Announcements

Reminder: ps3 due tonight 10/8 at midnight (Boston)

- ps4 out today, due 10/15 (1 week)
- ps3 self-grading form out Monday, due 10/19
- Grades for ps1 & ps2 are being posted to blackboard (by Monday)
- Midterm 10/22 – have to finish test once began, should have blank paper that you will submit work/steps for a solution

Neural Networks IV

Recurrent Networks

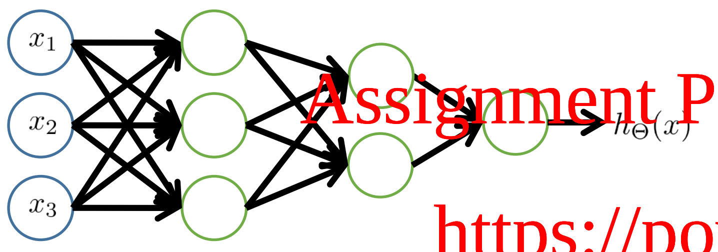
Today: Outline

- **Recurrent networks:** forward pass, backward pass
- **NN training strategies:** loss functions, dropout, etc.

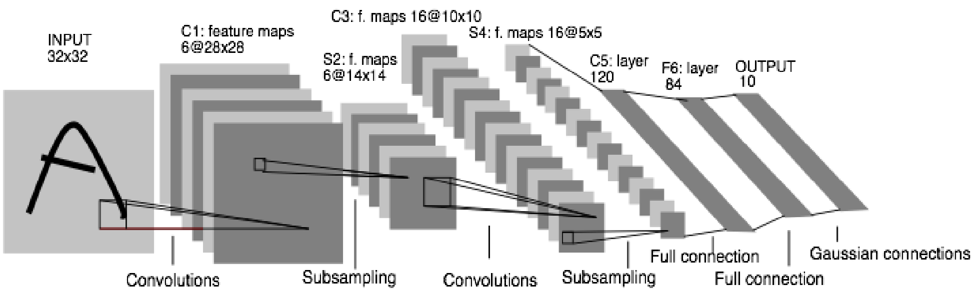
Network architectures

Feed-forward

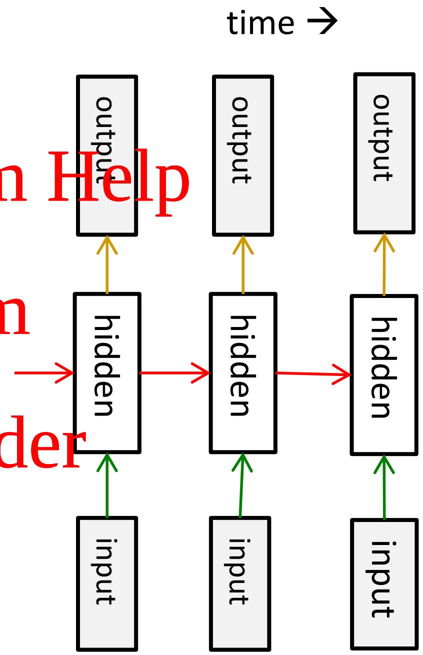
Fully connected



Convolutional



Recurrent



Neural Networks IV

Recurrent Architectures

Recurrent Networks for Sequences of Data

- Sequences in our world:
  - Audio
  - Text
  - Video
  - Weather
  - Stock market

- Sequential data is why we build RNN architectures.

- RNNs are tools for making predictions about sequences.

Limitations of Feed-Fwd Networks

- Limitations of feed-forward networks

– **Fixed length**

Inputs and outputs are of fixed lengths

– **Independence**

Data (example: images) are independent of one another

Advantages of RNN Models

- What feed-forward networks cannot do

  – **Variable length**

  "We would like to accommodate temporal sequences of various lengths."

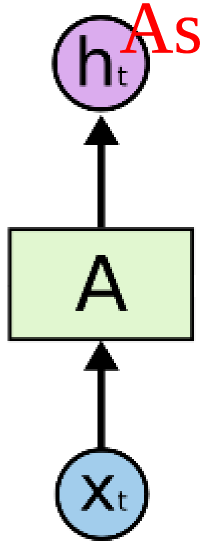
  – **Temporal dependence**

  "To predict where a pedestrian is at the next point in time, this depends on where he/she were in the previous time step."

Nervana, Tutorial, https://www.youtube.com/watch?v=Ukgii7Yd_cU

slide by Sarah Bargal

Vanilla Neural Network (NN)

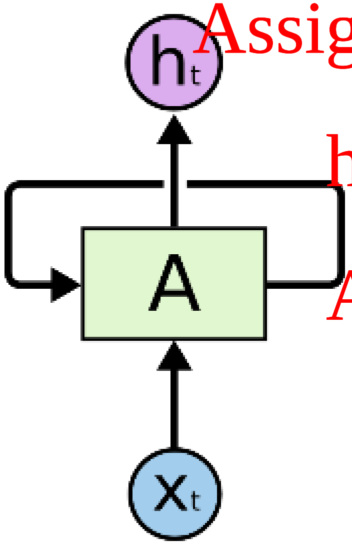


- NN

x_t : input/event
h_t : output/prediction
A : chunk of NN

Every input is treated independently.

Recurrent Neural Network (RNN)



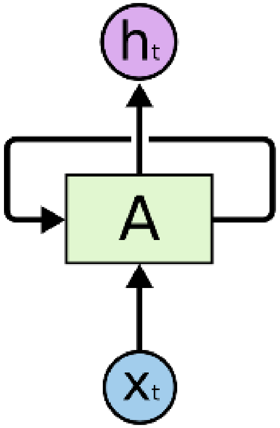
- RNN

The loop allows information to be passed from one time step to the next.

Now we are modeling the dynamics.

