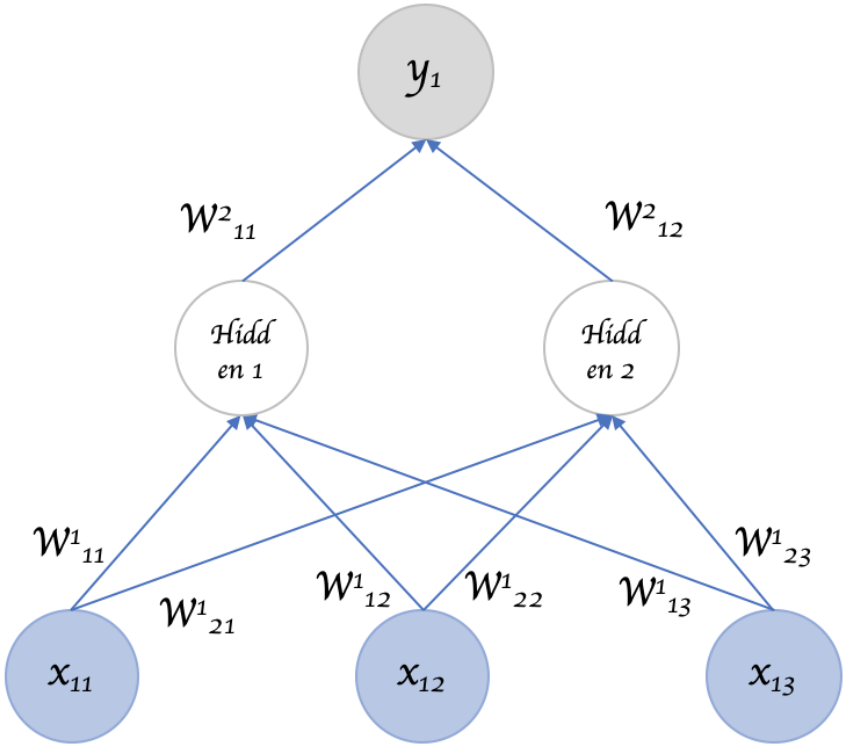
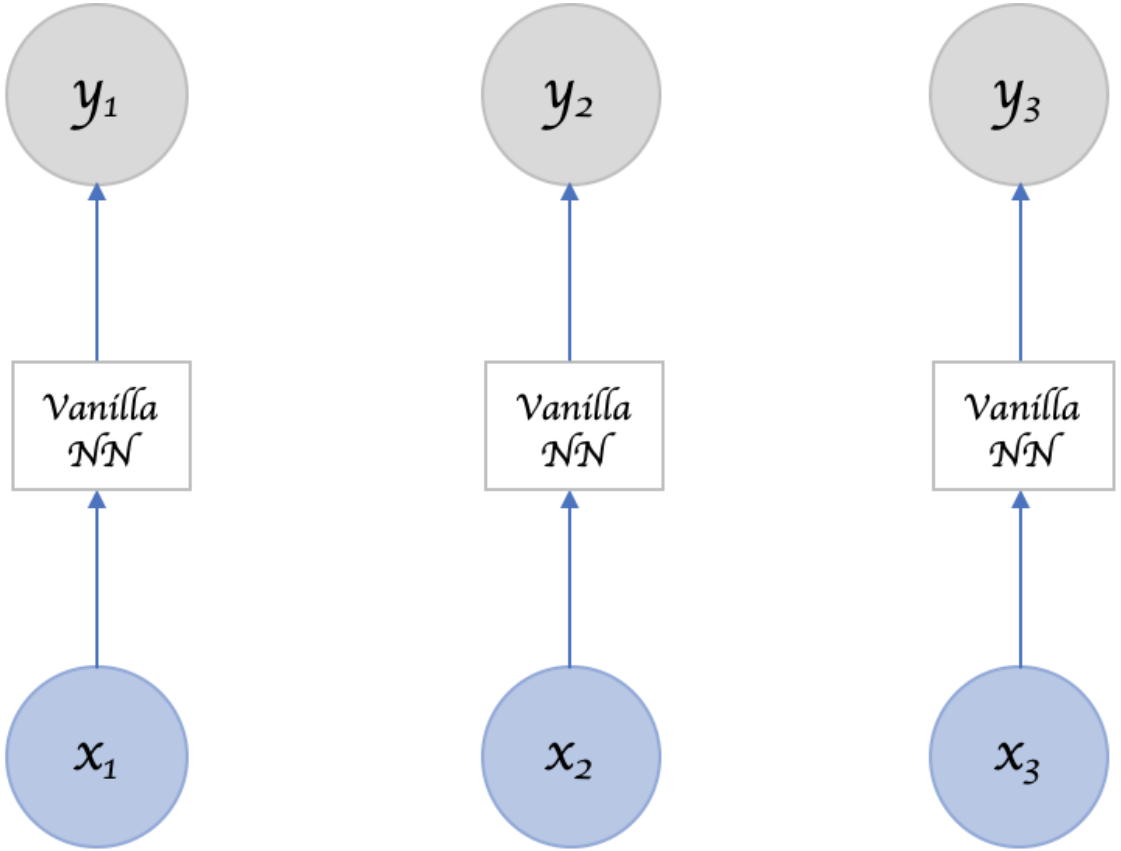
Introduction to Recurrent Neural Network

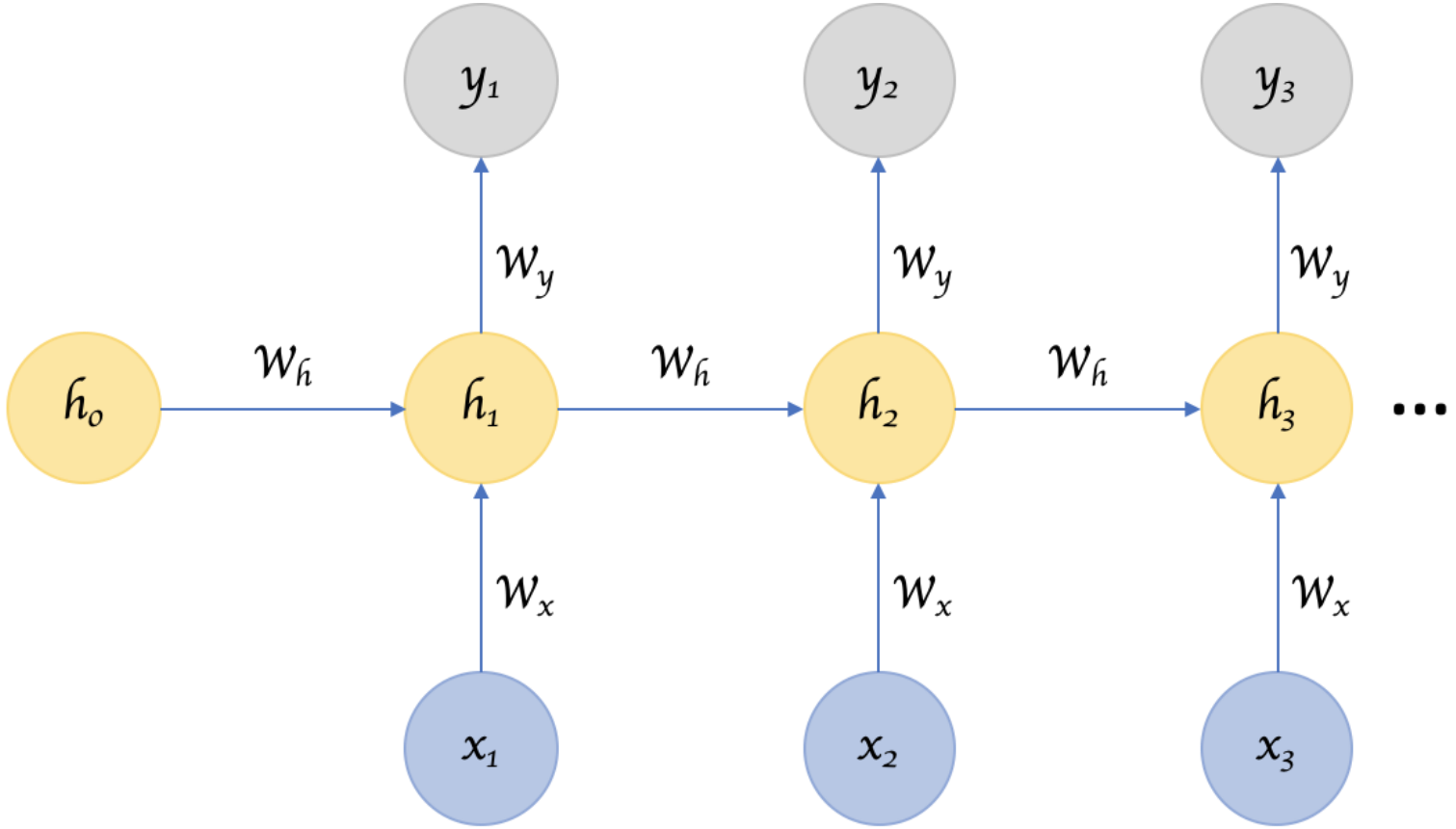
A simple neural network with 1 hidden layer



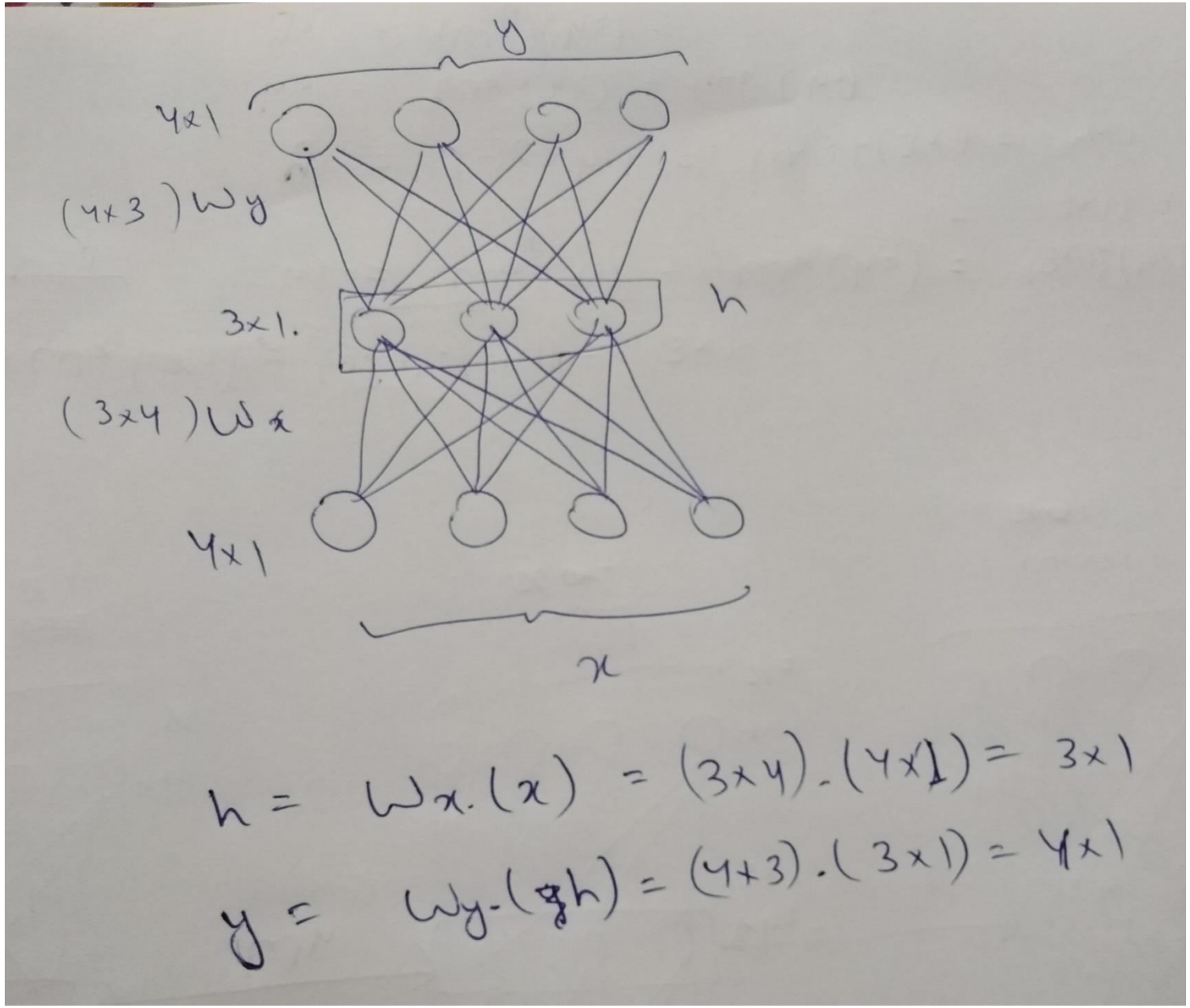
A simple neural network with 1 hidden layer



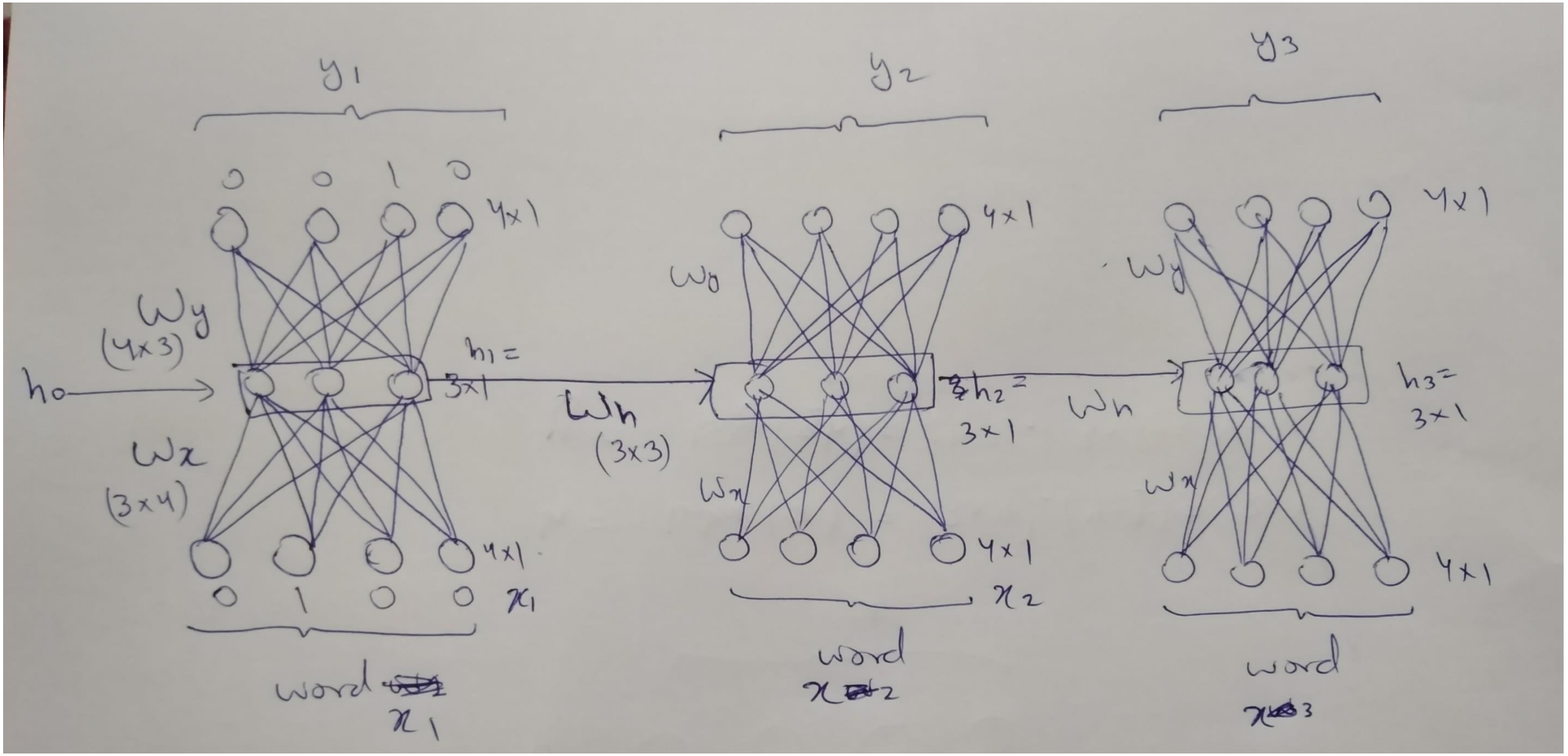
A recurrent neural network with 1 hidden layer



