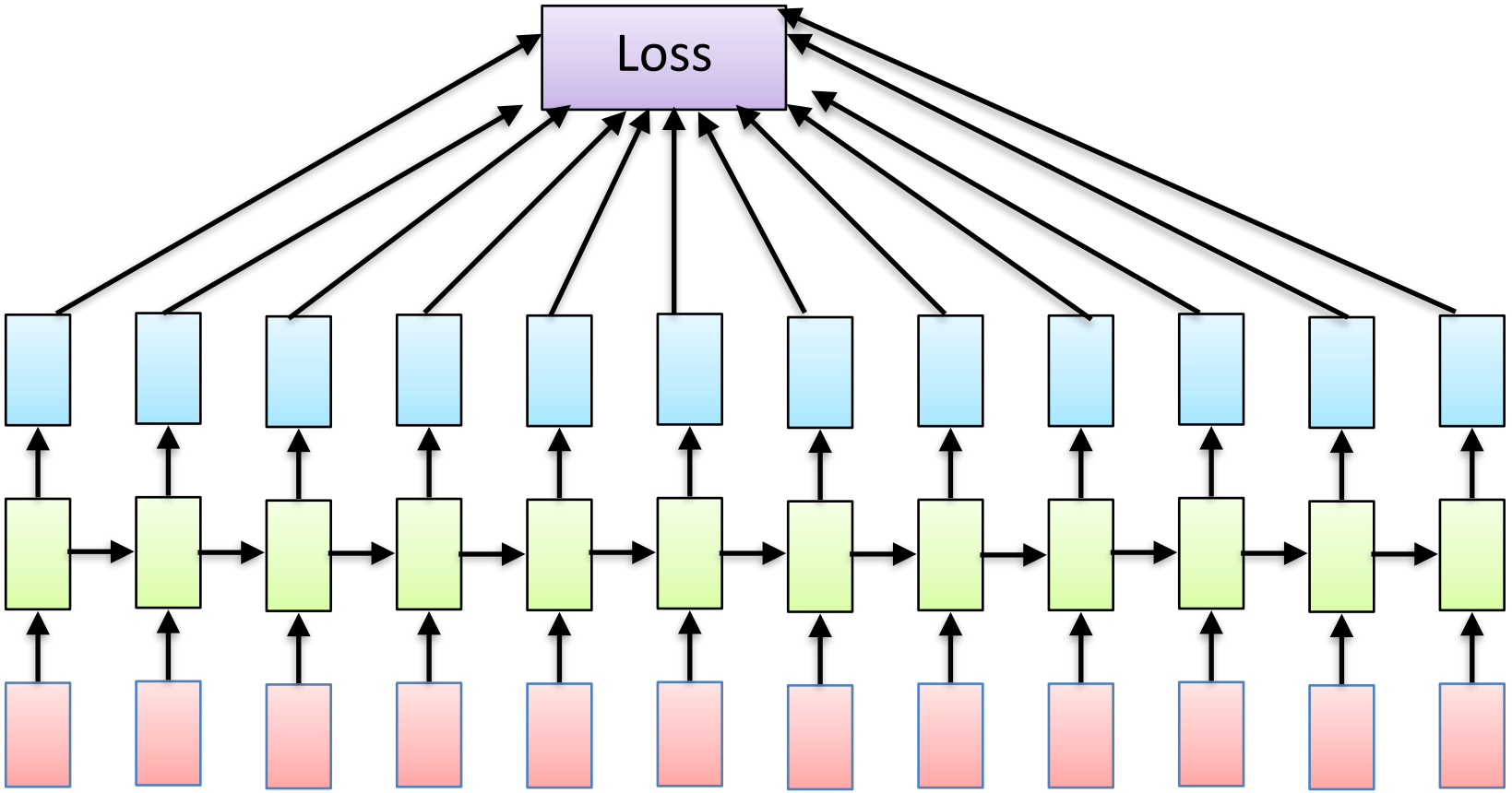
Backpropagation through time



Backpropagation through time

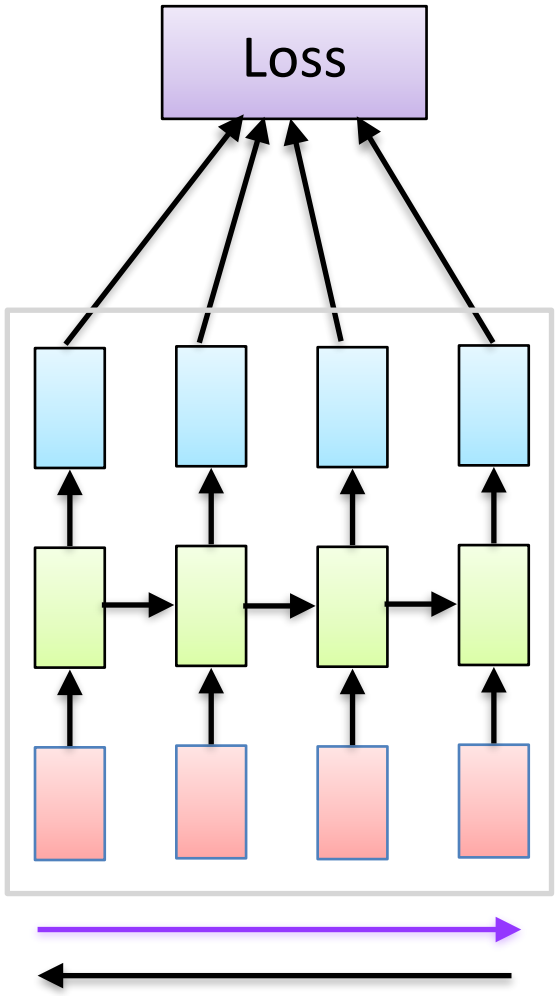


Backpropagation through time



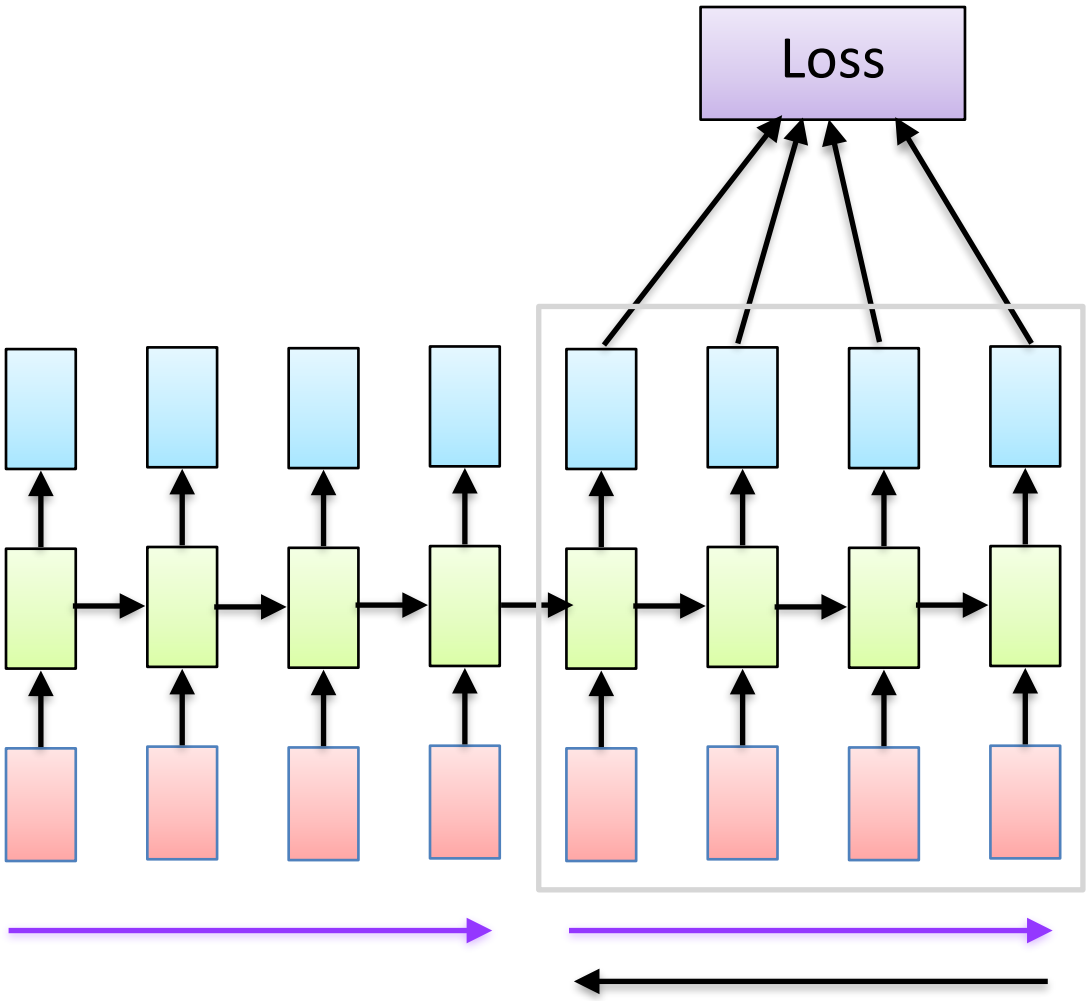