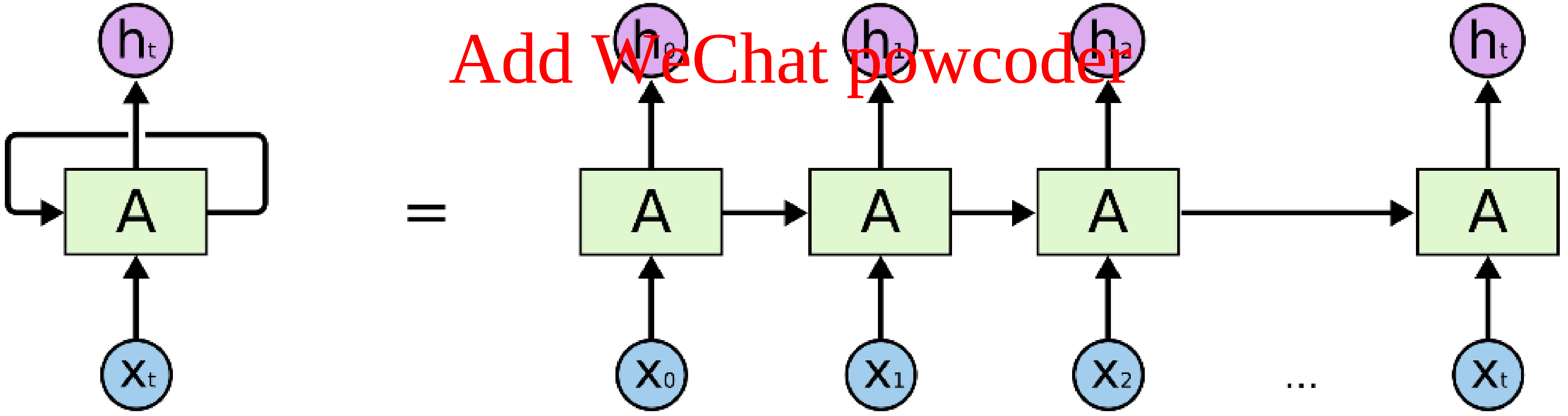
Recurrent Neural Network (RNN)

- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

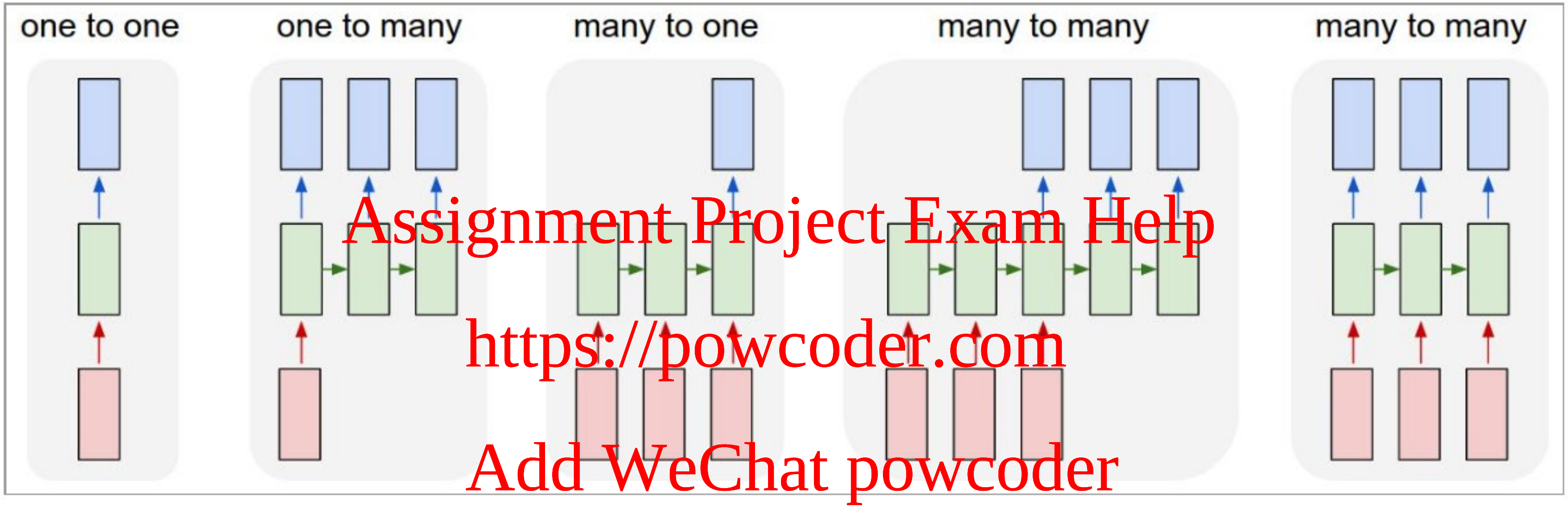


Recurrent Neural Network (RNN)

- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



RNN Architectures

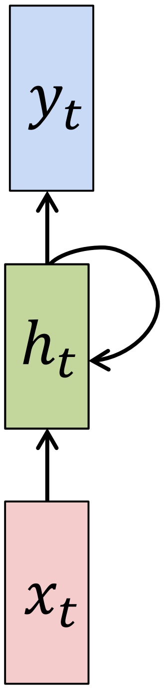


Vanilla NN

Recurrent NNs

Recurrent Neural Network

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



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

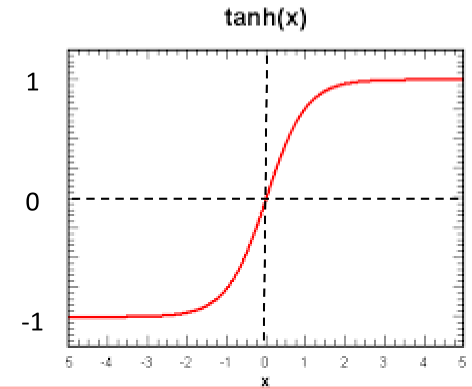
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

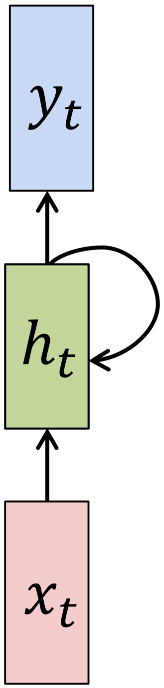
parameters

Recurrent Neural Network

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



$$\boxed{g(x) = 2\sigma(2x) - 1}$$



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

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

$$y_t = W_{hy}h_t$$

activation function
(elementwise)