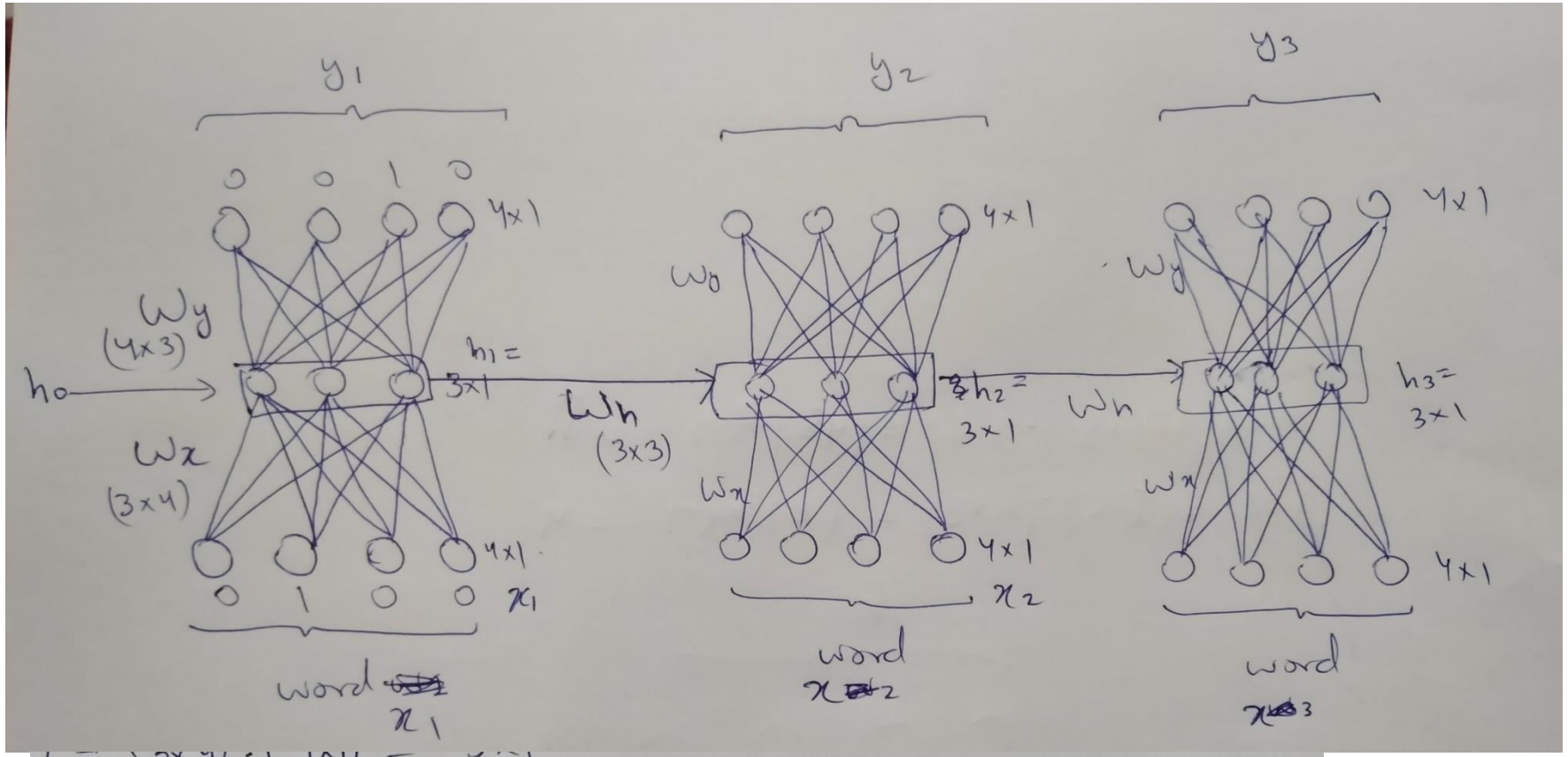
A simple neural network with 1 hidden layer



A simple recurrent neural network with 1 hidden layer



A simple recurrent neural network with 1 hidden layer



$h_1 = W_x(x_1) + W_h(h_0) = (3\times4)(4\times1) + (3\times3)(3\times1)$

$= (3\times1) + (3\times1) = 3\times1$

$y_1 = W_y(h_1) = (4\times3)(3\times1) = 4\times1$

$h_2 = W_x(x_2) + W_h(h_1)$

$y_2 = (W_y)(h_2) =$

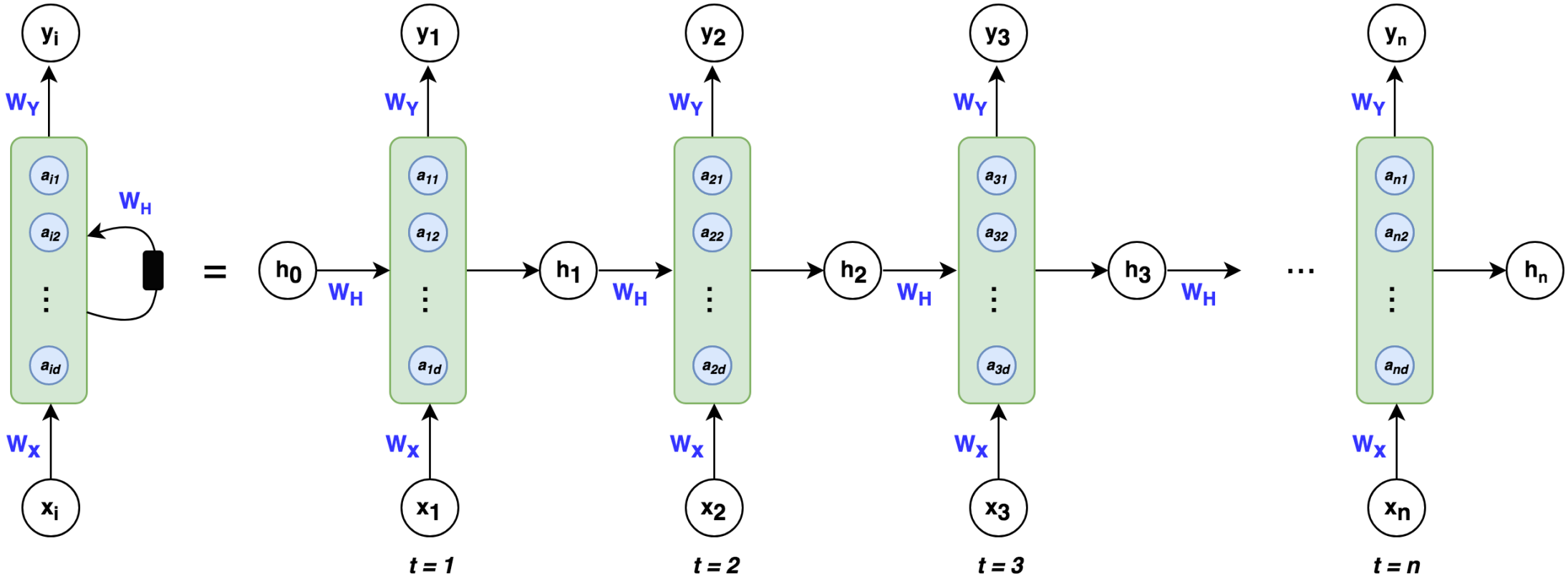
Applications of RNN

Sequence Data

- Text (sequence of words or sequence of characters)
- Stock prices (sequence of stock prices daily)
- DNA sequence data
- Consumer purchase history
- Web surfing history
- Many more

RNN Equations for Forward Pass

Left: Shorthand notation often used for RNNs, **Right**: Unfolded notation for RNNs



Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

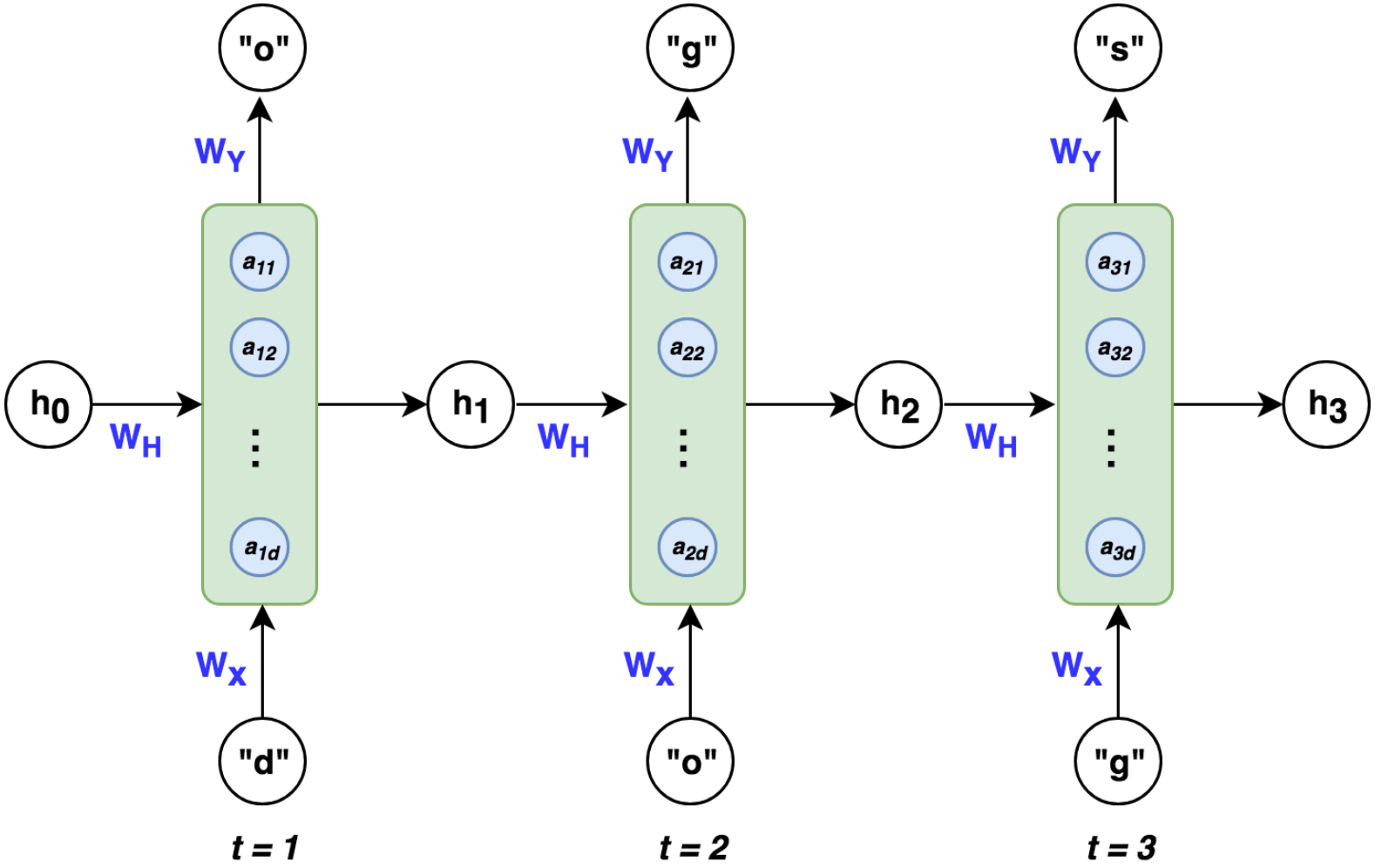
One-Hot Encoding of the word “dogs”

$$d = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad o = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad g = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad s = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

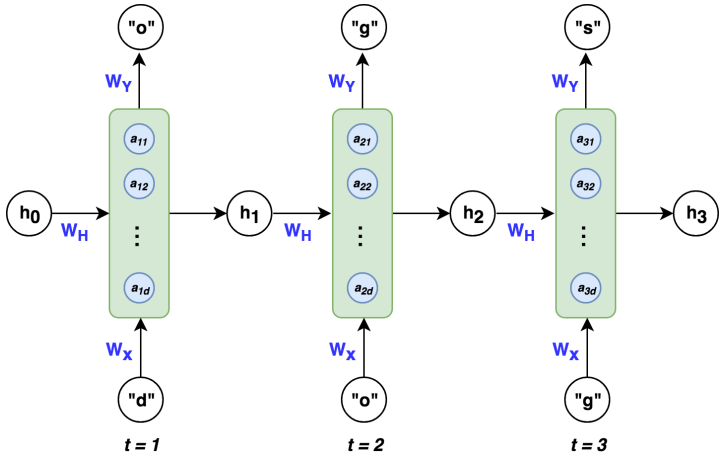
An Example

RNN architecture predicting the letter “s” in “dogs”



Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

RNN Equations



$$a_t = W_H h_{t-1} + W_X X_t$$ **Hidden Nodes**

Activation Function

Output From Hidden State $\longrightarrow h_t = tanh(a_t)$

Hidden State

Prediction at time t $\longrightarrow y_t = softmax(W_Y h_t)$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

RNN Equations

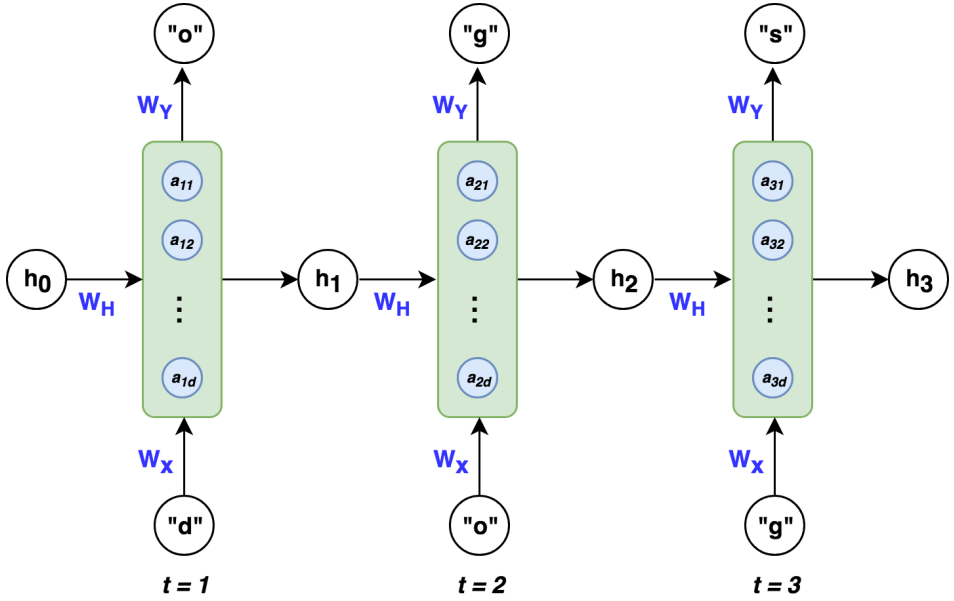
$$a_t = W_H h_{t-1} + W_X X_t$$

$(dxd)\ (dx1)\quad (kx1)\quad (dxk)$

$$(dx1) \longrightarrow h_t = tanh(a_t)$$

(kxd)