Problem: takes a lot of memory for long sequences

Truncated backpropagation through time



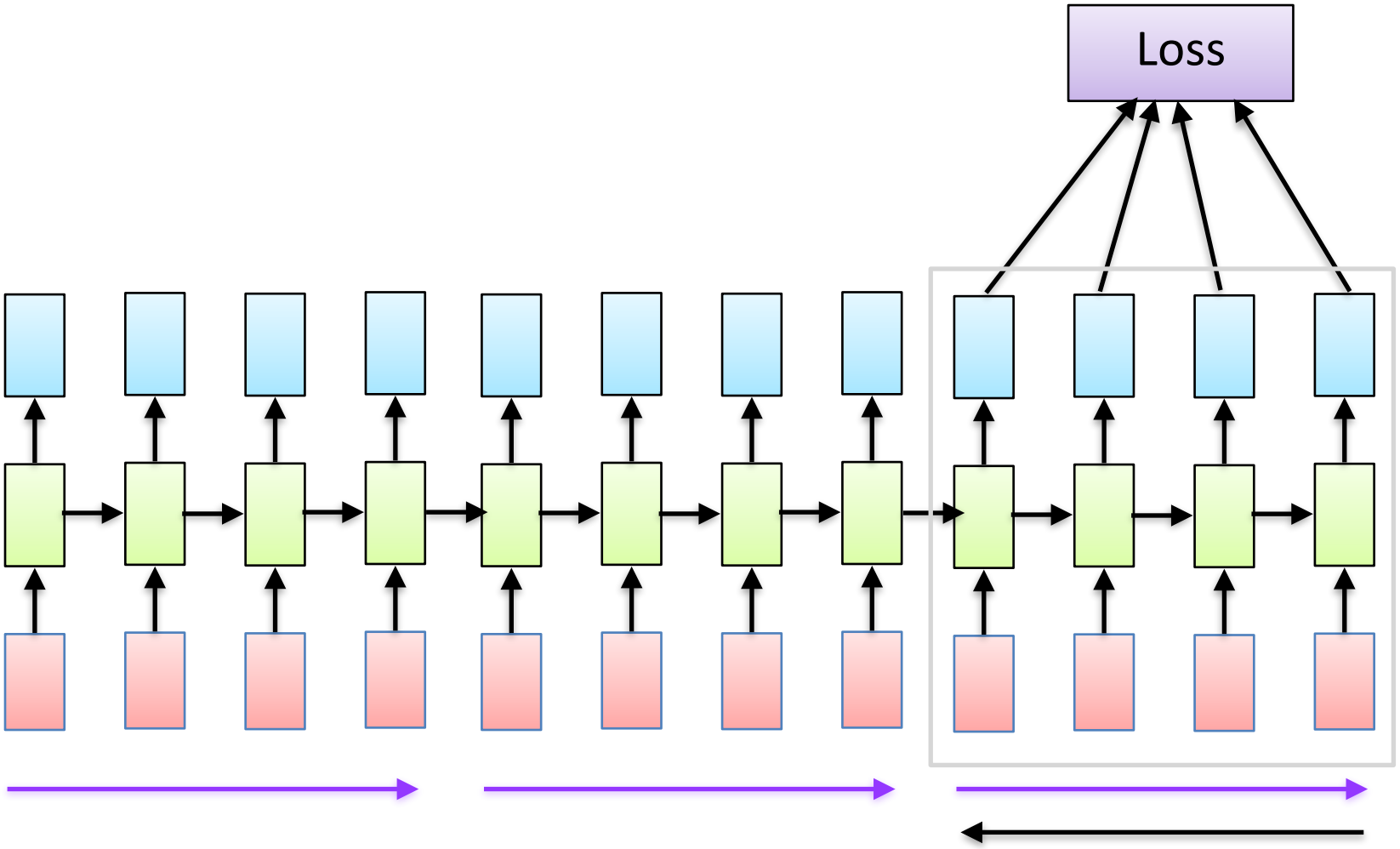
Run forward and backward through chunks of the sequence instead of the whole sequence