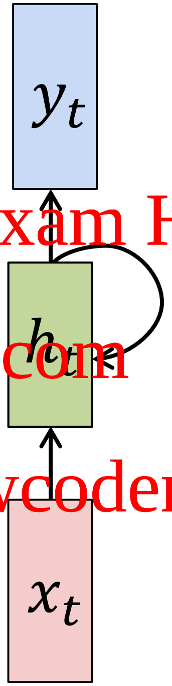
Neural Networks IV

Example: Character RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

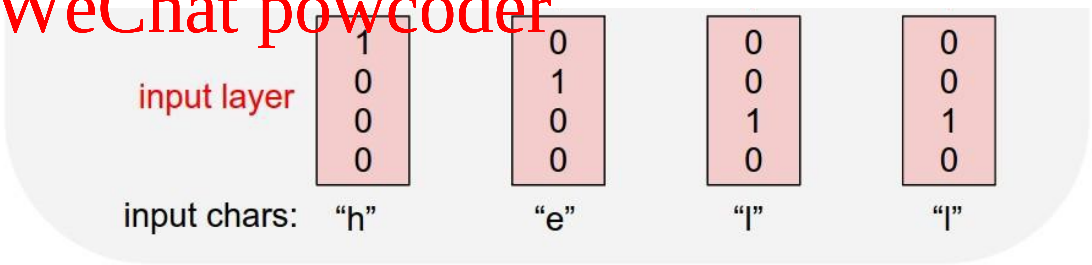
Example training sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

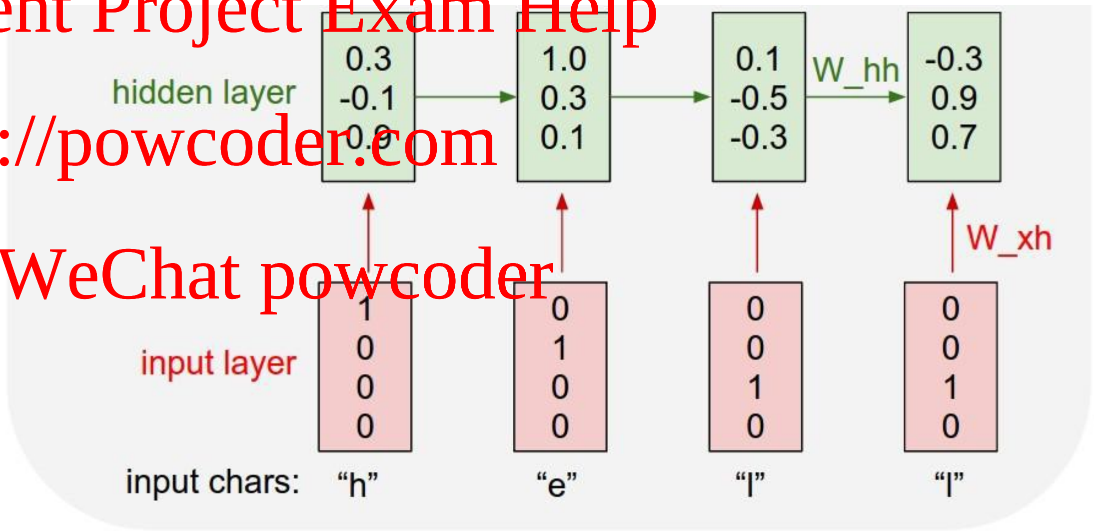


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

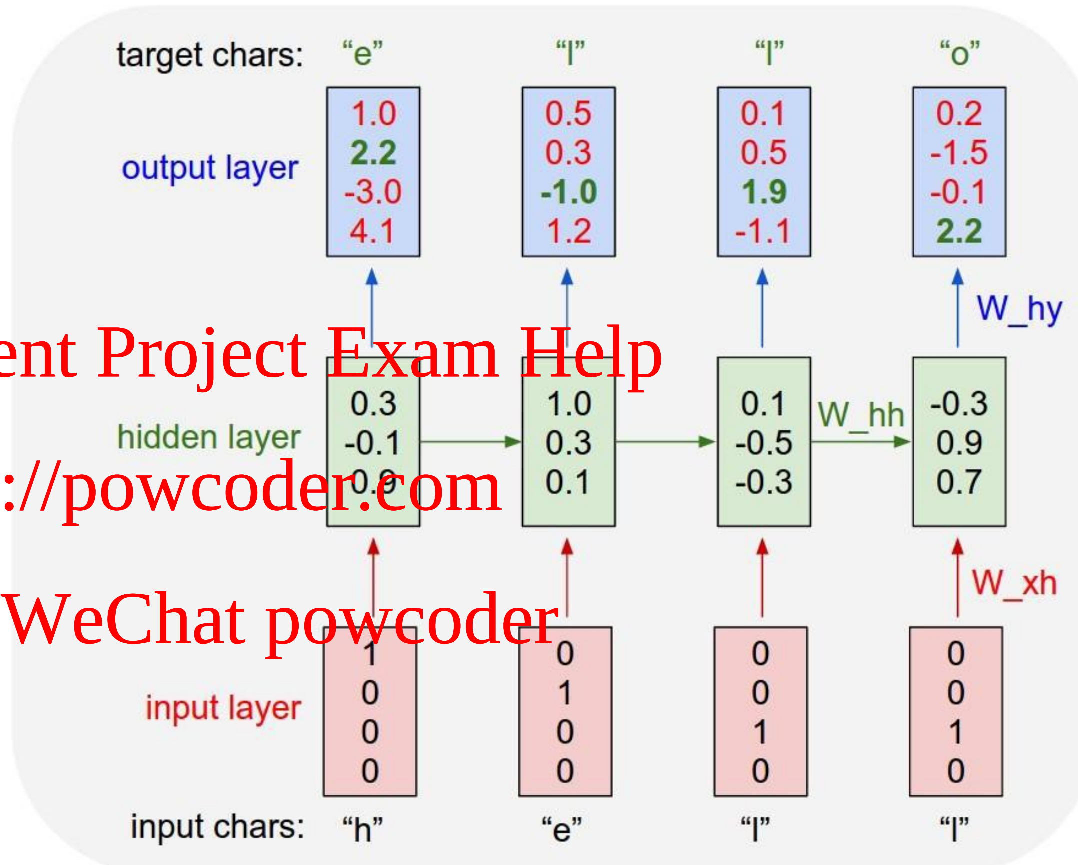
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