$$(kx1) \longrightarrow y_t = softmax(W_Y h_t)$$



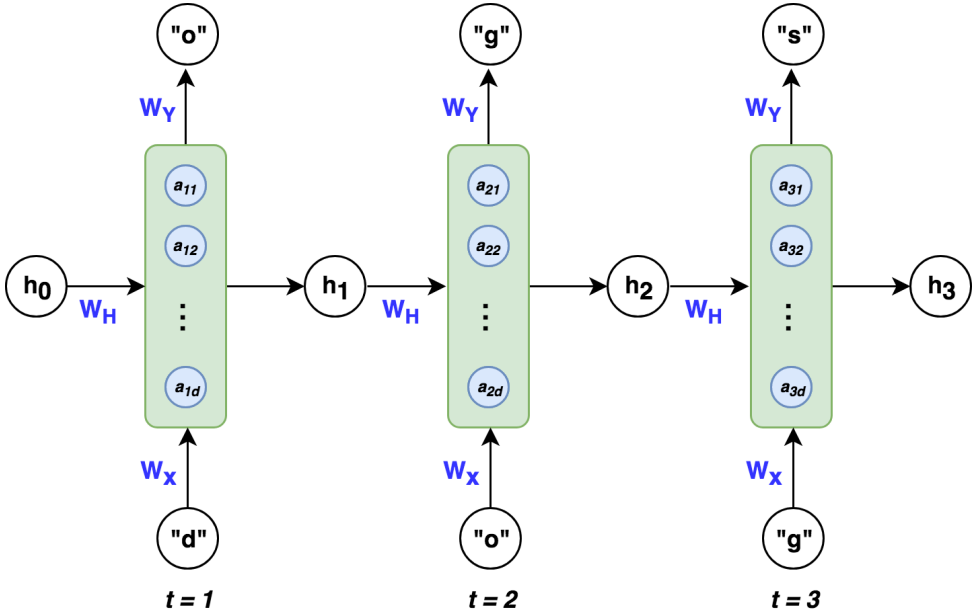
Where

k is the dimension of the input vector x_i

d is the number of hidden nodes

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Matrix Dimensions



$$(3x3)\ (3x1) \qquad (4x1)$$

$$a_t = W_H h_{t-1} + W_X X_t$$

$$(3x4)$$

$$(3x1) \longrightarrow h_t = tanh(a_t)$$

$$(4x3)$$

$$(4x1) \longrightarrow y_t = softmax(W_Y h_t)$$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Forward Propagation

$$a_t = W_H h_{t-1} + W_X X_t$$ Hidden Nodes

Activation Function

Output From Hidden State $\longrightarrow$ $h_t = tanh(a_t)$

Hidden State

Prediction at time t $\longrightarrow$ $y_t = softmax(W_Y h_t)$

At t =1

$$a_1 = \begin{pmatrix} W_{H,11} & W_{H12} & W_{H,13} \\ W_{H,21} & W_{H,22} & W_{H,23} \\ W_{H,31} & W_{H,32} & W_{H,33} \end{pmatrix} \begin{pmatrix} h_{0,1} \\ h_{0,2} \\ h_{0,3} \end{pmatrix} + \begin{pmatrix} W_{X,11} & W_{X,12} & W_{X,13} & W_{X,14} \\ W_{X,21} & W_{X,22} & W_{X,23} & W_{X,24} \\ W_{X,31} & W_{X,32} & W_{X,33} & W_{X,34} \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \end{pmatrix} = \begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}) = \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} W_{Y,11} & W_{Y,12} & W_{Y,13} \\ W_{Y,21} & W_{Y,22} & W_{Y,23} \\ W_{Y,31} & W_{Y,32} & W_{Y,33} \\ W_{Y,41} & W_{Y,42} & W_{Y,43} \end{pmatrix} \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}) = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ y_{1,3} \\ y_{1,4} \end{pmatrix}$$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Forward Propagation

$$a_t = W_H h_{t-1} + W_X X_t$$ Hidden Nodes

Activation Function

Output From Hidden State $\longrightarrow$ $h_t = tanh(a_t)$

Hidden State

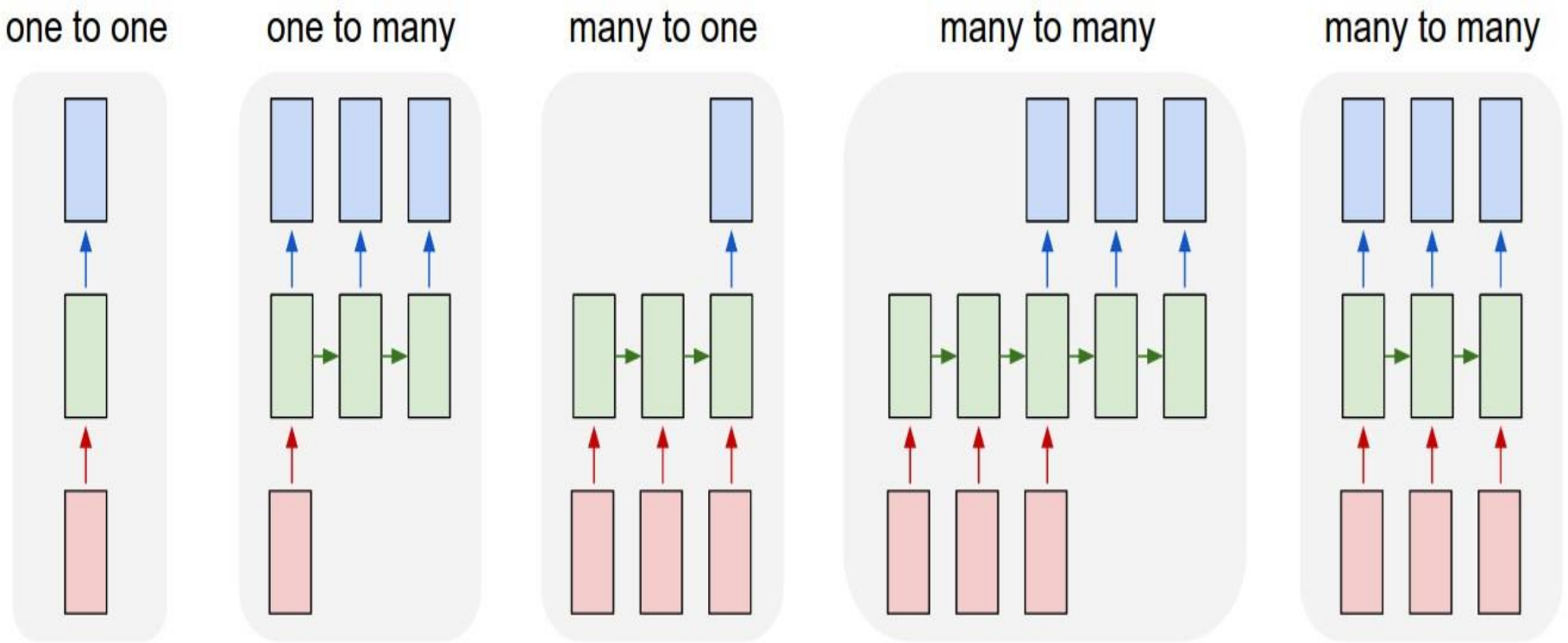
Prediction at time t $\longrightarrow$ $y_t = softmax(W_Y h_t)$

$$a_1 = \begin{pmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.2 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}$$

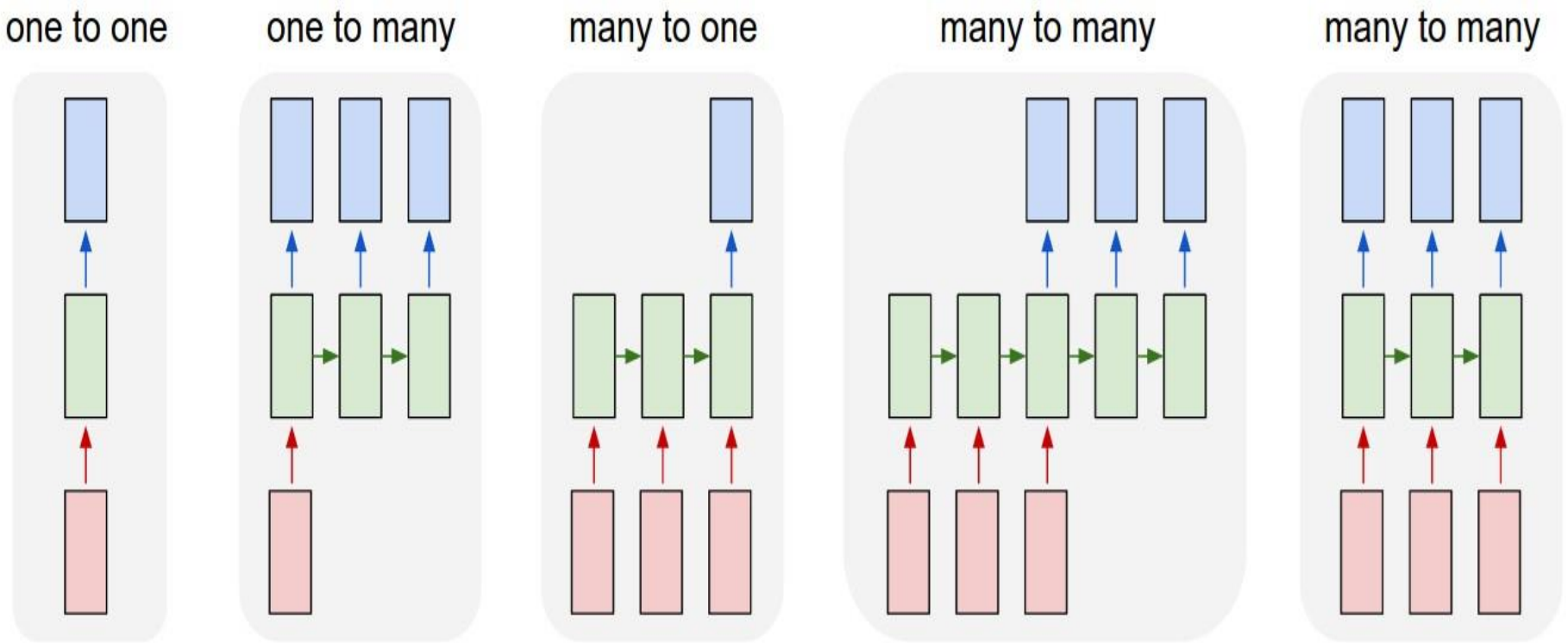
$$h_1 = tanh(\begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}) = \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{pmatrix} \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}) = \begin{pmatrix} 0.32 \\ 0.21 \\ 0.24 \\ 0.22 \end{pmatrix}$$

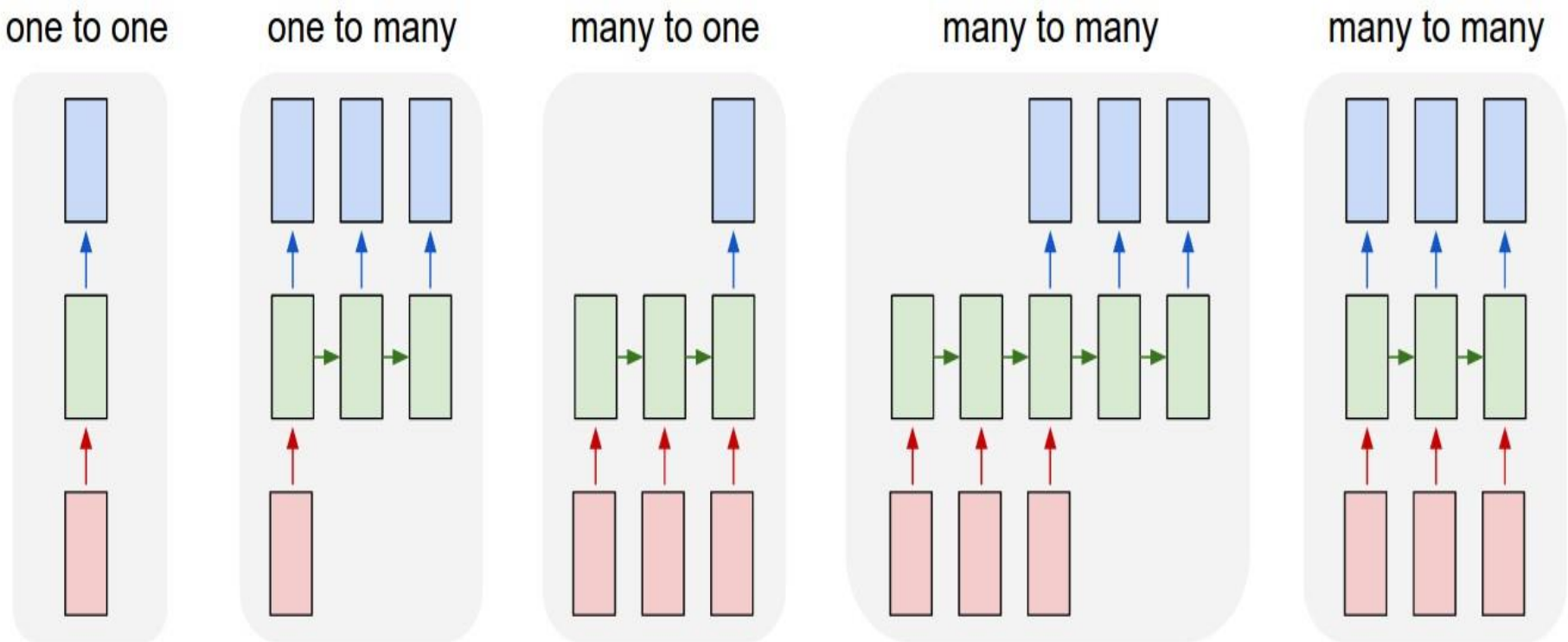
Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f