Truncated backpropagation through time



Carry hidden states forward in time but only backpropagate for some smaller number of steps

Truncated backpropagation through time



Recurrent Neural Networks

VANILLA RNN GRADIENT FLOW

Backpropagation with weight constraints

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$To\ \ constrain: \quad w_1 = w_2$$

$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

Backpropagation through time

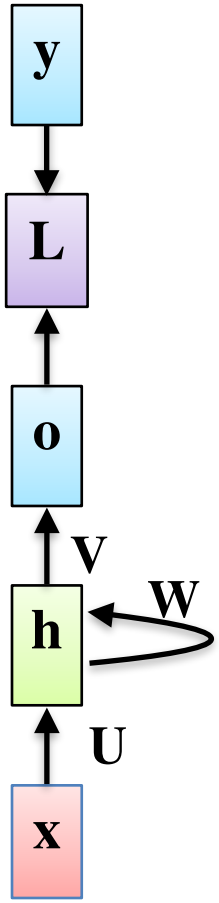
- We can think of an RNN as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.

- We can also think of this training algorithm in the time domain:
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight.

An irritating extra issue

- We need to specify the initial activity state of all the hidden and output units.
- We could just fix these initial states to have some default value like 0.5.
- But it is better to treat the initial states as learned parameters.
- We learn them in the same way as we learn the weights.
  - Start off with an initial random guess for the initial states.
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
  - Adjust the initial states by following the negative gradient.

Recurrent neural network



Map input sequence of $\mathbf{x}$ values to corresponding sequence of output $\mathbf{o}$ values.

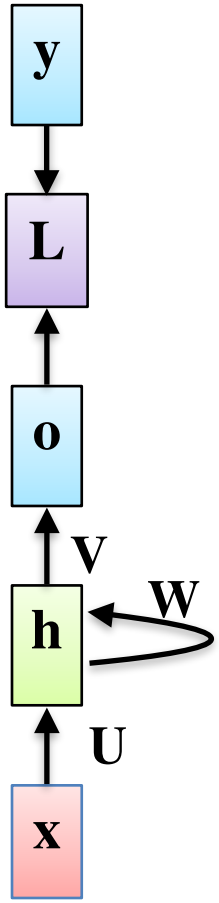
Loss $\mathbf{L}$ measures how far each $\mathbf{o}$ is from the corresponding training target $\mathbf{y}$

Input to hidden connections parametrized by weight matrix $\mathbf{U}$

Hidden-to-hidden recurrent connections parametrized by weight matrix $\mathbf{W}$

Hidden-to-output connections parametrized by weight matrix $\mathbf{V}$

Forward propagation

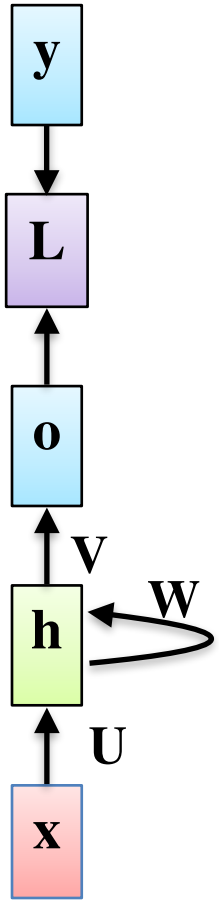


Assume hyperbolic tangent activation function for hidden units. Assume output is discrete. View output $\mathbf{o}$ as giving the unnormalized log probabilities of each possible value of the discrete variable. Apply the softmax operation as a post-processing step to obtain a vector $\hat{\mathbf{y}}$ of normalized probabilities over the output.

Forward propagation begins with a specification of the initial state $\mathbf{h}^{(0)}$. Then, for each time step from $t = 1$ to $t = \tau$, where τ is the final step, we apply the following update equations where the parameters are the bias vectors $\mathbf{b}$ and $\mathbf{c}$ along with the weight matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$
$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