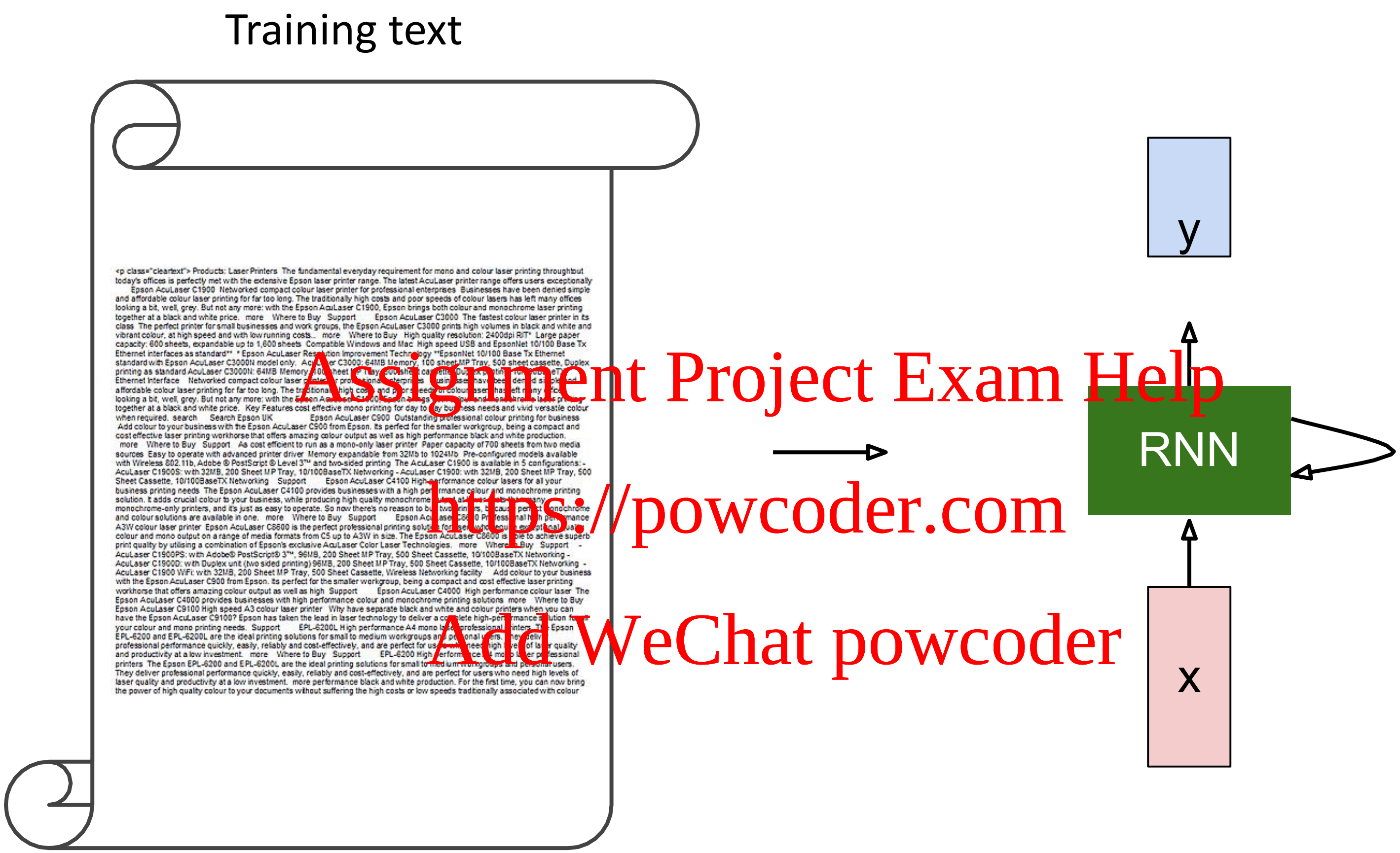
Example training sequence:
“hello”



[min-char-rnn.py gist: 112 lines of Python](https://gist.github.com/karpathy/d4dee566867f8291f086)

```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]

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)


Training text
y
RNN
x

Sonnet 116 – Let me not ...

by William Shakespeare

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

at first:

```
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.

Neural Networks IV

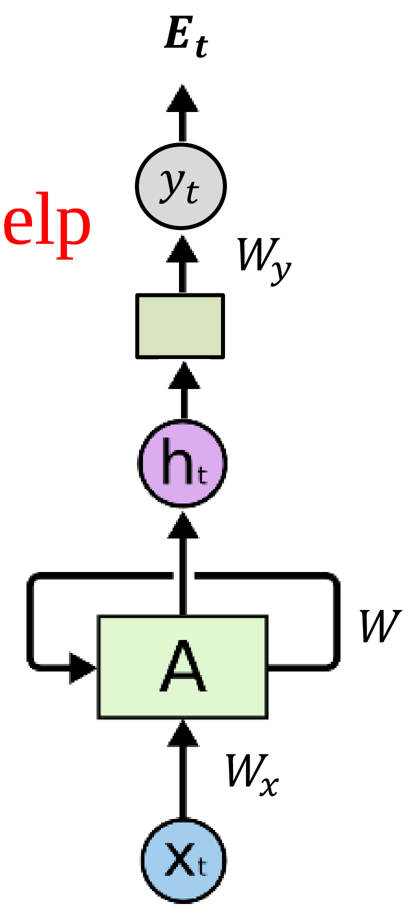
Learning in RNNs

Forward pass

- Forward pass through time

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

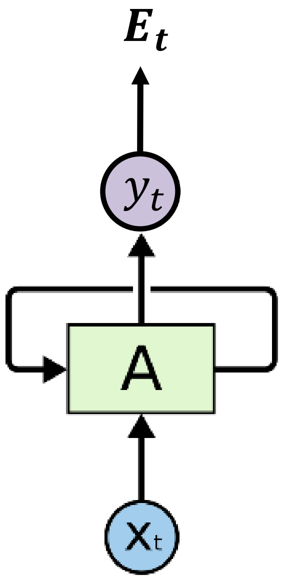
$$y_t = W_y \phi(h_t)$$



Recurrent Neural Network (RNN)

Aside: Forward pass

$$\boxed{\begin{aligned} h_t &= W\phi(h_{t-1}) + W_x x_t \\ y_t &= W_y\phi(h_t) \end{aligned}}$$