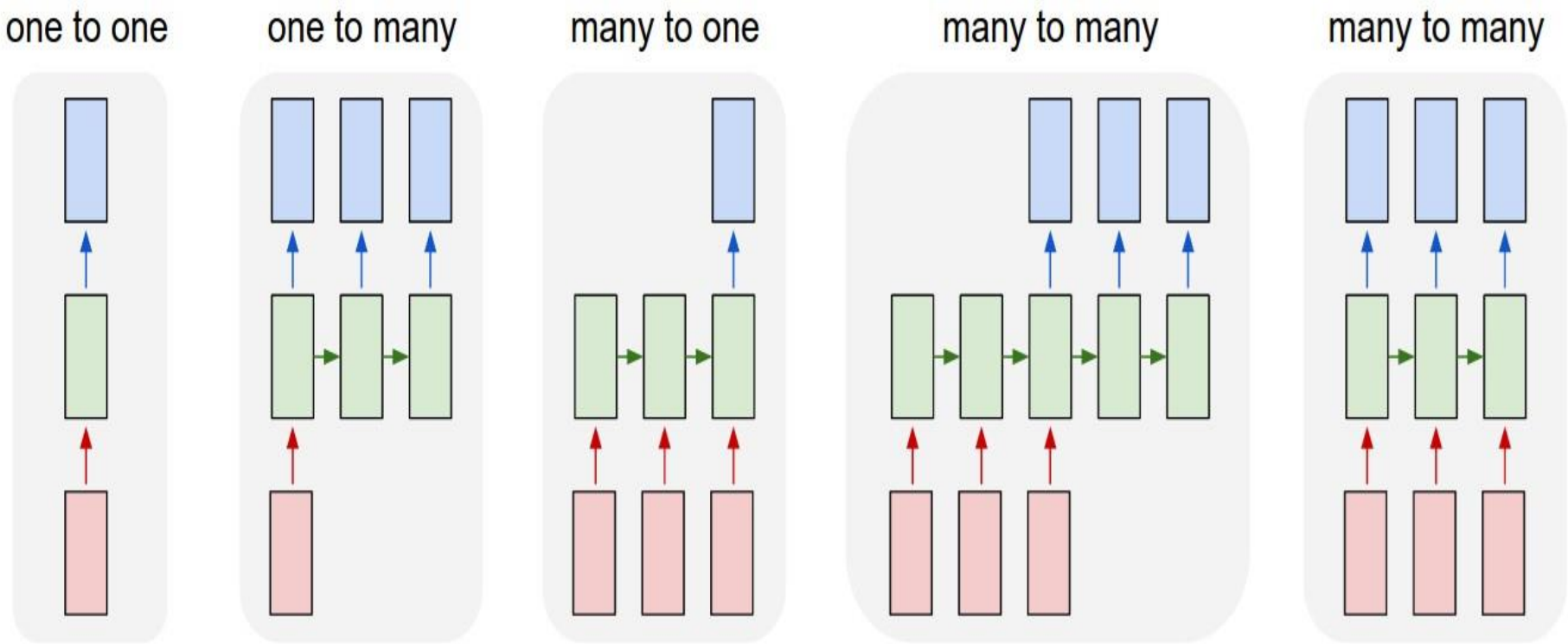
Vanilla Neural Networks



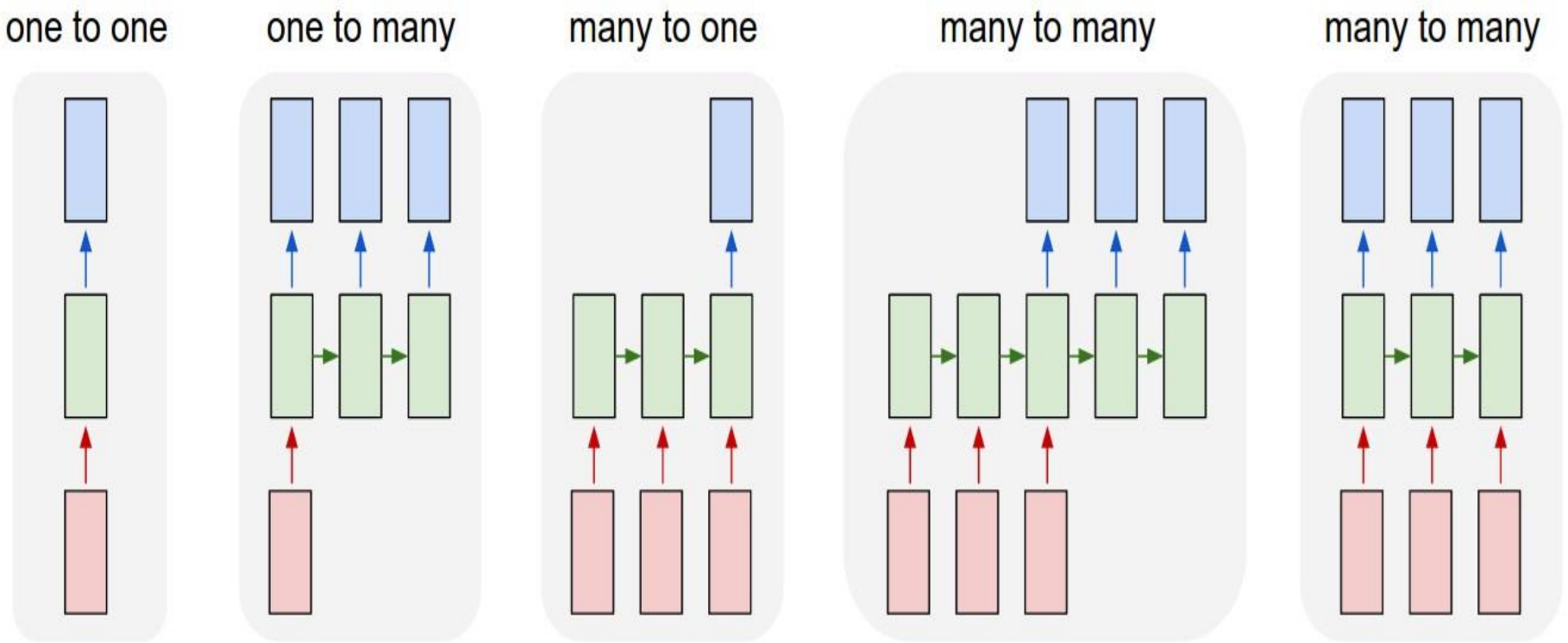
e.g. **Image Captioning**
image -> sequence of words



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



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



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

A RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

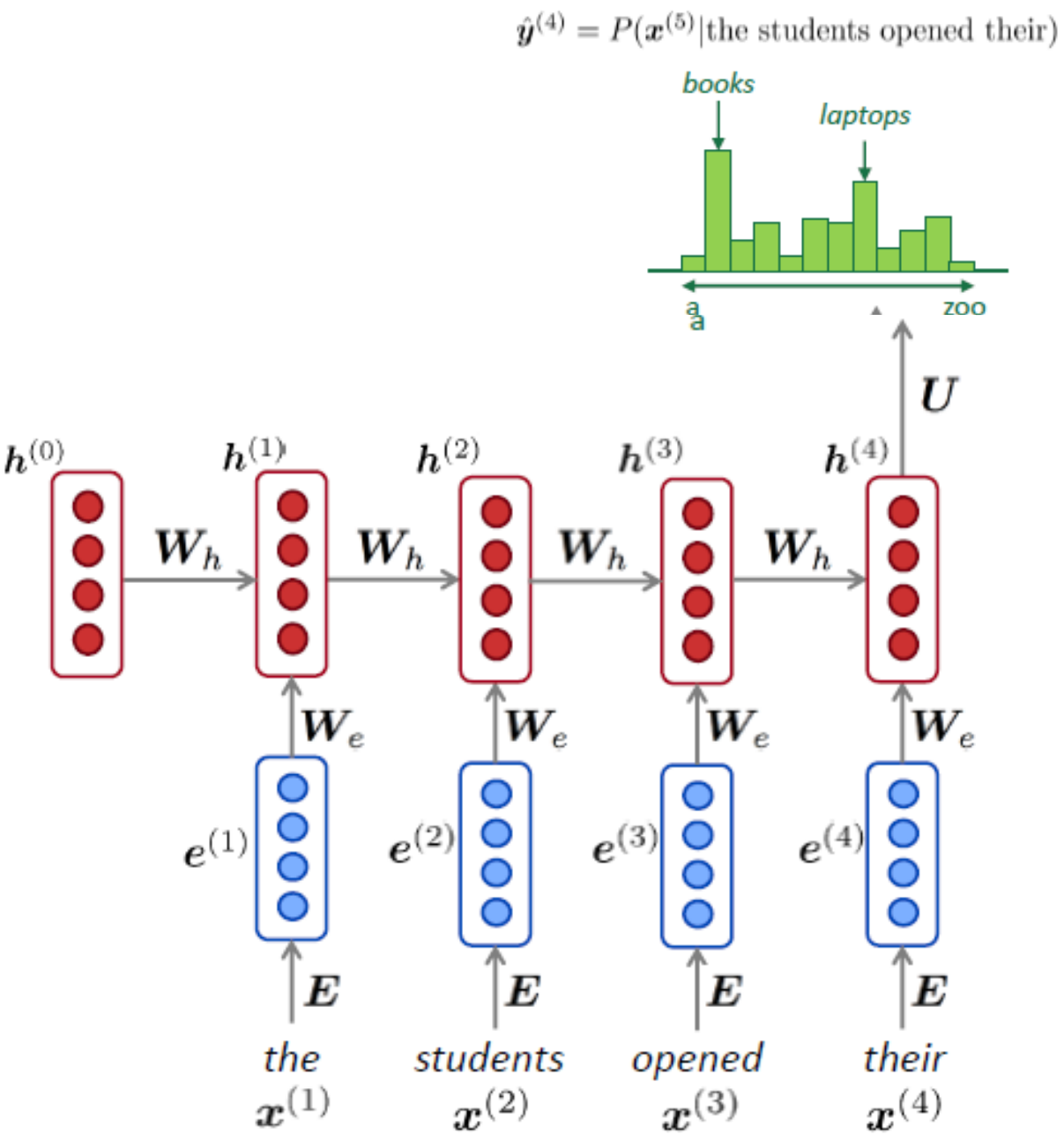
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



***Note**: this input sequence could be much longer, but this slide doesn't have space!*

A RNN Language Model

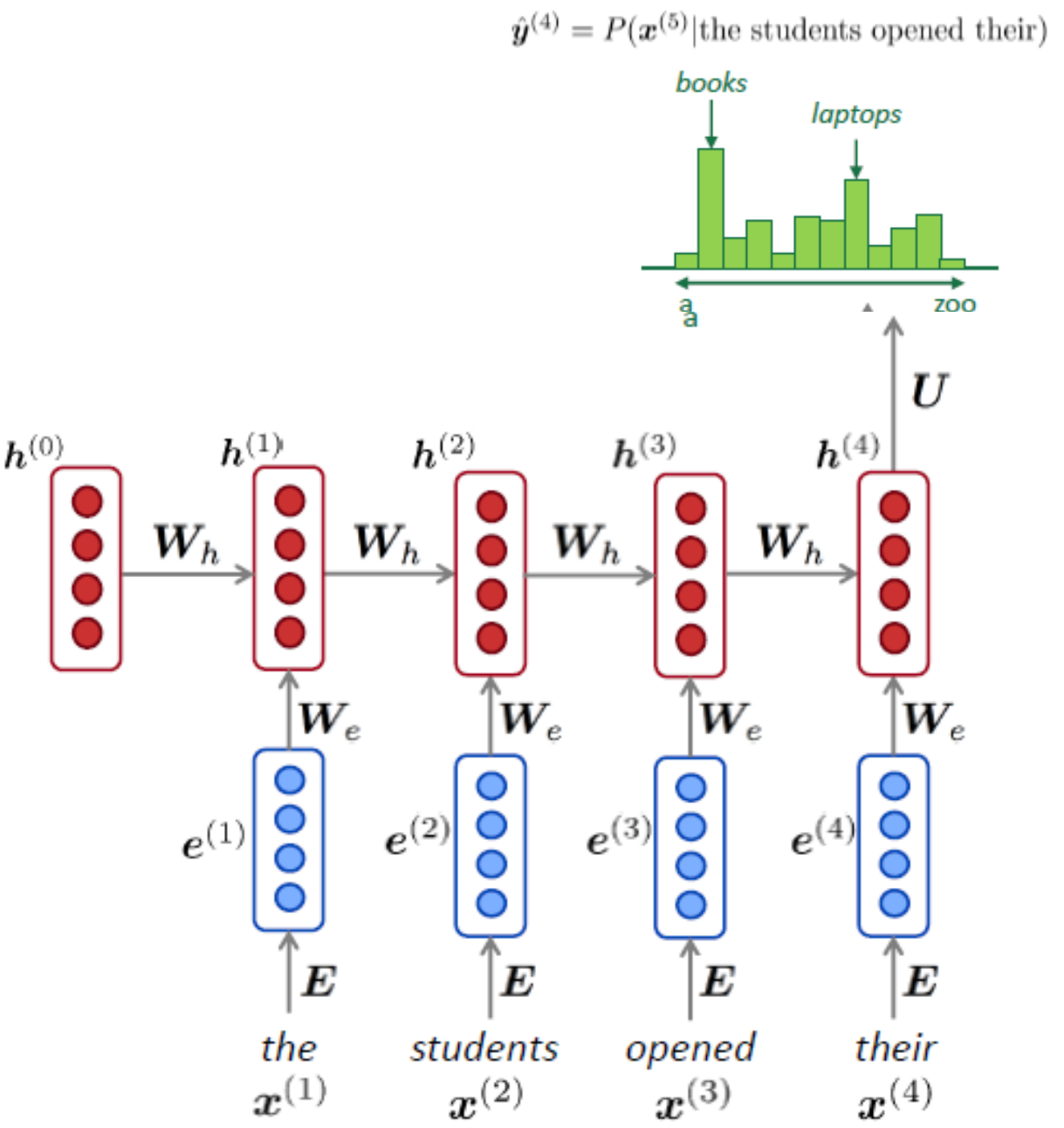
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there i symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course



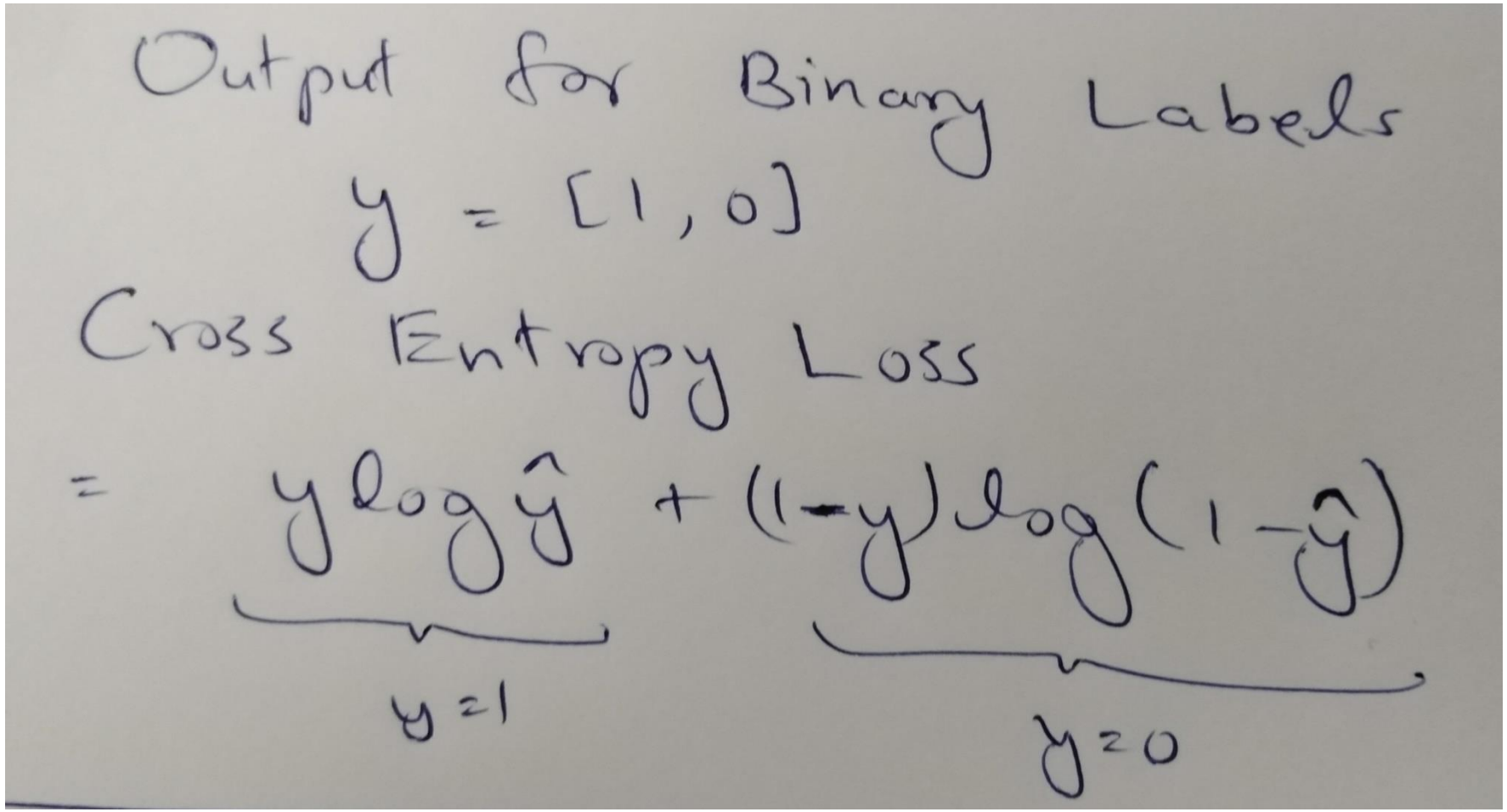
Task

- Draw RNN architecture diagram and write equations along with dimensions of all layers and weight matrices for following language model:
- input sequence of length 5 (lets say 5 words).
- Hidden layer units are 4.
- V = vocabulary = 10