Forward propagation



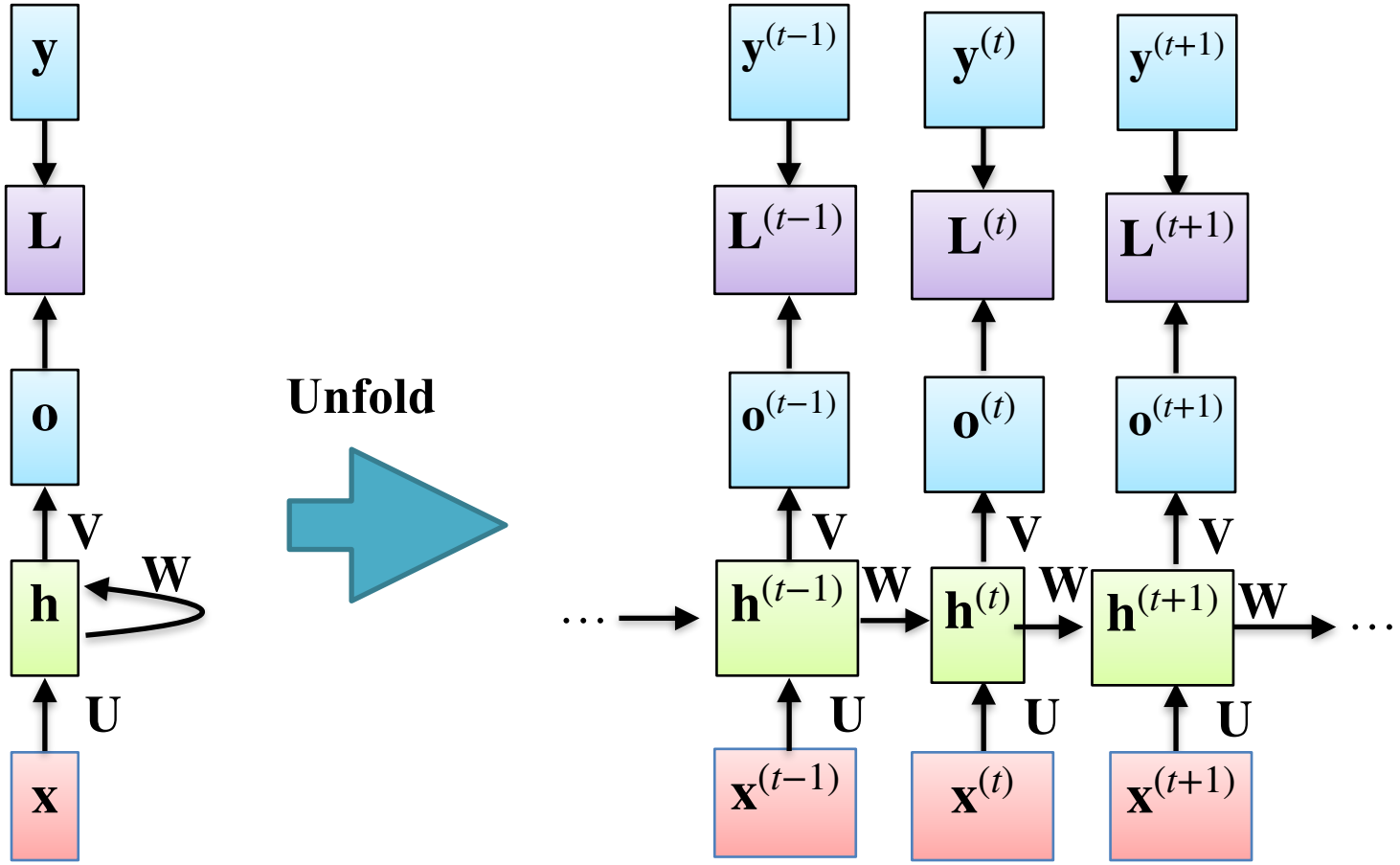
This recurrent network maps an input sequence to an output sequence of the same length. The total loss for a given sequence of $\mathbf{x}$ values paired with a sequence of $\mathbf{y}$ values is the sum of the losses over all the time steps. For example, if $L^{(t)}$ is the negative log-likelihood of $\mathbf{y}^{(t)}$ given $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$, then

$$L\left(\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\tau)}\}\right) = \sum_t L^{(t)} = -\sum_t \log p_{model}\left(y^{(t)} \mid \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}\right)$$

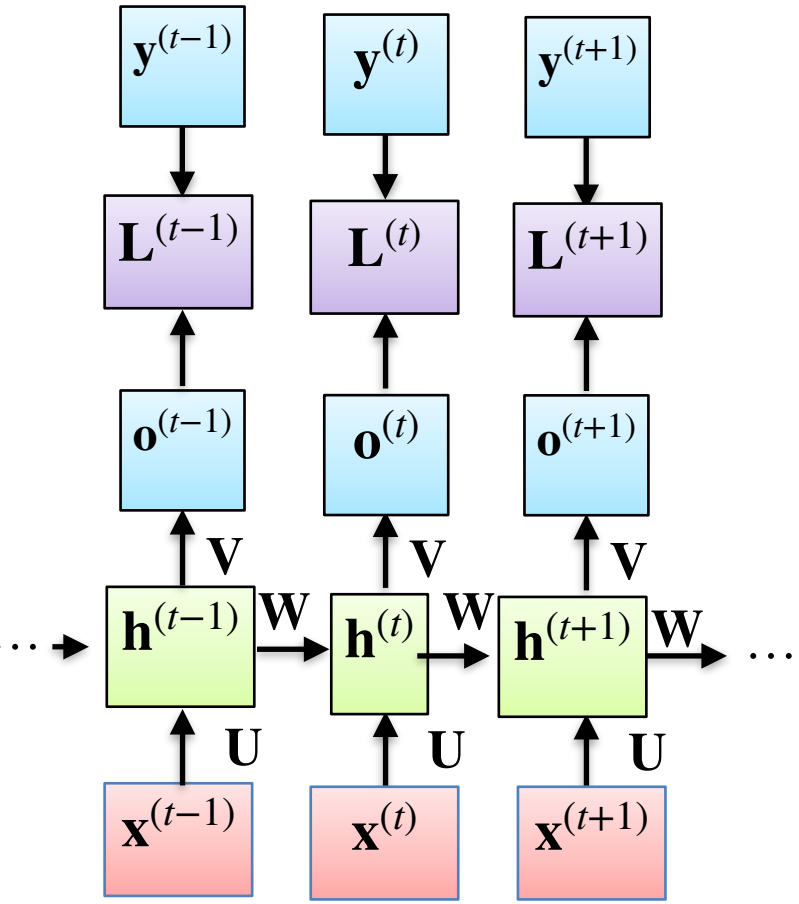
Log loss

where $p_{model}(y^{(t)} \mid \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)})$ is given by reading the entry for $y^{(t)}$ from the model's output vector $\hat{\mathbf{y}}^{(t)}$