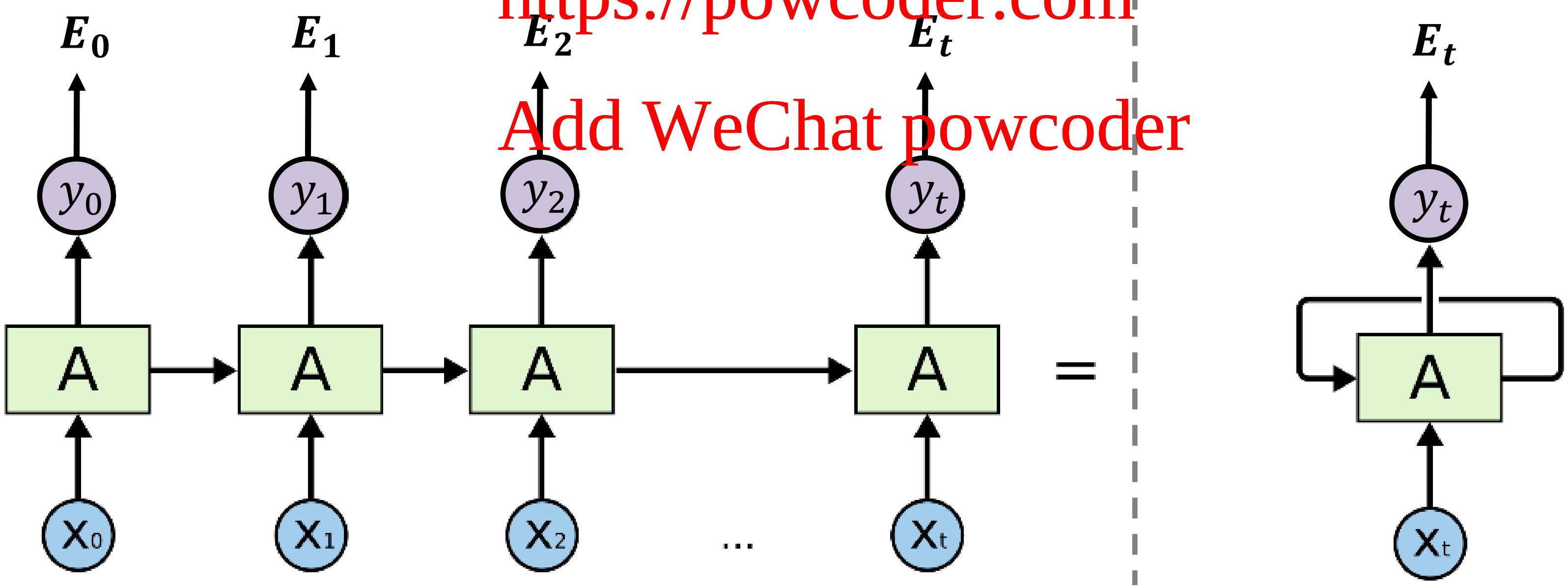
Recurrent Neural Network (RNN)

- Error or cost is computed for each prediction.

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$

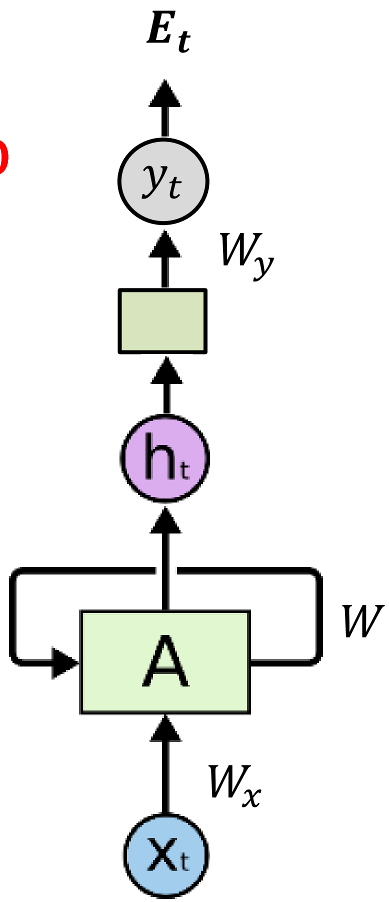


Backprop Through Time

- Backpropagation through time

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

Aside: Forward pass



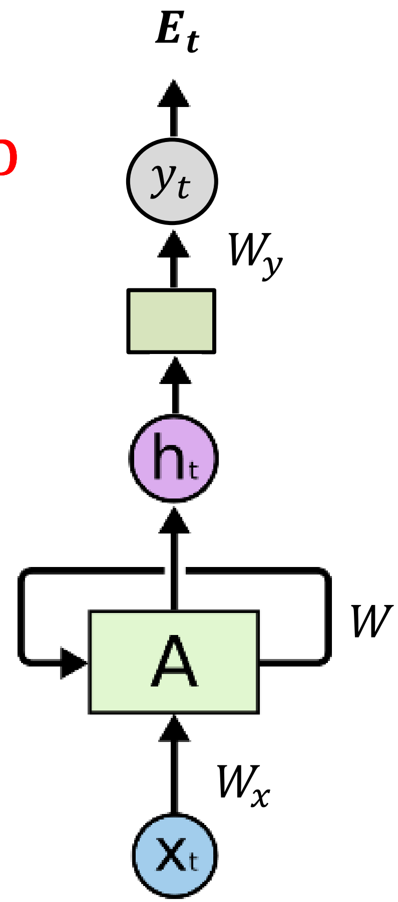
BP TT

- Backpropagation through time

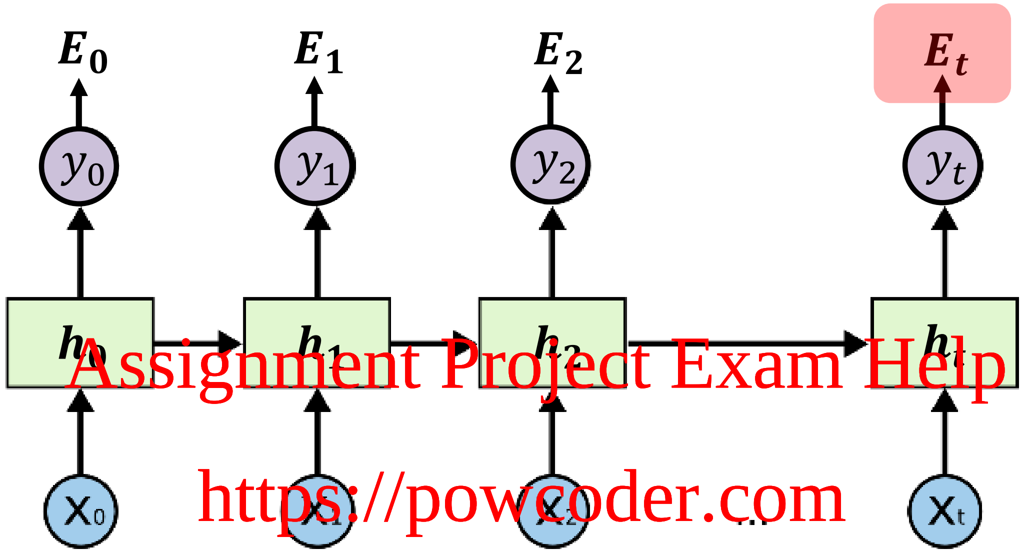
$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \boxed{\frac{\partial E_t}{\partial W}}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Aside: Forward pass