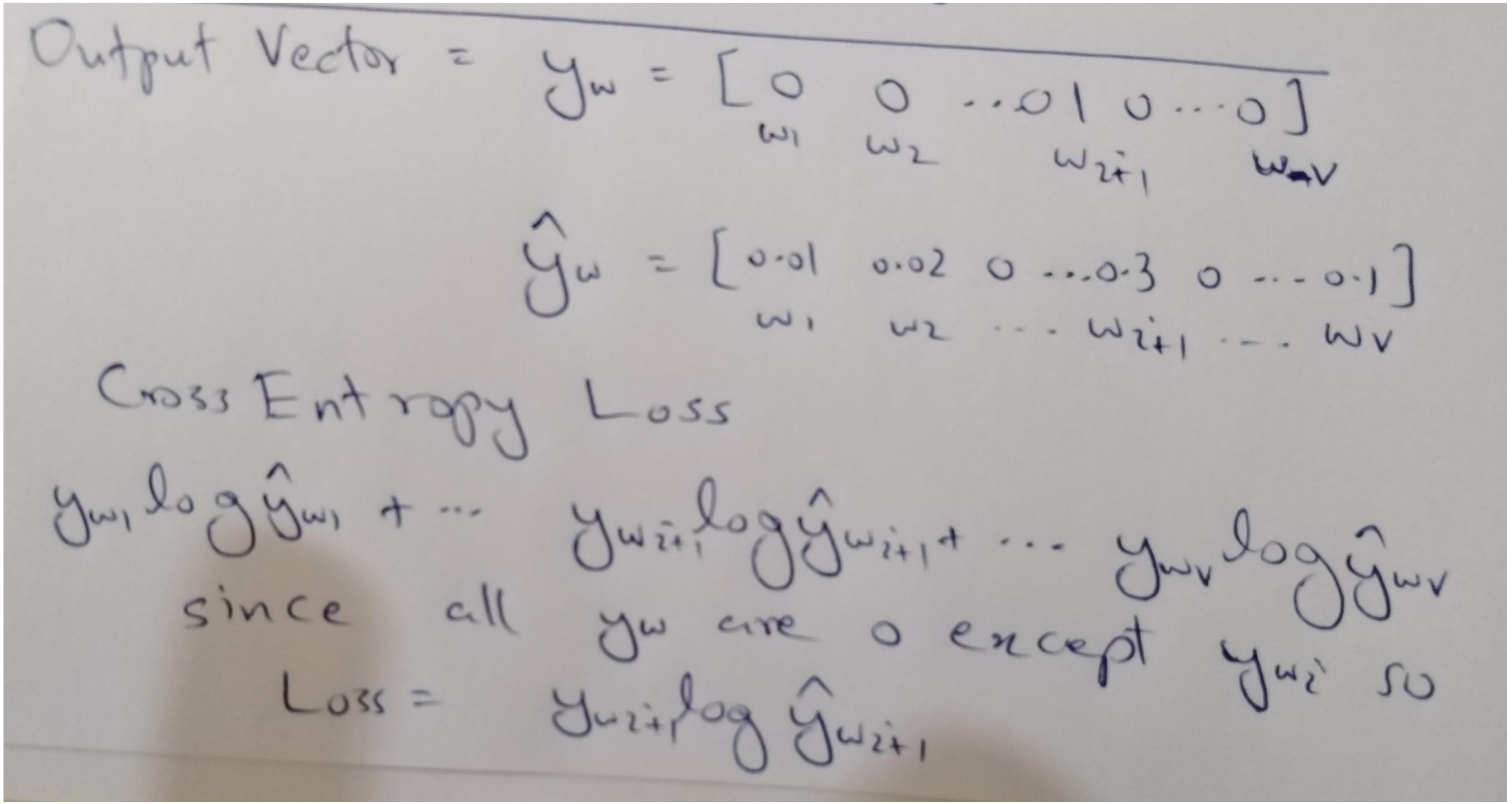
For reference revisit slides of lecture. Submit handwritten solution (snapshot) on Google classroom under assignment.

RNN for Language Modeling (Loss Function)

Cross Entropy Loss



Cross Entropy Loss for Language Modeling



Training a RNN Language Model (Loss Function)

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far

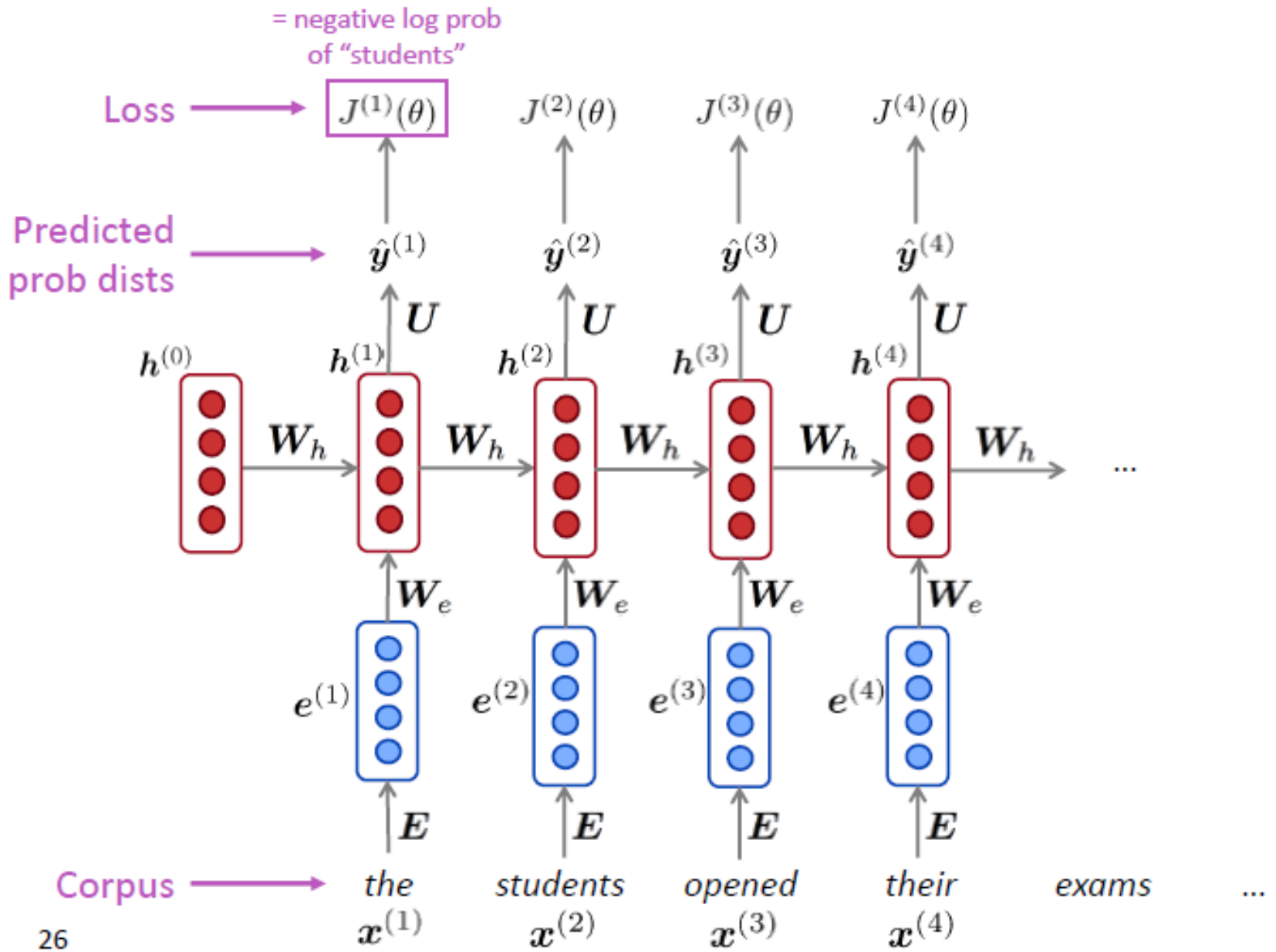
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

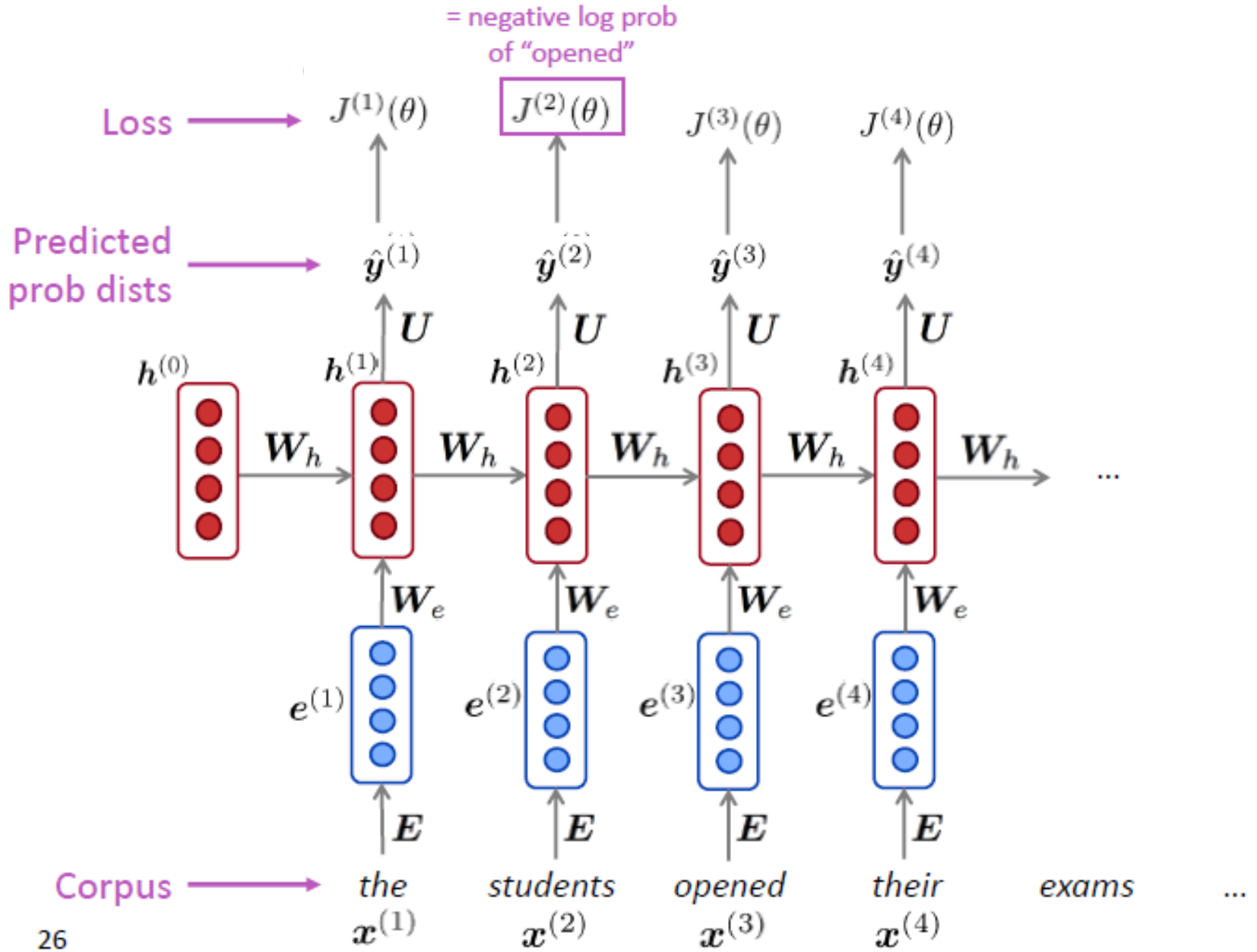
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

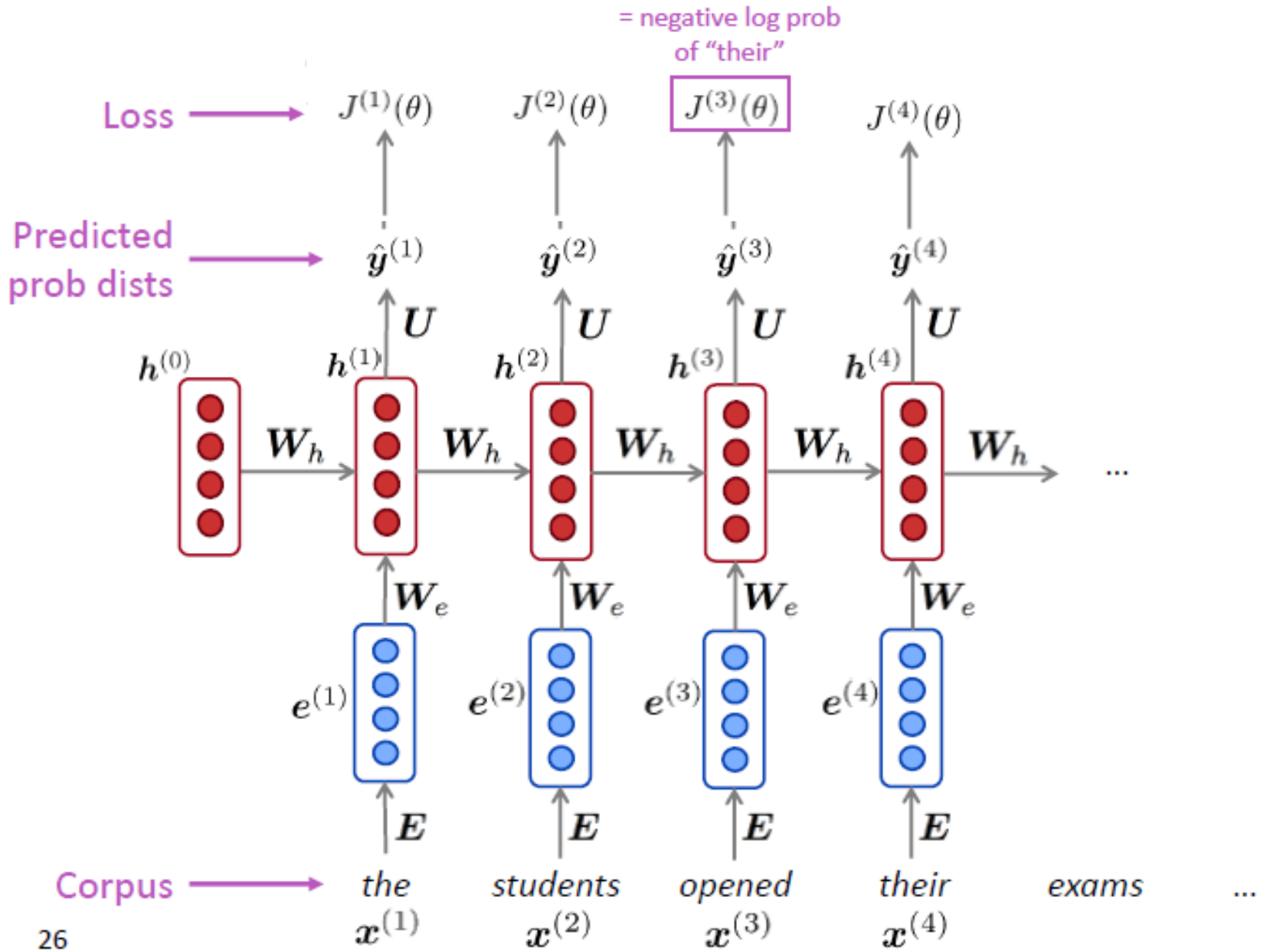
Training a RNN Language Model (Loss Function)



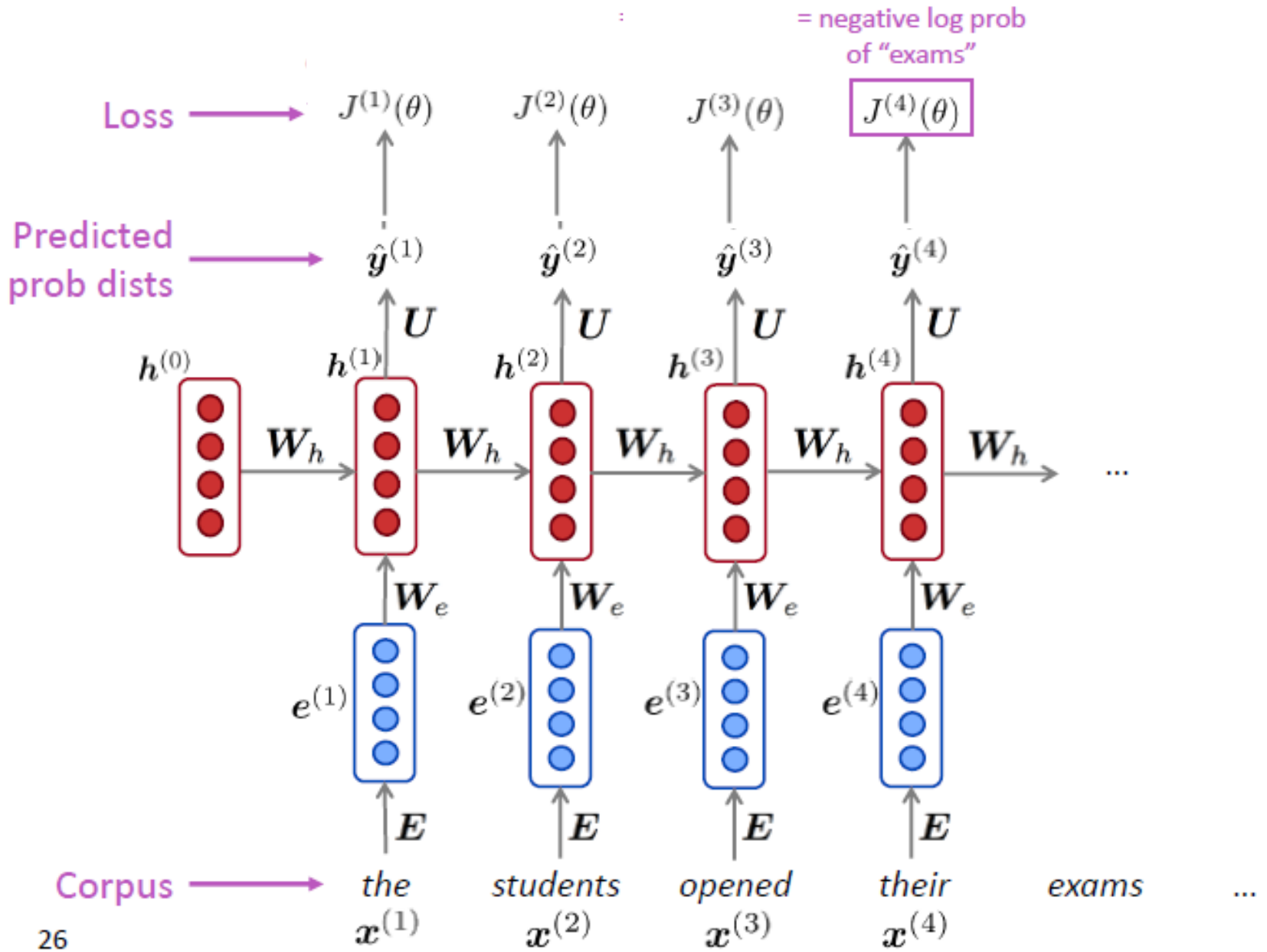
Training a RNN Language Model (Loss Function)