Backward propagation through time



Backward propagation through time



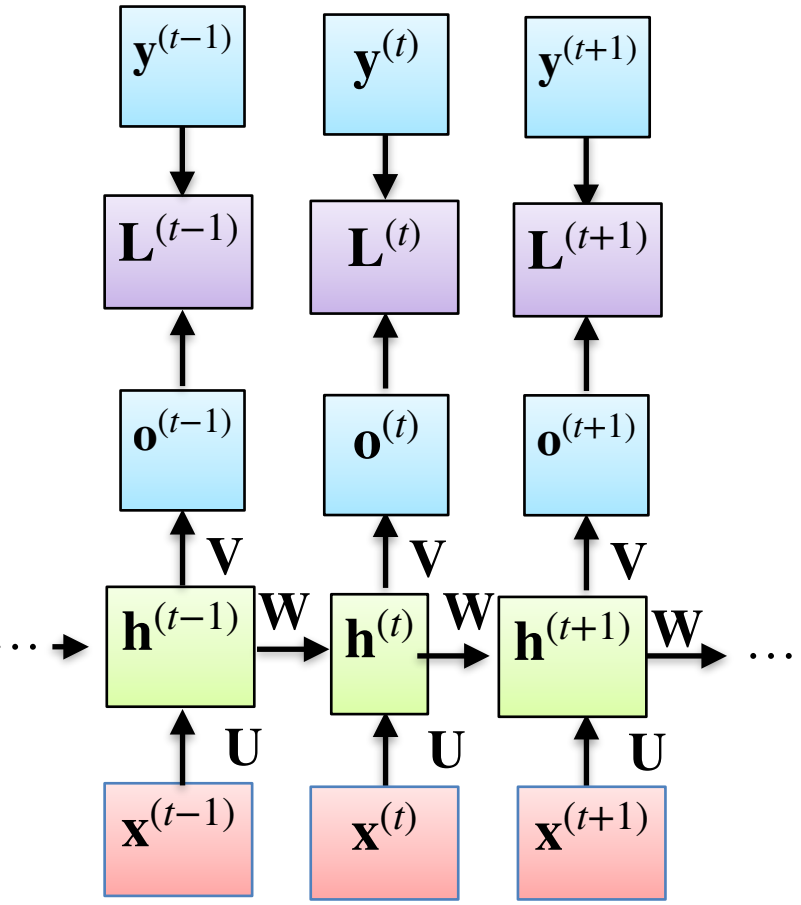
Computing the gradient of loss function L with respect to the parameters involves performing a forward propagation pass moving left to right through the unrolled graph, followed by a backward propagation pass moving right to left through the graph.

For each node N we need to compute the gradient $\nabla_N L$ recursively, based on the gradient computed at nodes that follow it in the graph. We start the recursion with the nodes immediately preceding the final loss:

$$\frac{\partial L}{\partial L^{(t)}} = 1$$

then compute $\nabla_{\mathbf{o}^{(t)}} L,\ \nabla_{\mathbf{h}^{(\tau)}} L,\ \nabla_{\mathbf{h}^{(t)}} L,\ \nabla_{\mathbf{c}} L,$ $\nabla_{\mathbf{b}} L, \nabla_{\mathbf{V}} L,\ \nabla_{\mathbf{W}} L,\ \nabla_{\mathbf{U}} L$

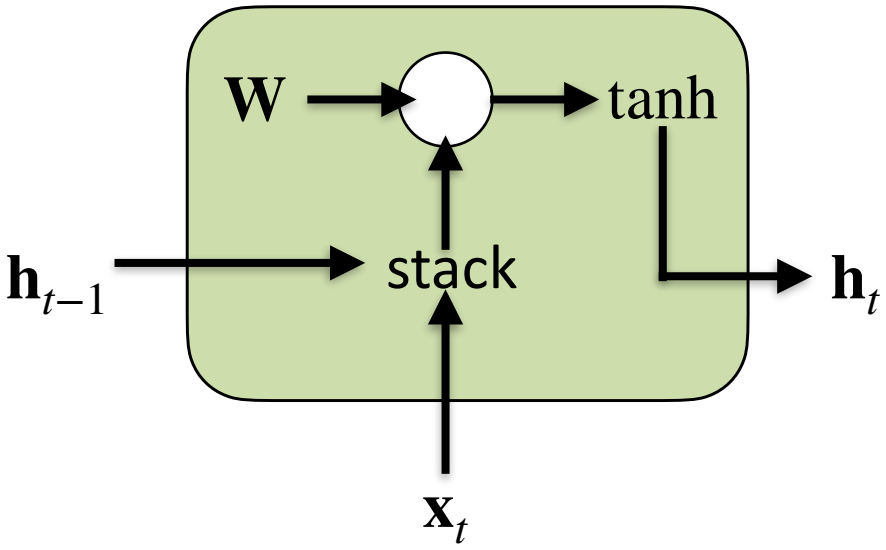
Backward propagation through time



Backpropagation through time is computationally expensive. For a single weight update, need to update based on sequence length. Multiple outputs means more of these updates taking into consideration sequence length.

Can we do better? Truncated Backpropagation Through Time

Vanilla RNN Gradient Flow



$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

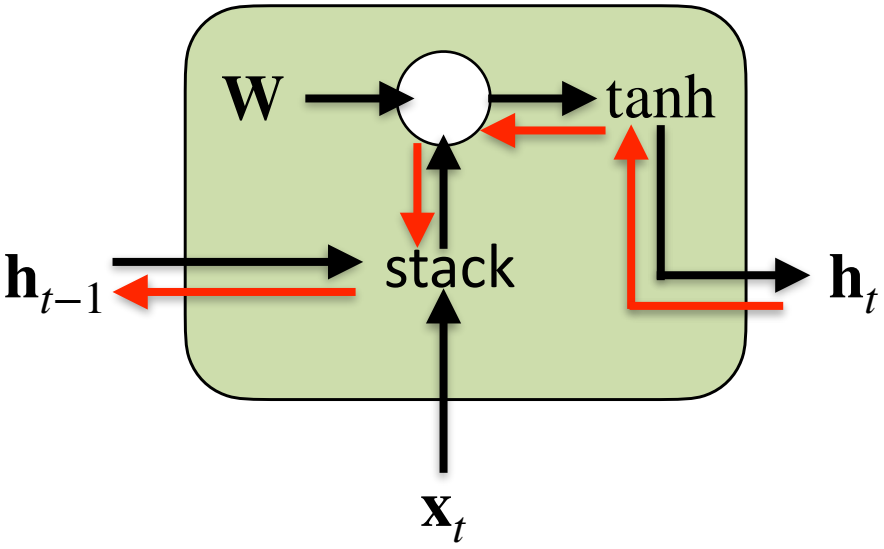
$$= \tanh\left((\mathbf{W}_{hh} \quad \mathbf{W}_{xh})\begin{pmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{pmatrix}\right)$$

$$= \tanh\left(\mathbf{W}\begin{pmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{pmatrix}\right)$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994 Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Vanilla RNN Gradient Flow

Backpropagation from $\mathbf{h}_t$ to $\mathbf{h}_{t-1}$ multiplies by $\mathbf{W}$ (actually $\mathbf{W}_{hh}^T$)