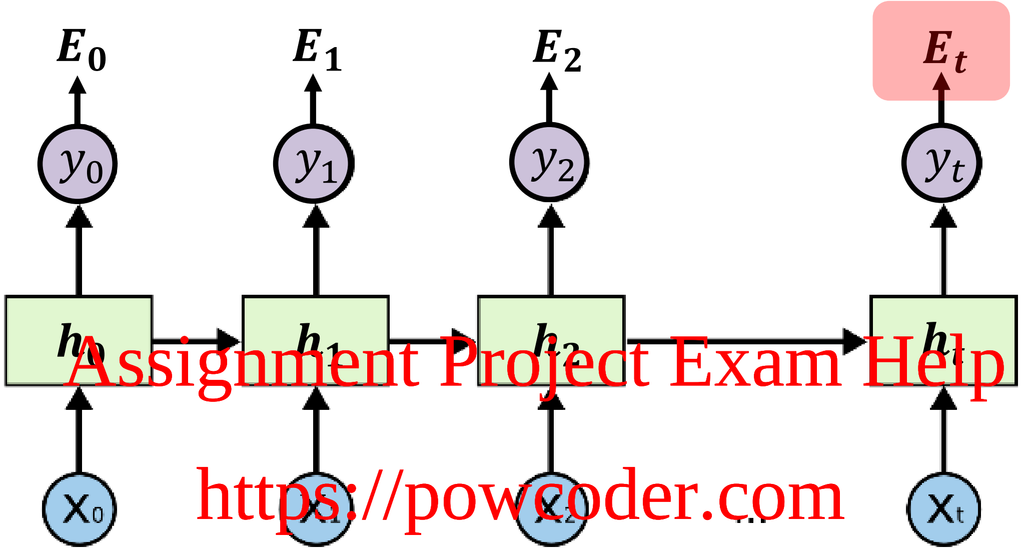


BP TT



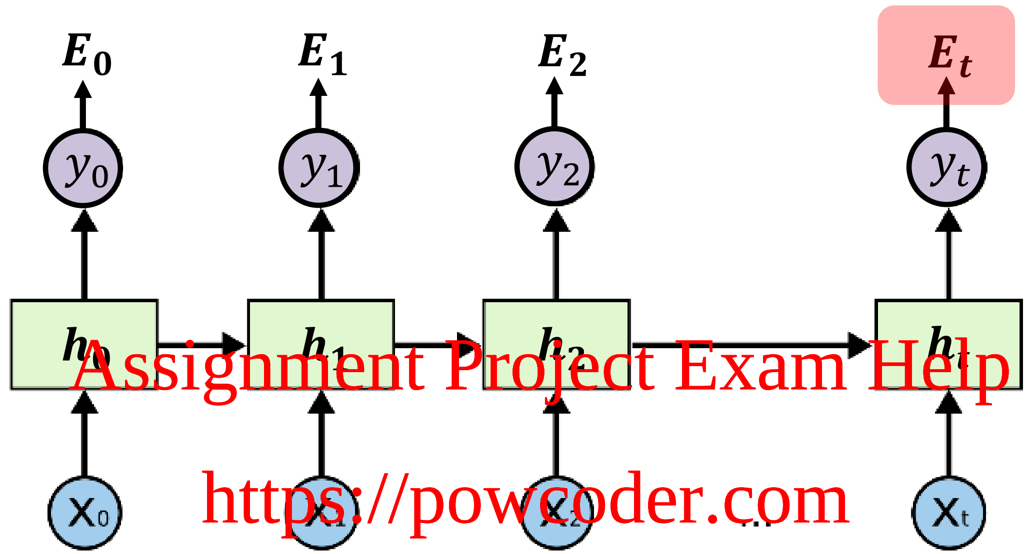
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

BP TT



$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

BP TT

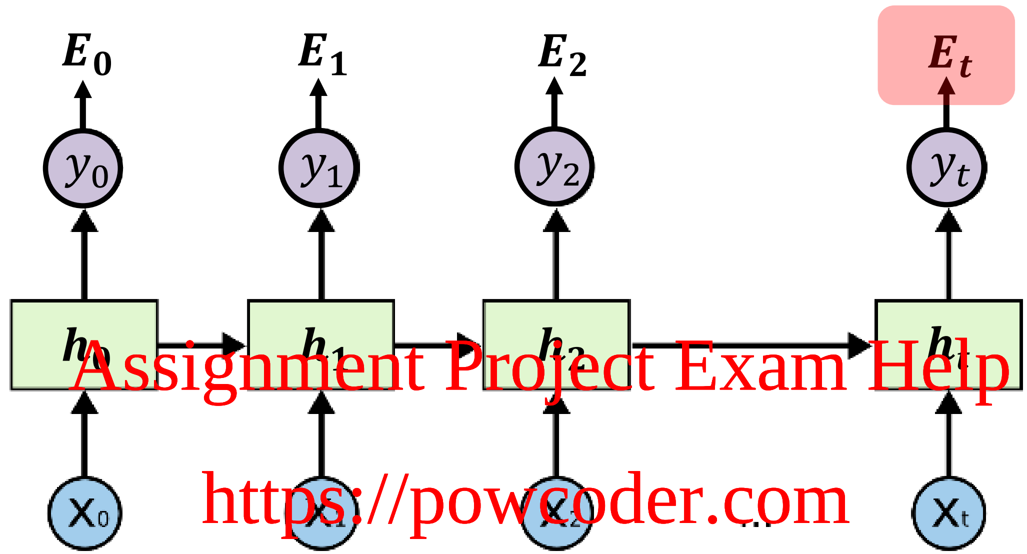


$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Nando de Freitas , lecture, https://www.youtube.com/watch?v=56TYLaQN4N8 slide by Sarah Bargal

BP TT



$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

For example @ $t = 2$,

$$\frac{\partial h_2}{\partial h_0} = \prod_{i=1}^{2} \frac{\partial h_i}{\partial h_{i-1}} = \frac{\partial h_1}{\partial h_0} \frac{\partial h_2}{\partial h_1}$$

Vanishing (and Exploding) Gradients

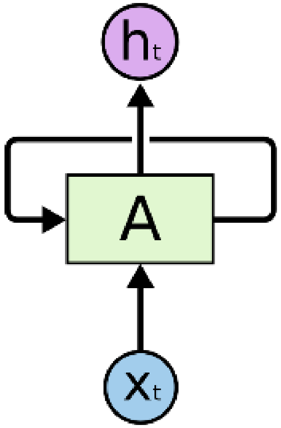
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