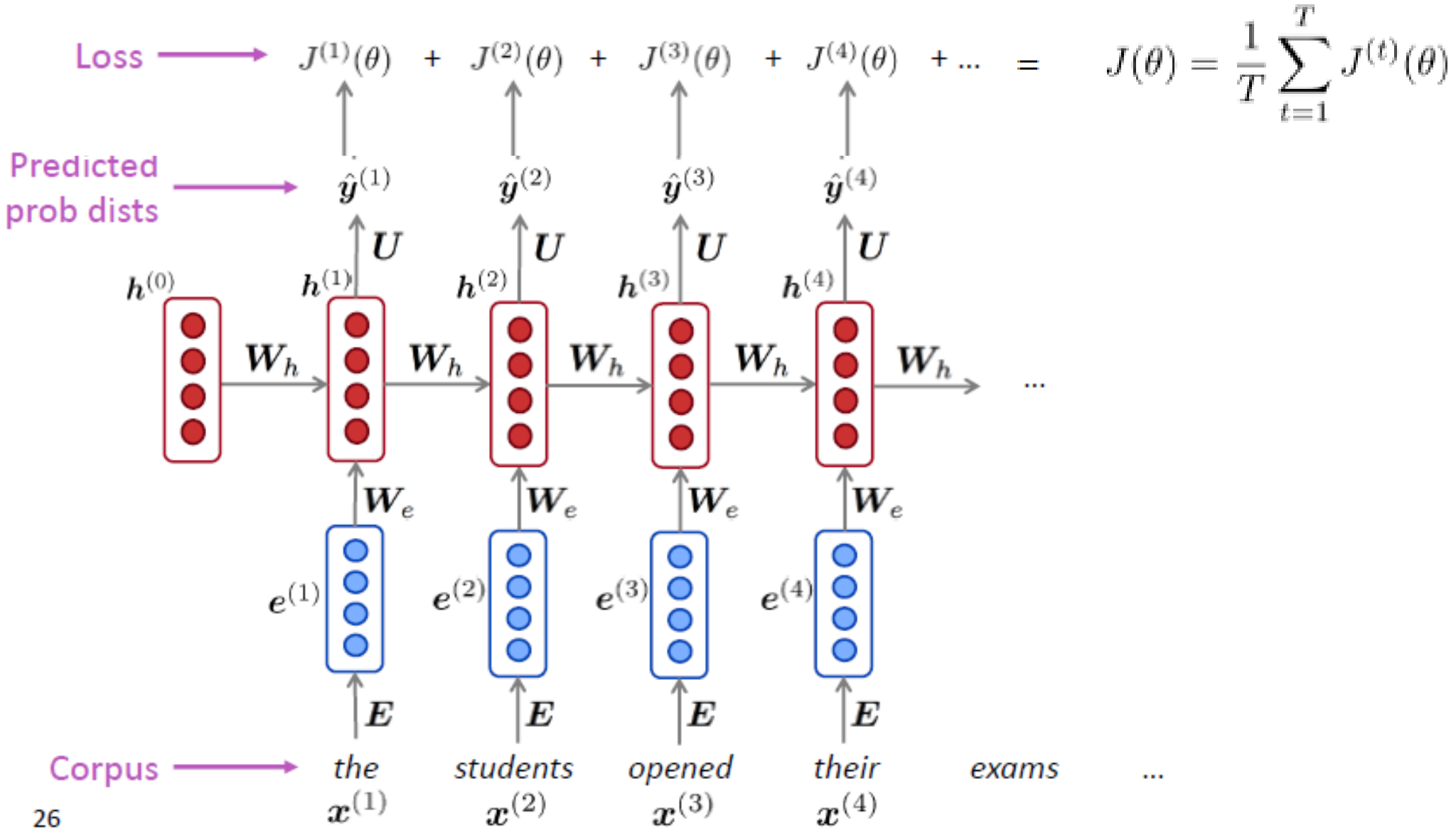
Training a RNN Language Model (Loss Function)



Training a RNN Language Model (Loss Function)



Training a RNN Language Model (Loss Function)



Training a RNN Language Model (Loss Function)

- However: Computing loss and gradients across entire corpus $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ is too expensive!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ as a sentence (or a document)

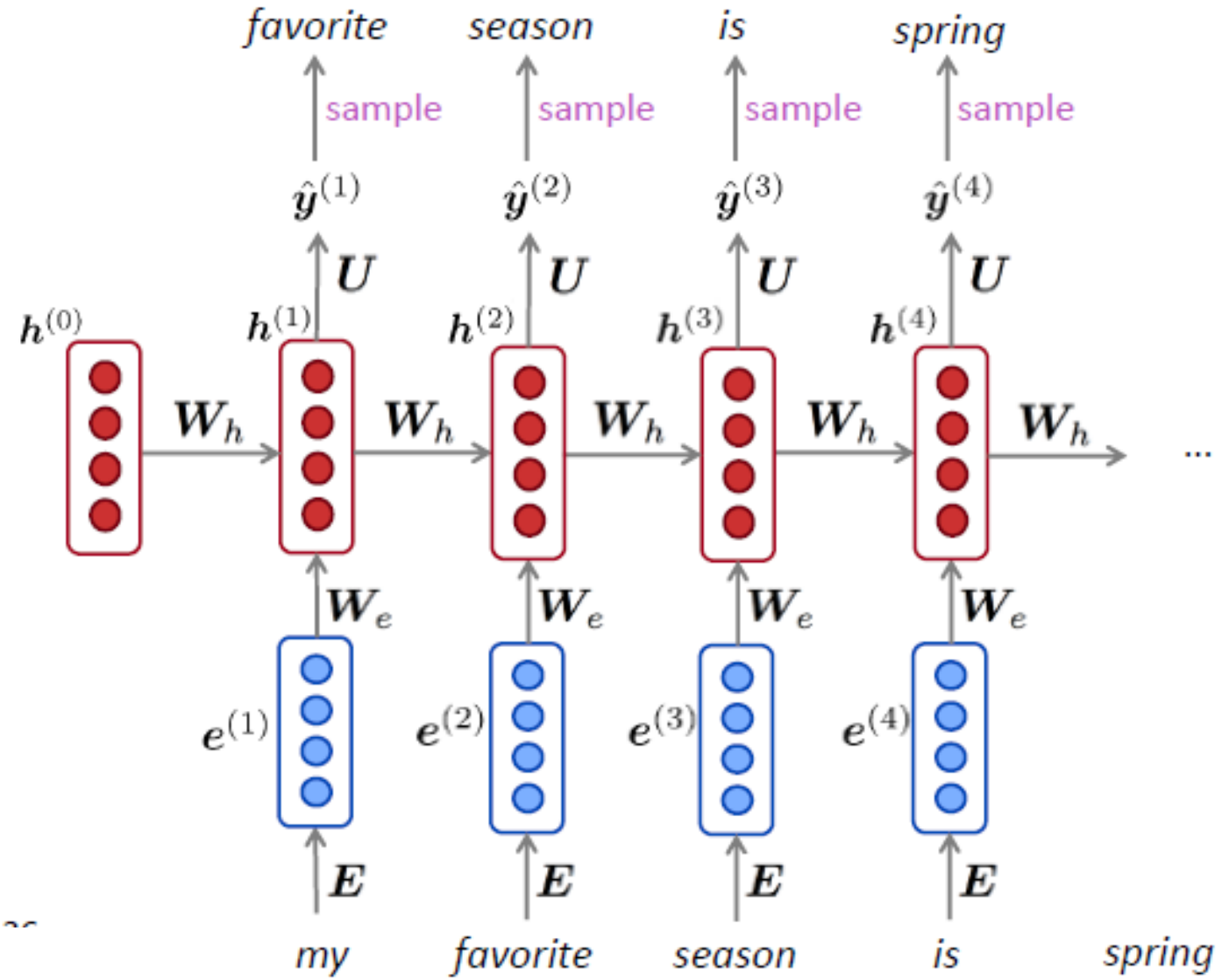
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

- Compute loss $J(\theta)$ for a sentence (actually a batch of sentences), compute gradients and update weights. Repeat.

Generating Text with RNN

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step's input.



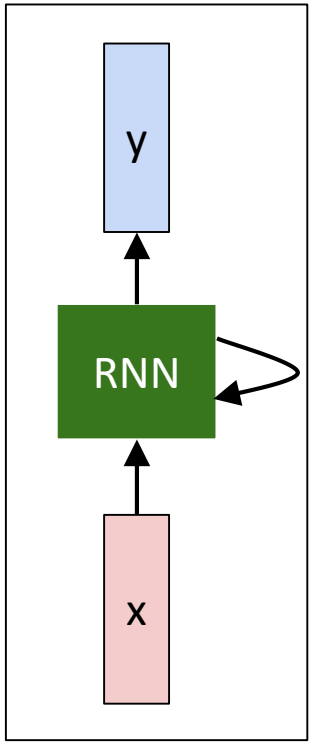
Sampling

Character	Probability
a	0.09
b	0.31
c	0.30
d	0.25
e	0.05

A Character based RNN Language Model

Vocabulary:
[h,e,l,o]

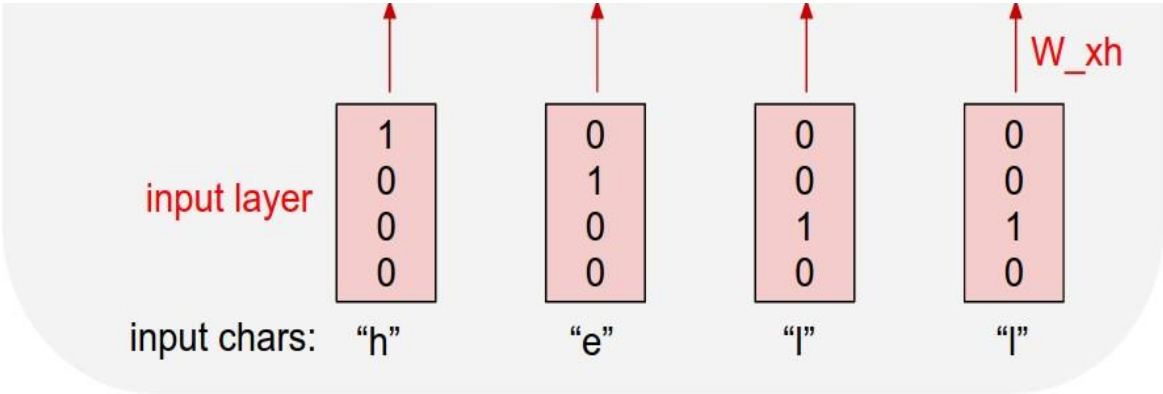
Example training sequence:
“hello”