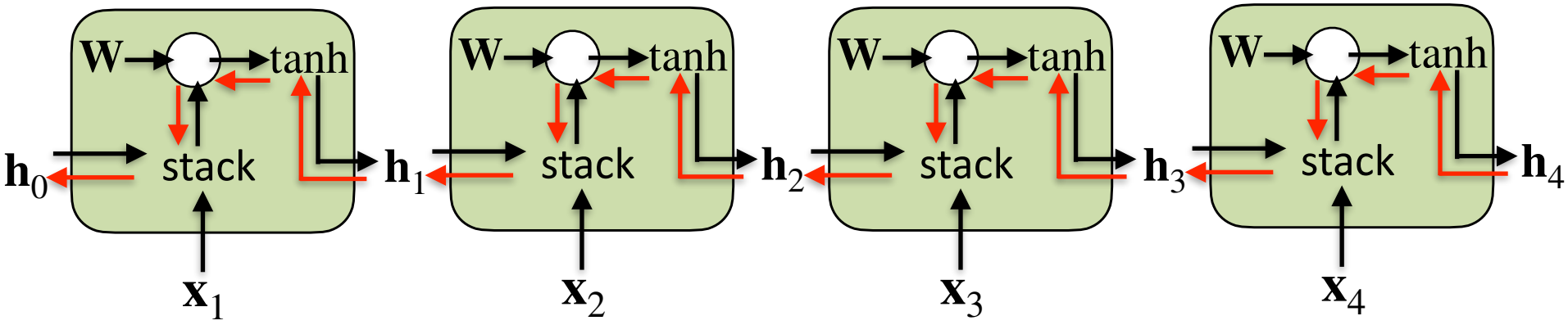
$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

$$= \tanh\left((\mathbf{W}_{hh} \quad \mathbf{W}_{xh})\begin{pmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{pmatrix}\right)$$

$$= \tanh\left(\mathbf{W}\begin{pmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{pmatrix}\right)$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994 Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Vanilla RNN Gradient Flow

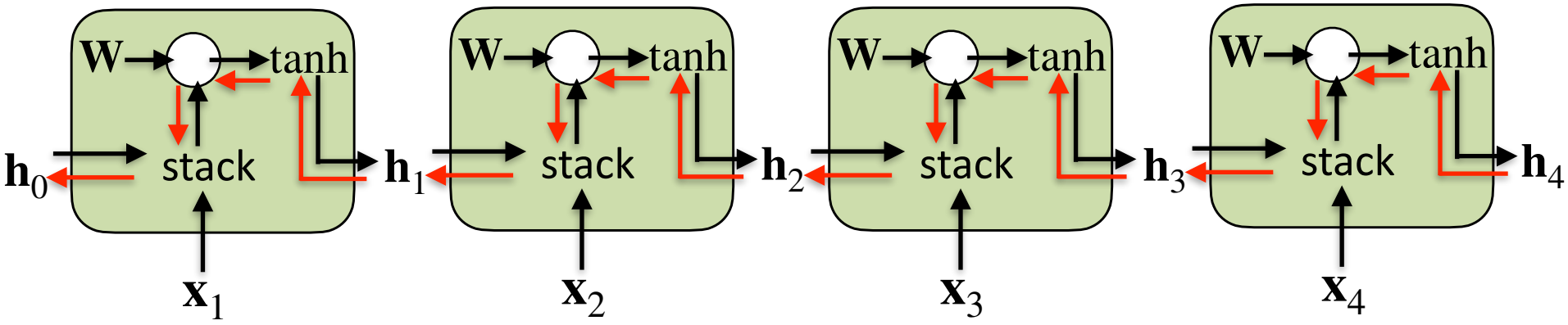


Computing gradient of $\mathbf{h}_0$ include many factors of $\mathbf{W}$ (and repeated $\tanh$)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of $\mathbf{h}_0$ include many factors of $\mathbf{W}$ (and repeated tanh)

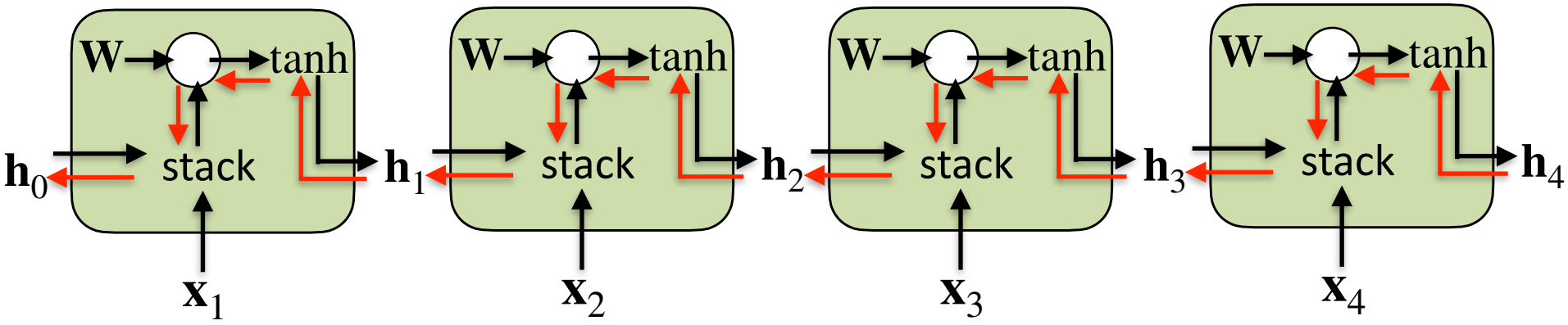
Largest singular value > 1:
Exploding gradients

→ **Gradient clipping:** scale gradient if its norm is too big

Largest singular value < 1:
Vanishing gradients

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

Vanilla RNN Gradient Flow



Computing gradient of $\mathbf{h}_0$ include many factors of $\mathbf{W}$ (and repeated tanh)

Largest singular value > 1:
Exploding gradients

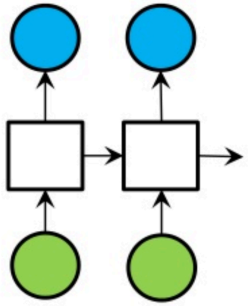
Largest singular value < 1:
Vanishing gradients → **Change RNN architecture!**

Recurrent Neural Networks

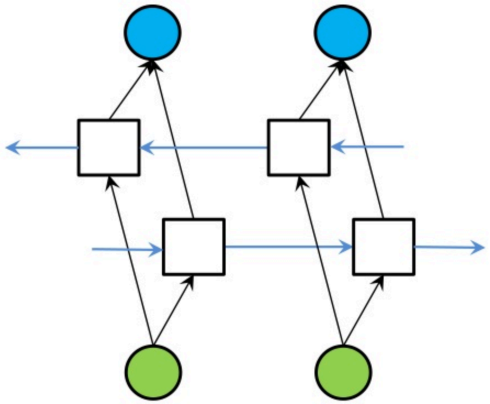
ISSUES

Bi-directional RNN

- One of the issues with RNN:
  - Hidden variables capture only one side context
- A bi-directional structure



RNN



Bi-directional RNN

For example mask out a word in sentence and then try to predict that word. Having full context of sentence is helpful: e.g., later words may give context to earlier words.