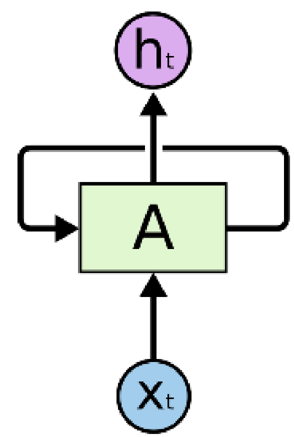
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

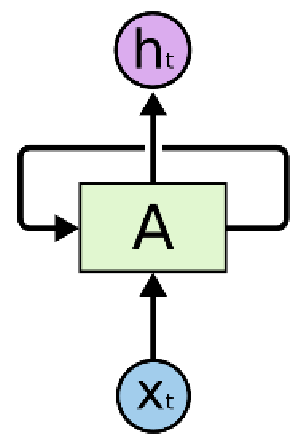
$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

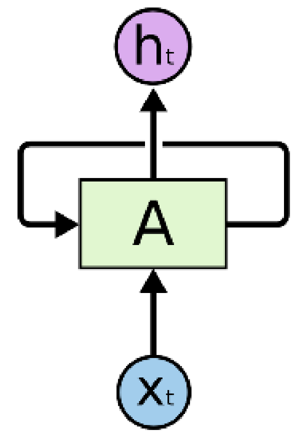
$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \|W^T\| \, \|diag[\phi'(h_{i-1})]\|$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

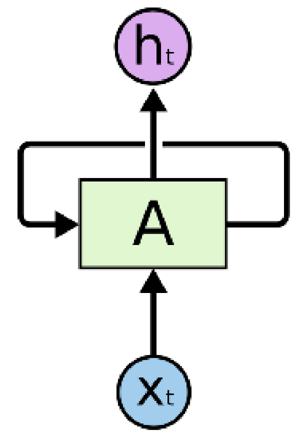
$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \|W^T\| \, \|diag[\phi'(h_{i-1})]\| \leq \gamma_W \gamma_\phi$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

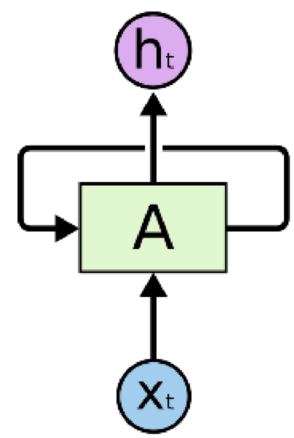
$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \|W^T\| \, \|diag[\phi'(h_{i-1})]\| \leq \gamma_W \gamma_\phi$$

$$\prod_{i=k+1}^{t} \left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq (\gamma_W \gamma_\phi)^{t-k}$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T diag[\phi'(h_{i-1})]$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \|W^T\| \, \|diag[\phi'(h_{i-1})]\| \leq \gamma_W \gamma_\phi$$

<1 vanishing

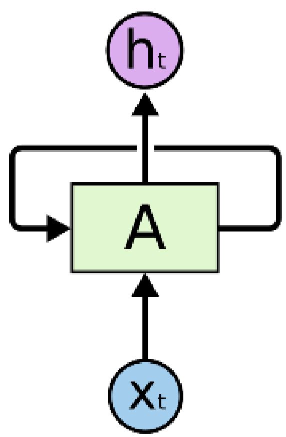
>1 exploding

$$\prod_{i=k+1}^{t} \left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq (\gamma_W \gamma_\phi)^{t-k}$$

Aside: Forward pass

$$h_t = W\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$



Vanishing (and Exploding) Gradients

- Exploding Gradients
  - Easy to detect
  - Clip the gradient at a threshold

- Vanishing Gradients
  - More difficult to detect
  - Architectures designed to combat the problem of vanishing gradients. Example: LSTMs by *Schmidhuber et al.*

Neural Networks IV

Training strategies

Universality

- Why study neural networks in general?
  - Neural network can approximate any continuous function, even with a single hidden layer!
  - http://neuralnetworksanddeeplearning.com/chap4.html

Why Study Deep Networks?