A Character based RNN Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

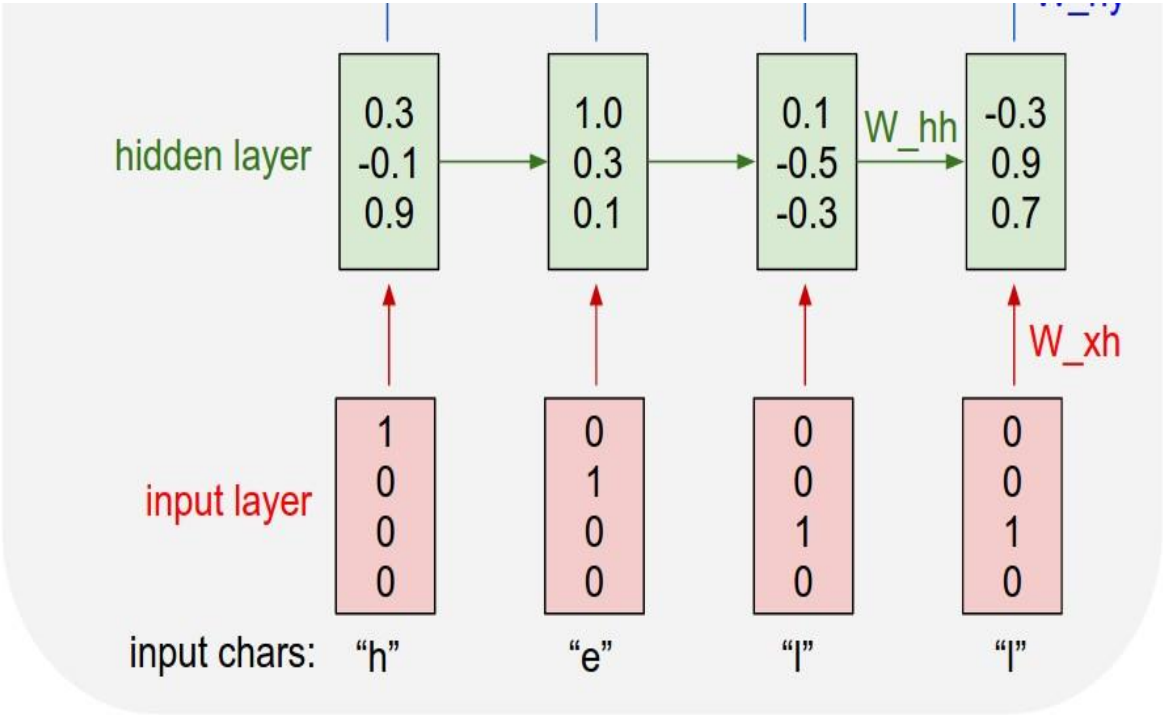


A Character based RNN Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

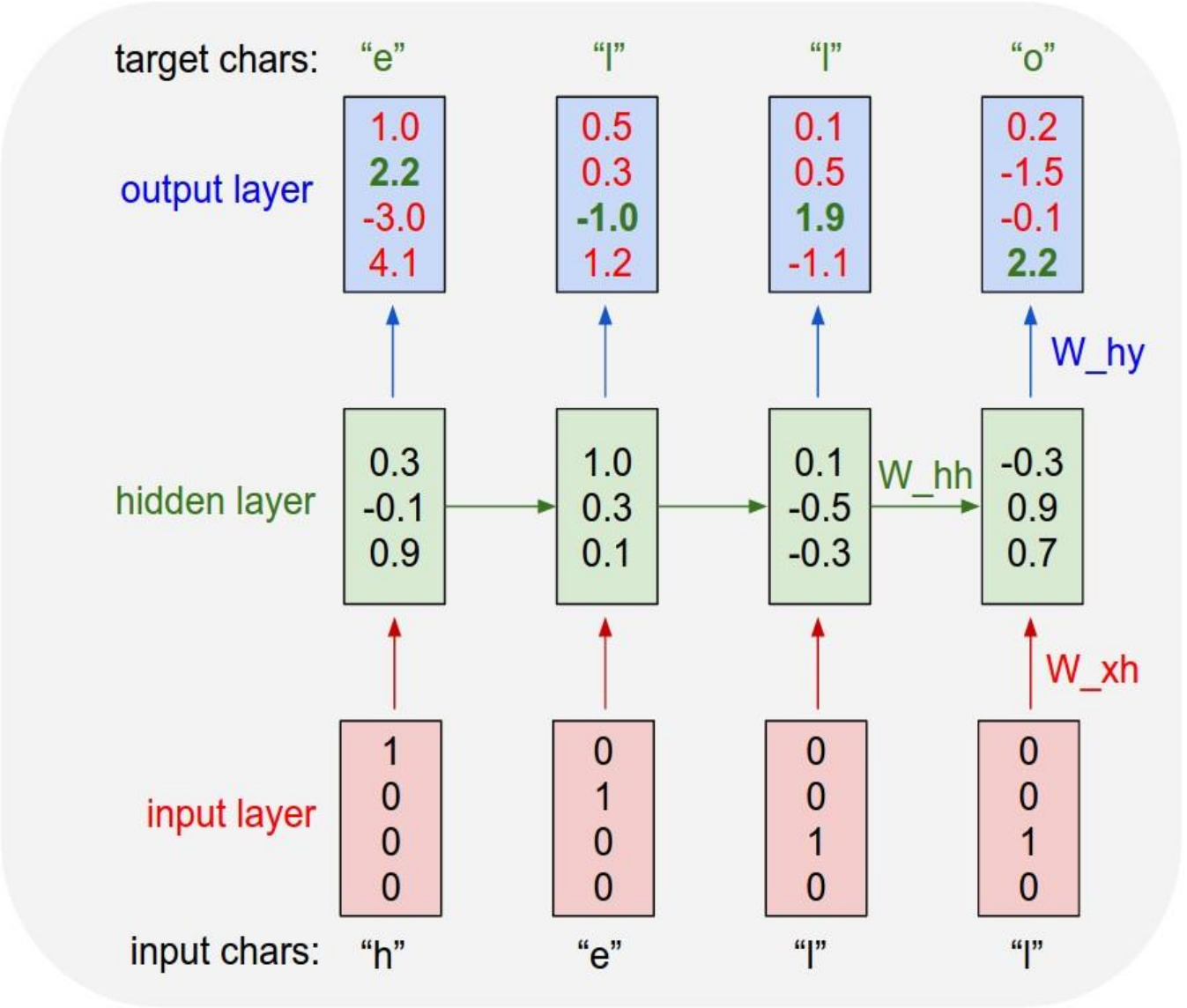
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



A Character based RNN Language Model

Vocabulary:
[h,e,l,o]

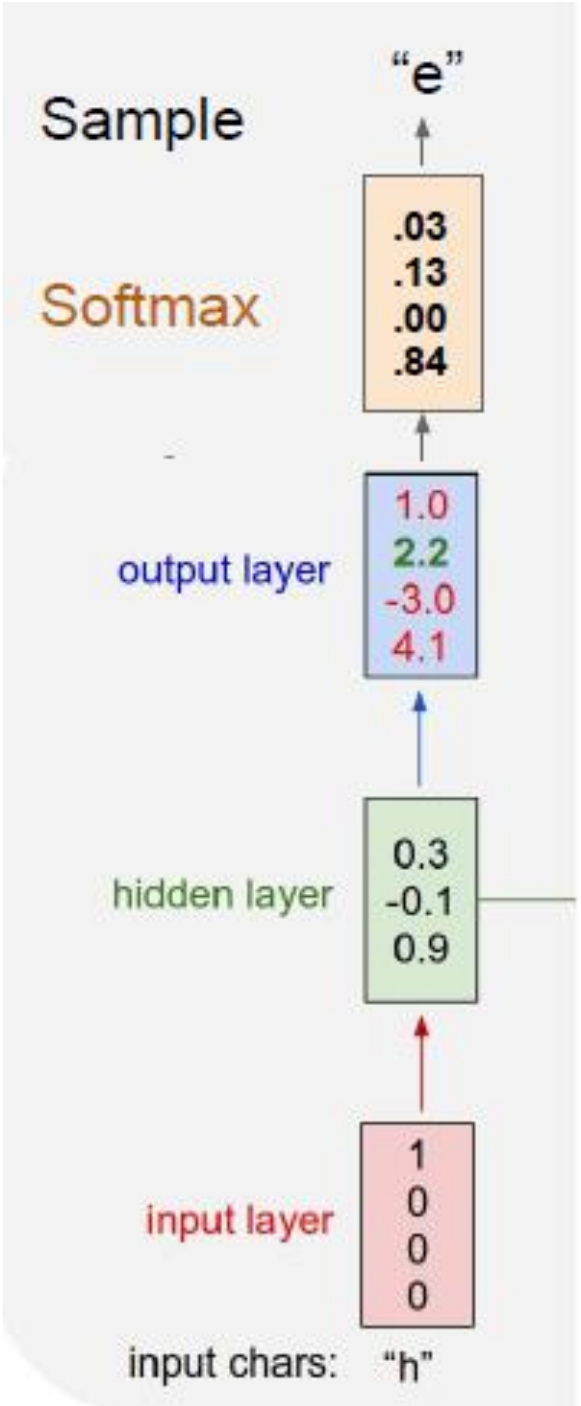
Example training sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

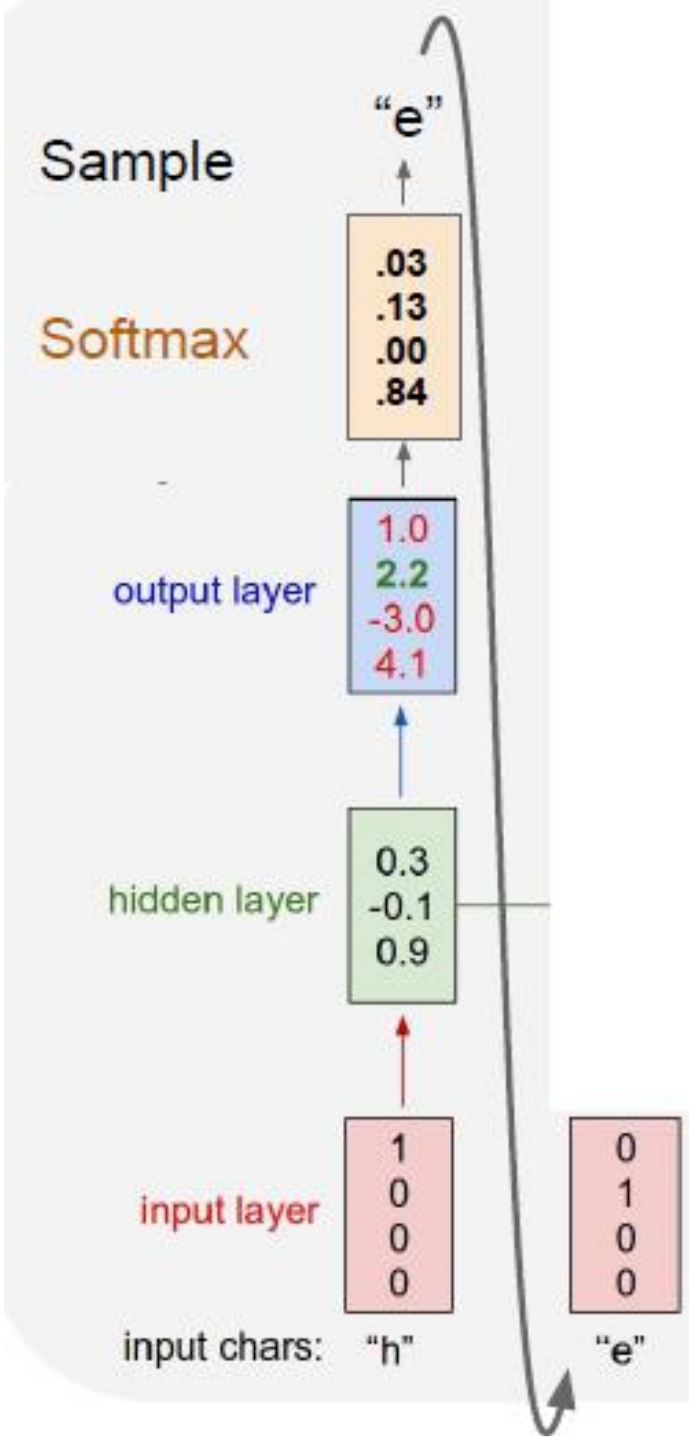
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

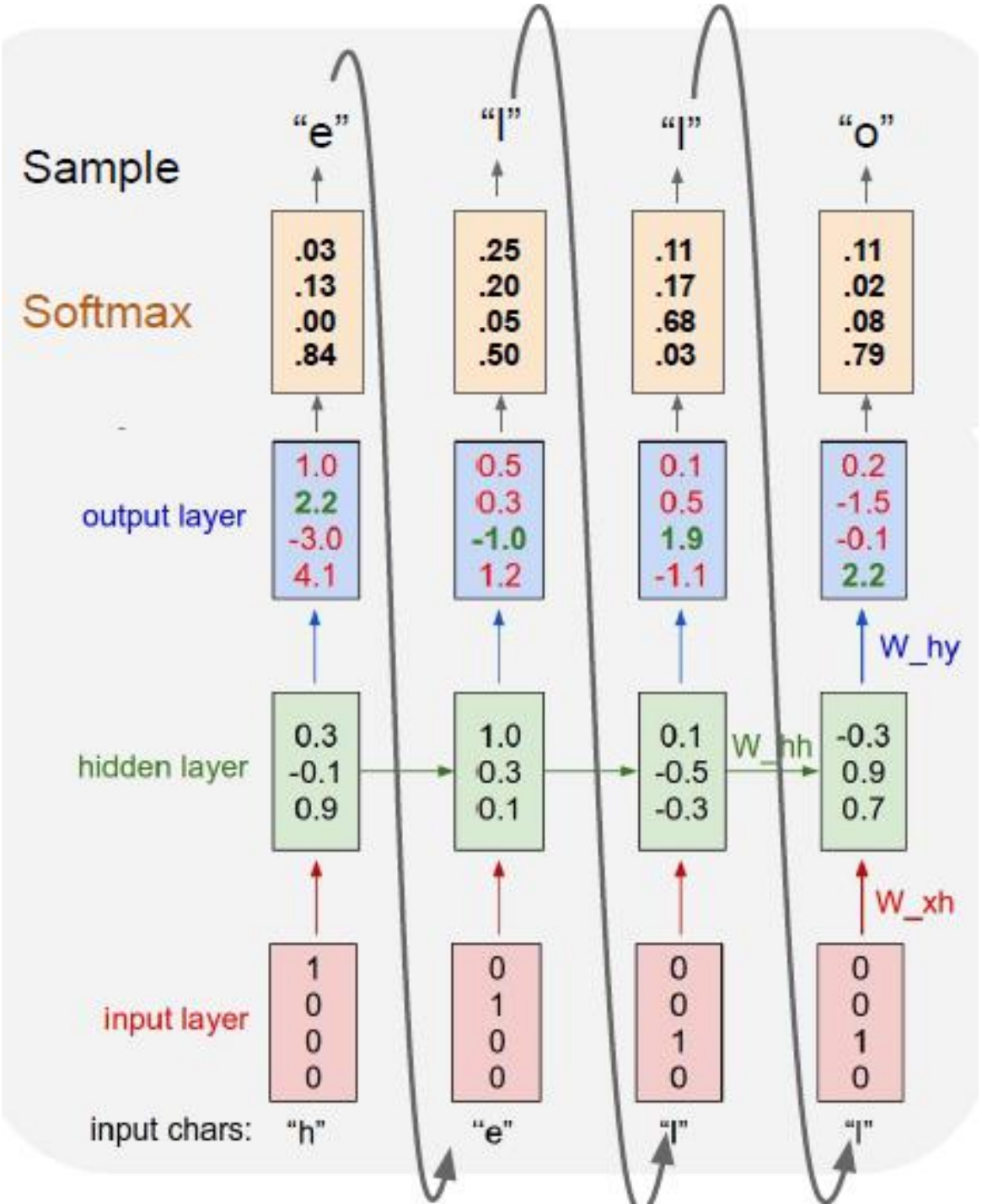
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