The problem of exploding or vanishing gradients

What happens to the magnitude of the gradients as we backpropagate through many layers?

- If the weights are small, the gradients shrink exponentially
- If the weights are big the gradients grow exponentially

Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers

In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.

- We can avoid this by initializing the weights very carefully.

Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.

- So RNNs have difficulty dealing with long-range dependencies.

The problem of exploding or vanishing gradients

Recurrent networks use the same matrix $\mathbf{W}$ at each timestep but feedforward networks do not, so even very deep feedforward networks can largely avoid the vanishing and exploding gradient problem

The problem of exploding or vanishing gradients

The function composition employed by RNNs somewhat resembles matrix multiplication. We can think of the recurrence relation

$$\mathbf{h}^{(t)} = \mathbf{W}^T \mathbf{h}^{(t-1)}$$

as a very simple RNN lacking a nonlinear activation function and lacking inputs $\mathbf{x}$. It may be simplified to

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^T \mathbf{h}^{(0)}$$

and if $\mathbf{W}$ admits an eigendecomposition of the form $\mathbf{W} = \mathbf{Q}\Lambda\mathbf{Q}^{\mathbf{T}}$ with orthogonal $\mathbf{Q}$, the recurrence may be simplified further to

$$\mathbf{h}^{(t)} = \mathbf{Q}^T \Lambda^t \mathbf{Q} \mathbf{h}^{(0)}$$

The eigenvalues are raised to the power of t causing eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than zero to explode. Any component of $\mathbf{h}^{(0)}$ that is not aligned with the largest eigenvalue will eventually be discarded

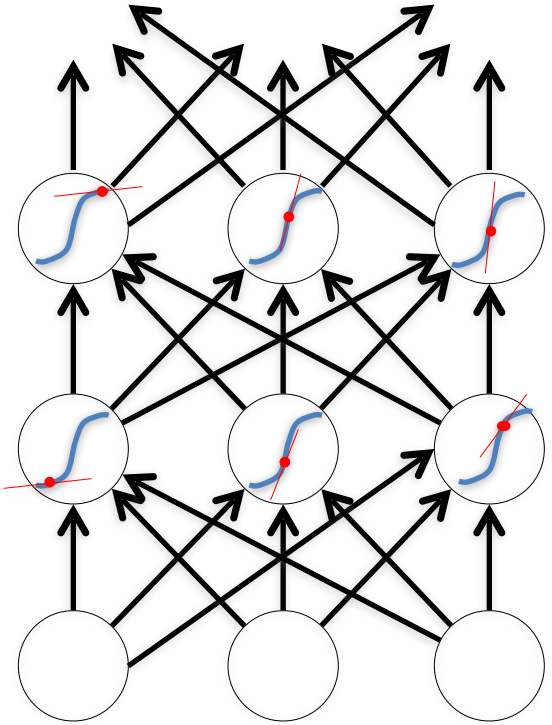
The backward pass is linear

There is a big difference between the forward and backward passes.

In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding.

The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.

- The forward pass determines the slope of the linear function used for backpropagating through each neuron.



Can we show why this is true mathematically?

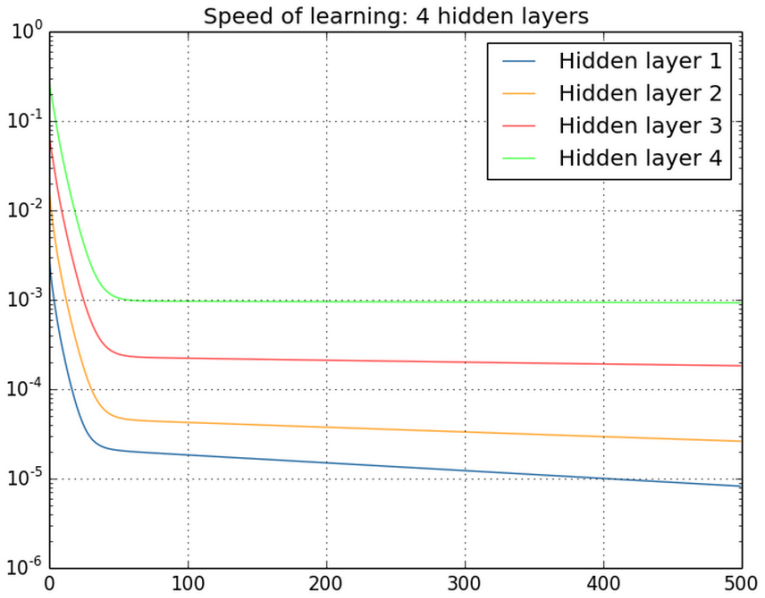
Vanishing/exploding gradients

Vanishing gradients are quite prevalent and a serious issue.

Gradient can become very small or very large quickly, and the locality assumption of gradient descent breaks down (Vanishing gradient) [Bengio et al 1994]

A real example

- Training a feed-forward network
- y-axis: sum of the gradient norms
- Earlier layers have exponentially smaller sum of gradient norms
- This will make training earlier layers much slower.



Vanishing/exploding gradients

In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.

- So RNNs have difficulty dealing with long-range dependencies.