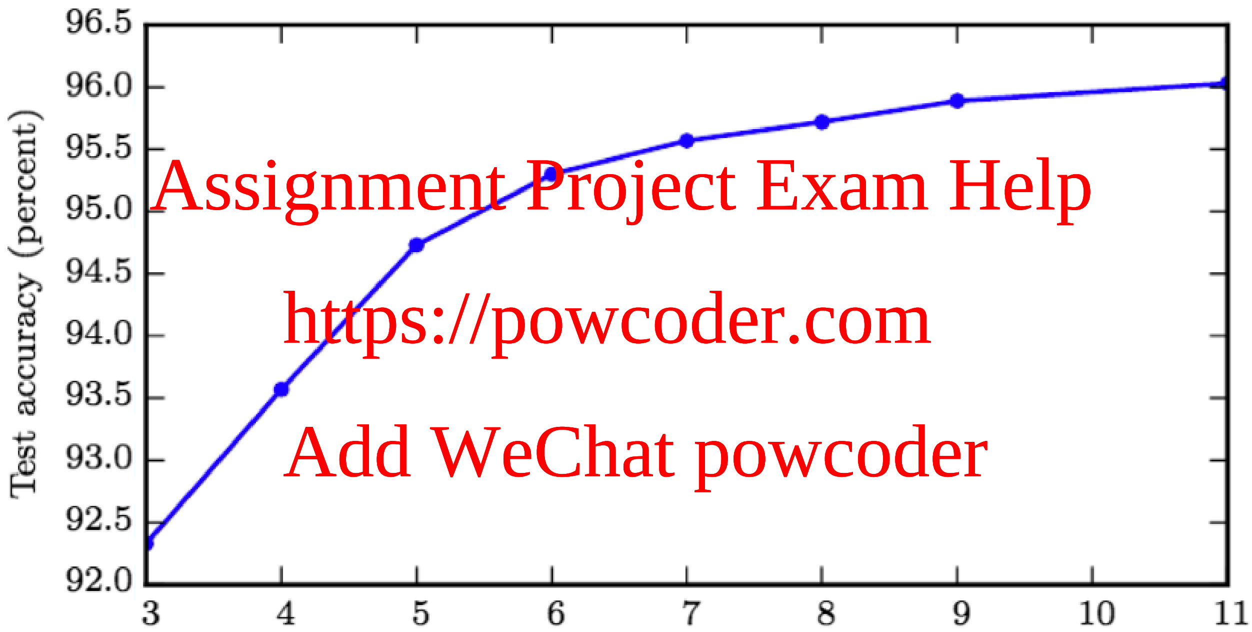
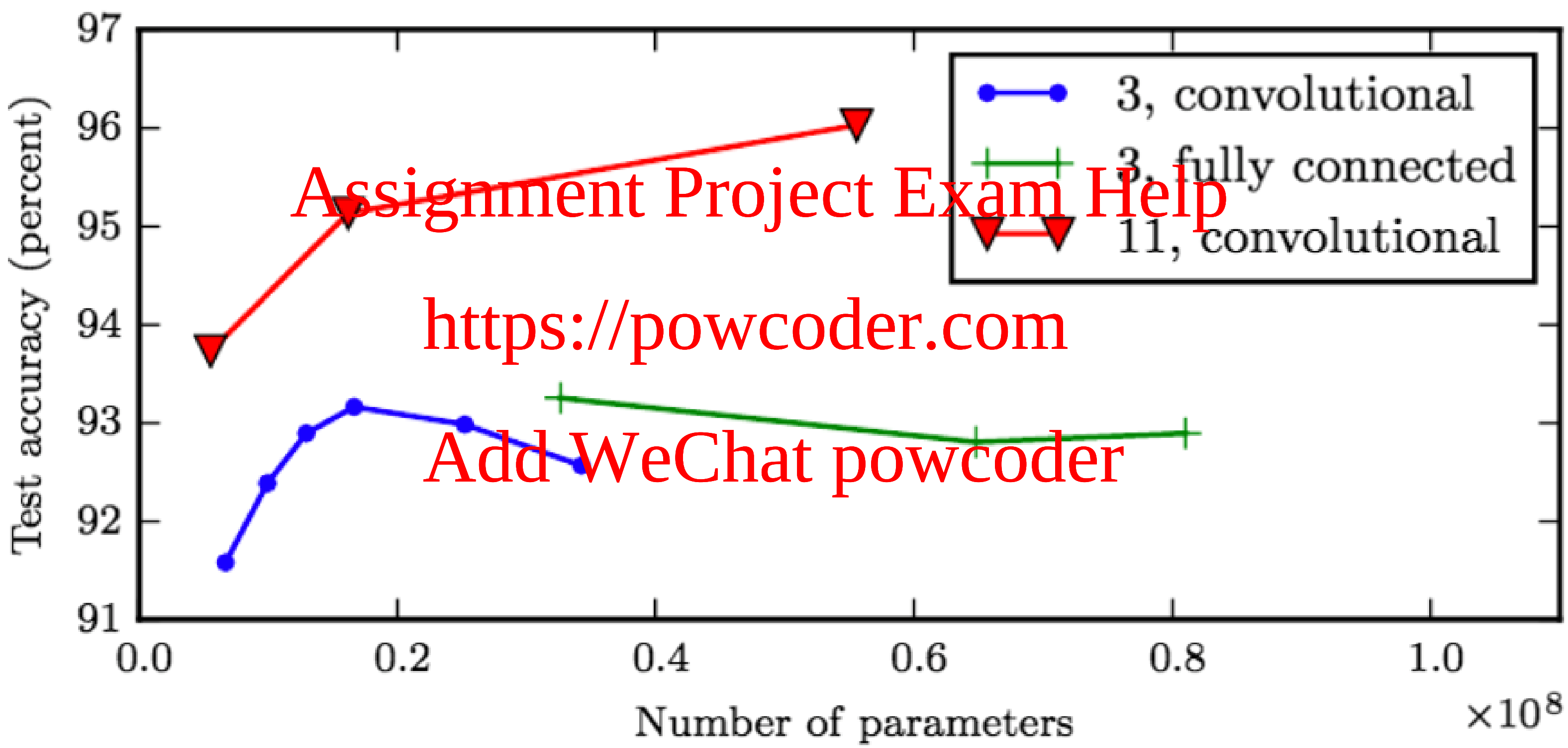


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.

Efficiency of convnets



But… Watch Out for Vanishing Gradients

- Consider a simple network, and perform backpropagation

  - For simplicity, just a single neuron
  - Sigmoid at every layer, $z_j = \sigma(w_j a_{j-1} + b_j), a_j = \sigma(z_j)$
  - Cost function *C*

- Gradient $\partial C / \partial b_1$ is a product of terms:

$$\partial C / \partial b_1 = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) (\partial C / \partial a_4)$$

Vanishing Gradients

- Gradient of sigmoid is in (0,1/4)
- Weights are also typically initialized in (0,1)
- Products of small numbers → small gradients
- Backprop does not change weights in earlier layers by much!
  - This is an issue with backprop, not with the model itself

RNNs: vanishing and exploding gradients

- Exploding: easy to fix, clip the gradient at a threshold
- Vanishing: More difficult to detect
- Architectures designed to combat the problem of vanishing gradients. Example: LSTMs by *Schmidhuber et al.*

Rectified Linear Units (RELU)

- Alternative non-linearity:

$$g(x) = \max(0, x)$$

- Gradient of this function?
  - Note: need subgradient descent here.
- https://cs224d.stanford.edu/notebooks/vanishing_grad_example.html
- Increasing the number of layers can result in requiring exponentially fewer hidden units per layer (see "Understanding Deep Neural Networks with Rectified Linear Units")
- Biological considerations
  - On some inputs, biological neurons have no activation
  - On some inputs, neurons have activation proportional to input

Other Activation Functions

- Leaky ReLU: $g(x) = \max(0, x) + \alpha \min(0, x) \quad (\alpha \approx .01)$
- Tanh: $g(x) = 2\sigma(2x) - 1$
- Radial Basis Functions: $g(x) = \exp(-(u - x)^2/\sigma^2)$
- Softplus: $g(x) = \log(1 + e^x)$
- Hard Tanh: $g(x) = \max(-1, \min(1, x))$
- Maxout: $g(x) = \max_{j \in \mathbb{G}} x_j$
- ....

Architecture Design and Training Issues

- How many layers? How many hidden units per layer? How to connect layers together? How to optimize?
  - Cost functions
  - L2/L1 regularization
  - Data Set Augmentation
  - Early Stopping
  - Dropout
  - Minibatch Training
  - Momentum
  - Initialization
  - Batch Normalization

Next Class

Computing cluster/Tensorflow Intro (next Thursday):

Intro to SCC and Tensorflow: please have laptops ready to follow along with the lecture. Expected to last 2 hours