Many methods proposed for reduce the effect of vanishing gradients; although it is still a problem

- Introduce shorter path between long connections
- Abandon stochastic gradient descent in favor of a much more sophisticated Hessian-Free (HF) optimization
- Add fancier modules that are robust to handling long memory; e.g. Long Short Term Memory (LSTM)

One trick to handle the exploding-gradients:

- Clip gradients with bigger sizes:

Name for doing this simplification with a vector: g /||g|| would just be unit vector, so here we would get just a threshold vector below

$$\textbf{Define } g = \frac{\partial E}{\partial W}. \textbf{ If } \|g\| \geq threshold \textbf{ then } g \leftarrow \frac{threshold}{\|g\|} g$$

||g|| is dot product /magnitude

Recurrent Neural Networks

SOME EXAMPLES

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

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

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

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

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

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    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(xrange(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
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  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]
```

```python
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

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
```

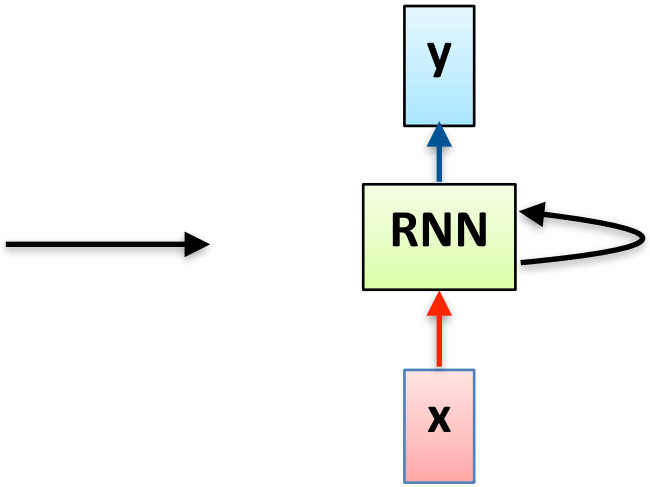
https://gist.github.com/karpathy/d4dee566867f8291f086

The sonnets

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
    Pity the world, or else this glutton be,
    To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
    This were to be new made when thou art old,
    And see thy blood warm when thou feel'st it cold.



at first:

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

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd 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 shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd 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 shuwy 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.
```

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd 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 shuwy 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.
```

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain 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.

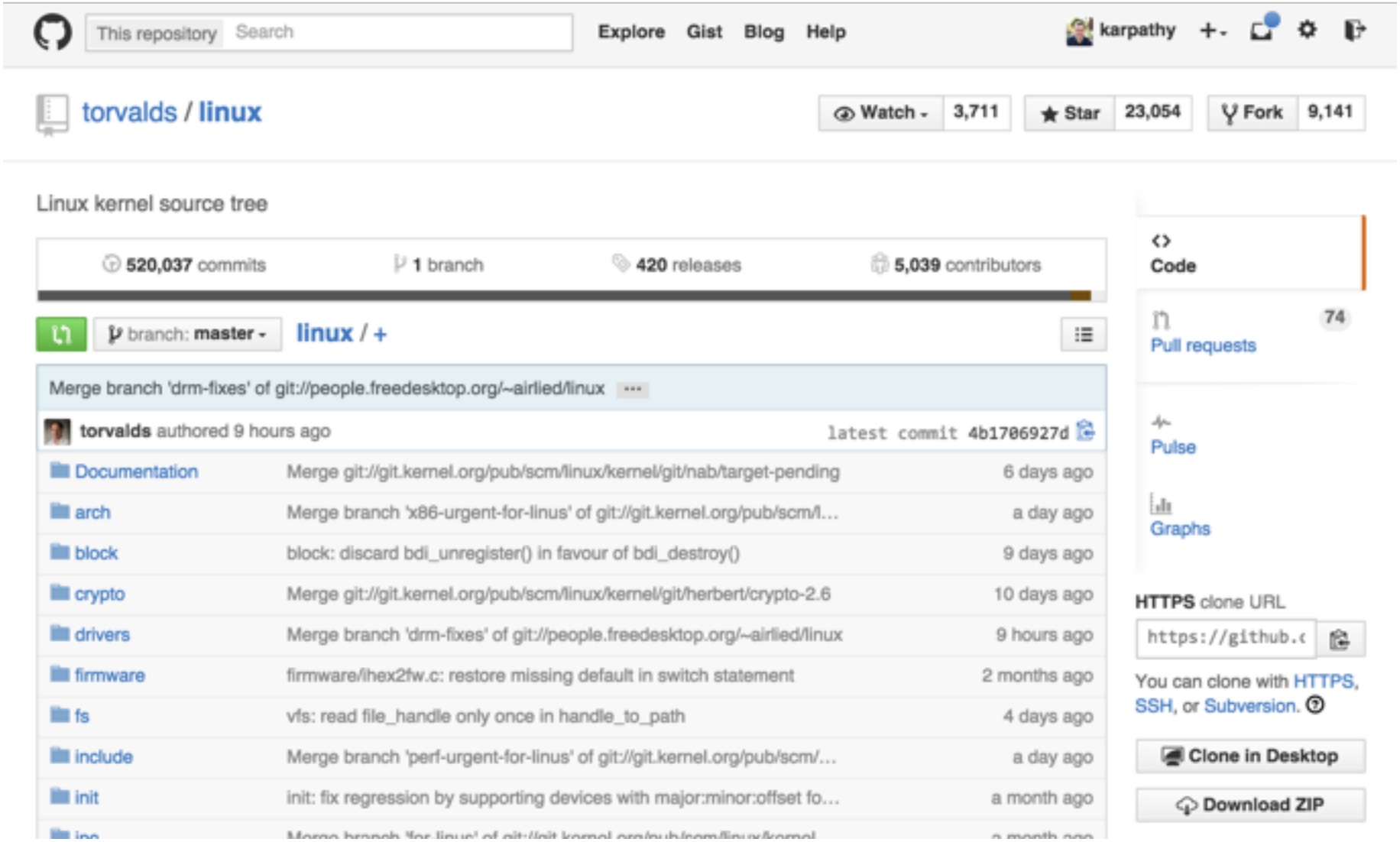
Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

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.


This repository
Search
Explore
Gist
Blog
Help
karpathy
torvalds / linux
Watch 3,711
Star 23,054
Fork 9,141
Linux kernel source tree
520,037 commits
1 branch
420 releases
5,039 contributors
branch: master
linux / +
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
torvalds authored 9 hours ago
latest commit 4b1706927d
Documentation Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending 6 days ago
arch Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l... a day ago
block block: discard bdi_unregister() in favour of bdi_destroy() 9 days ago
crypto Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6 10 days ago
drivers Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux 9 hours ago
firmware firmware/ihex2fw.c: restore missing default in switch statement 2 months ago
fs vfs: read file_handle only once in handle_to_path 4 days ago
include Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/... a day ago
init init: fix regression by supporting devices with major:minor:offset fo... a month ago
Code
74
Pull requests
Pulse
Graphs
HTTPS clone URL
https://github.c
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 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

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

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

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

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

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```