A Simple RNN code based on RNN equations is available at following link

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

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

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:

```
      Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
      Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

RNN Trained on some novel

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.

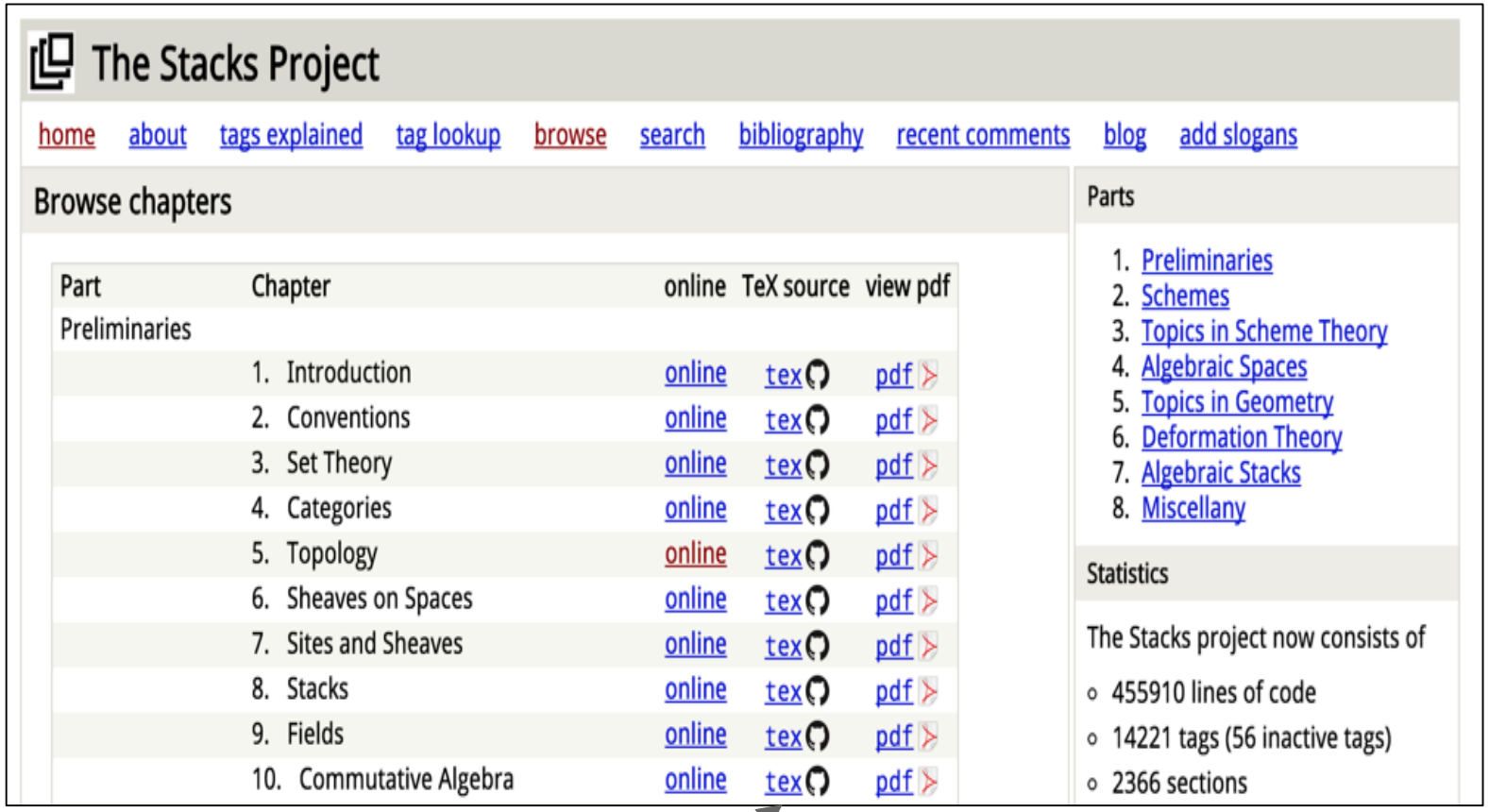
Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

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.



Latex source

Algebra Equations and proofs using RNN

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, 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 $U \to T$ is a separated algebraic space.

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

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

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ 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 $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

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

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

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, 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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ 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* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

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

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *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* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

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

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

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* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

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

Proof. We will use the property we see that $\mathfrak{p}$ is the mext 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. □

Algebra Equations and proofs using RNN

Proof. Omitted. □

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

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

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

.

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

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

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

Lemma 0.2. *This is an integer $\mathcal{Z}$ 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

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

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

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \mathrm{gor}_s & & \end{array} \qquad \begin{array}{ccc} = \alpha' & \longrightarrow & \\ \updownarrow & & \\ = \alpha' & \longrightarrow & \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \\ \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G}) \end{array}$$

$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets}$$

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 $X = \mathrm{Spec}(R)$ 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. □

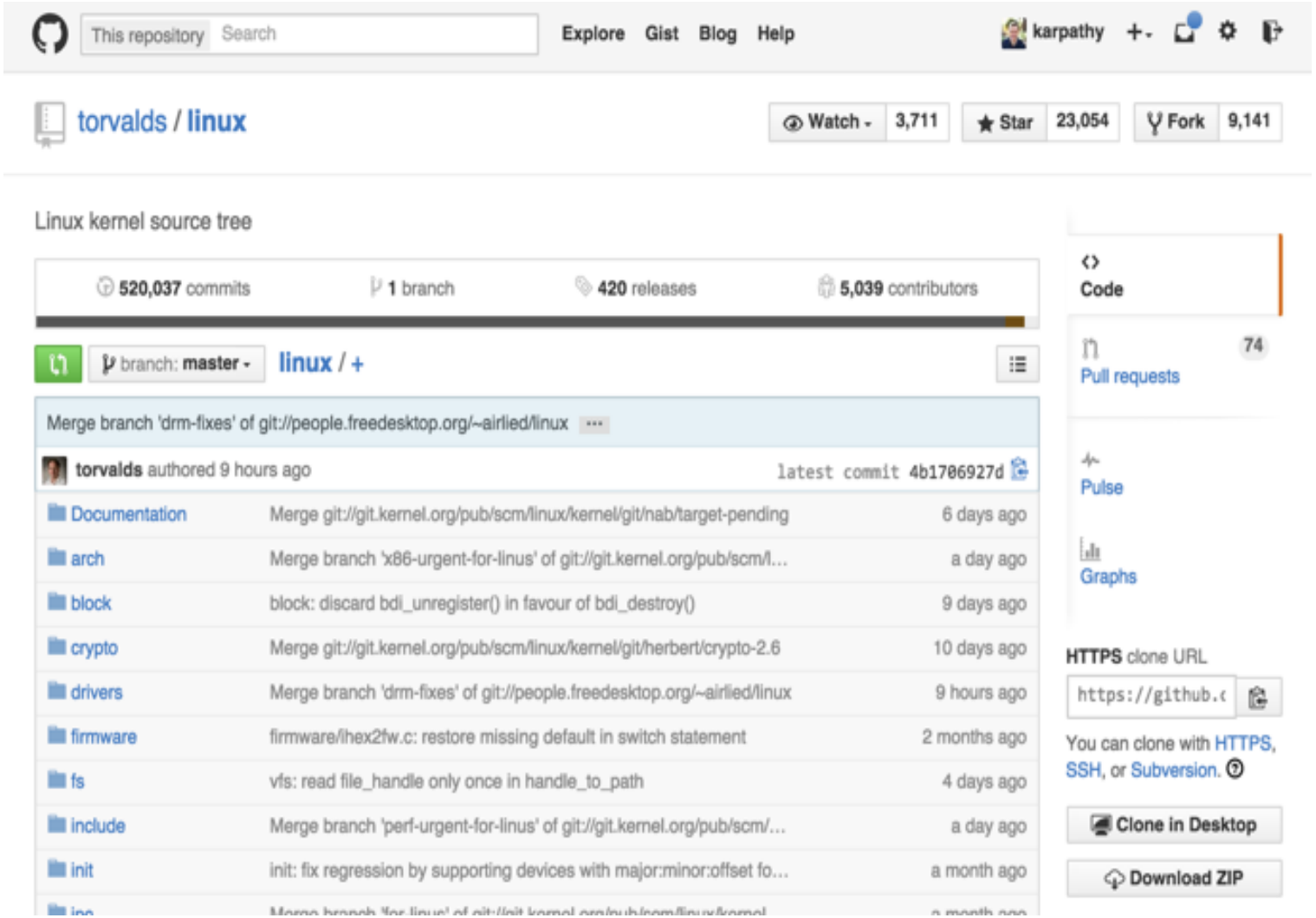
Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

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.

C Code generation using RNN



```
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 & 0x00000000ffffffff8) & 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>
```

Generated C code

```
#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;
}
```

Generated C code

RNNs have greatly improved perplexity

n-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

↓ Perplexity improves (lower is better)

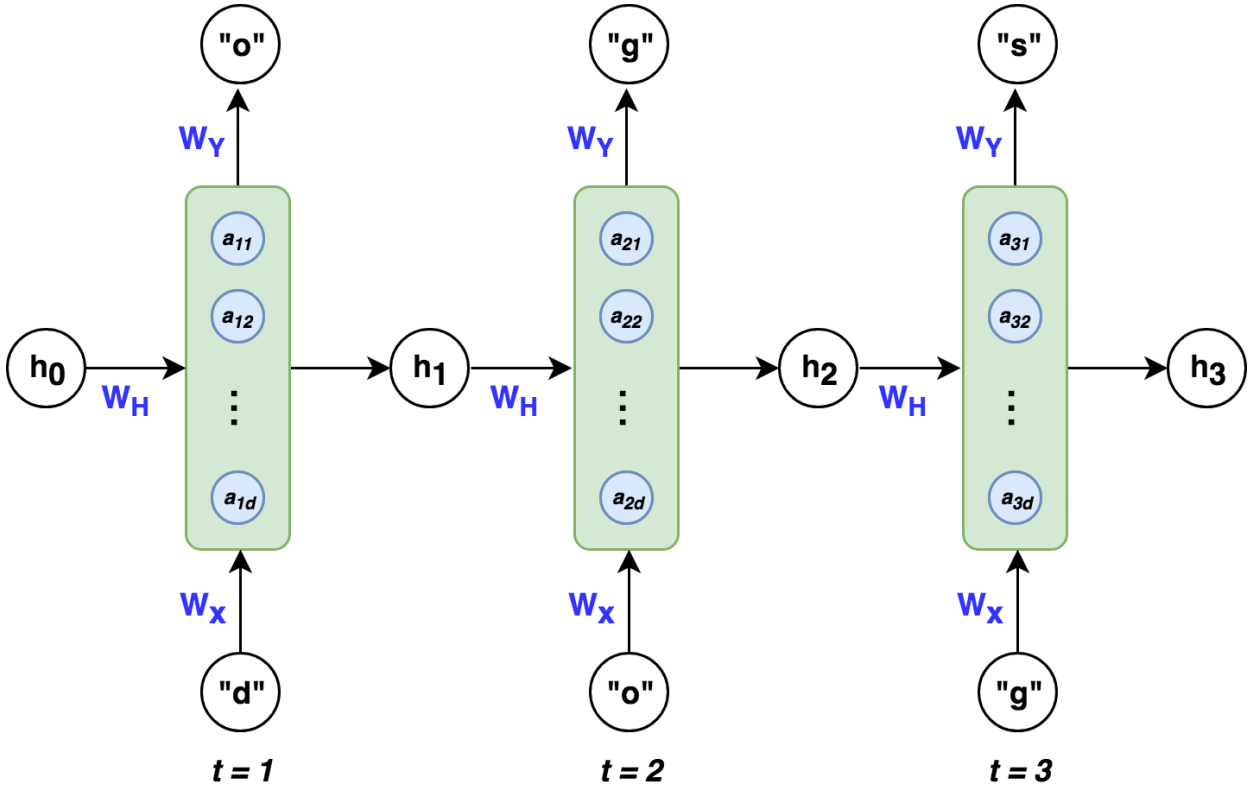
Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

RNN Back Propagation

RNN Forward Pass



$$a_t = W_H h_{t-1} + W_X X_t$$ Hidden Nodes

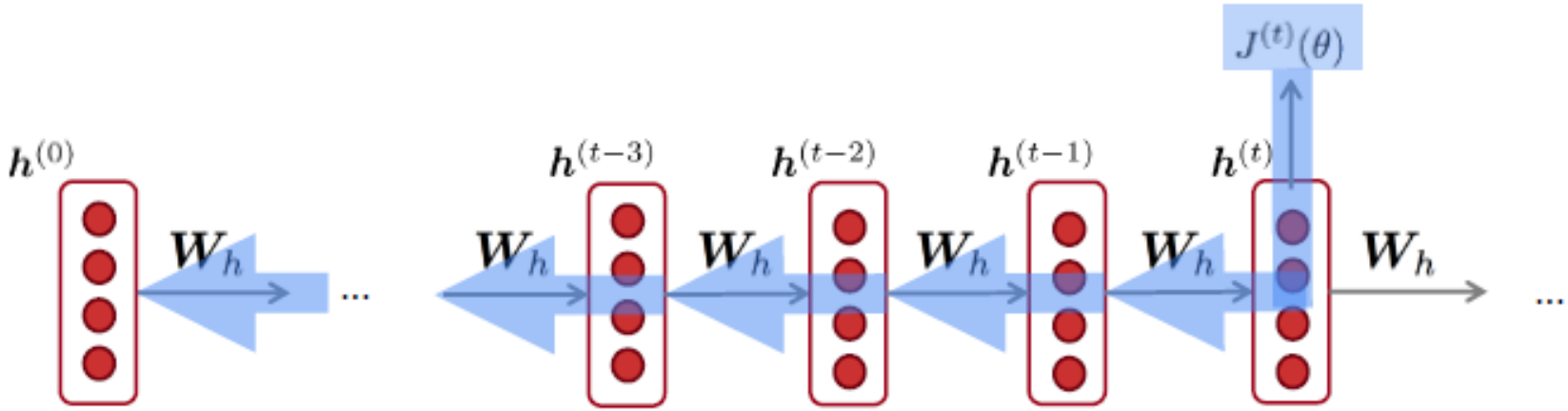
Activation Function

Output From Hidden State → $h_t = tanh(a_t)$

Hidden State

Prediction at time t → $y_t = softmax(W_Y h_t)$

Backpropagation for RNNs



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} \right|_{(i)}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i*=*t*,...,0, summing gradients as you go.
This algorithm is called
"backpropagation through time"

Backpropagation for RNNs

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

RNN can model long term dependency

The brown and black dog, which was playing with the cat, was a german shepherd.

(x_2) (x_4) (x_5) (x_{14}) (x_{15})

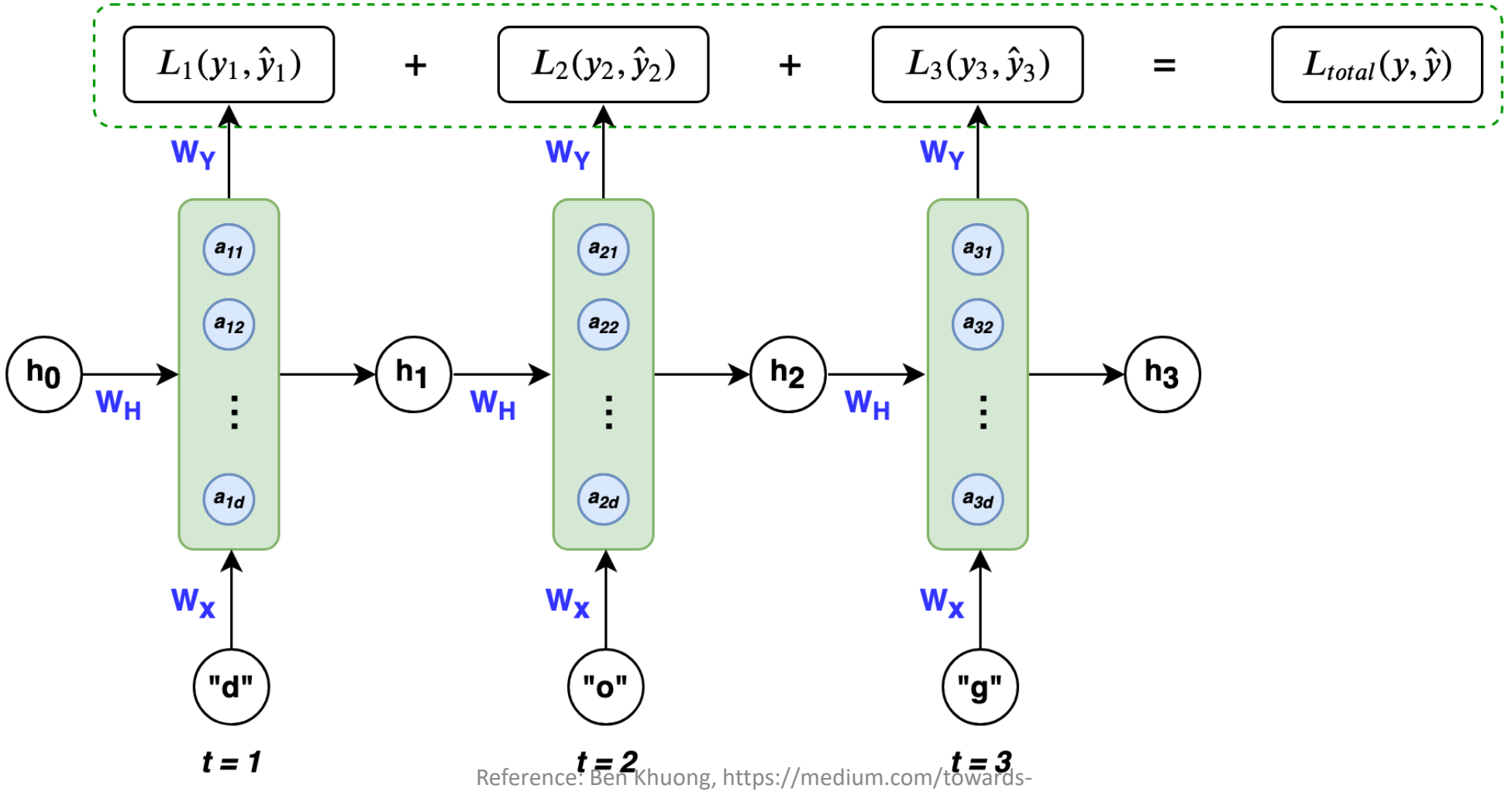
Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Backpropagation through time

1. Initialize weight matrices *Wx, Wy, Wh* randomly
2. Forward propagation to compute predictions
3. Compute the loss
4. Backpropagation to compute gradients
5. Update weights based on gradients
6. Repeat steps 2–5

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Total Loss



Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Weight Update

$$W_i := W_i - \eta \frac{\partial L_{total}(y, \hat{y})}{\partial W_i}$$

where i = x, y, and h as a shorthand for the 3 weight matrices

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

The main idea is chain rule and to account for the loss at each time step

Function dependencies with respect to W_Y

$$L_t = -y_t log(\hat{y}_t) \rightarrow \hat{y}_t = softmax(z_t) \rightarrow z_t = W_Y h_t$$

Chain rule with respect to W_Y

$$\frac{\partial L_t}{\partial W_Y} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial W_Y}$$

$$L_{total}(y, \hat{y}) = -\sum_{t=1}^{n} y_t log(\hat{y}_t)$$

Gradient for W_Y

$$\frac{\partial L_{total}}{\partial W_Y} = \frac{\partial L_1}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial W_Y} + \frac{\partial L_2}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial z_2} \frac{\partial z_2}{\partial W_Y} + \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial W_Y} = \sum_{t=1}^{n} \frac{\partial L_t}{\partial W_Y}$$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Function Dependencies with respect to W_X

$$L_t = -y_t log(\hat{y}_t) \rightarrow \hat{y}_t = softmax(z_t) \rightarrow z_t = W_Y h_t \rightarrow h_t = tanh(W_H h_{t-1} + W_X X_t) \quad (1)$$

Chain rule with respect to W_x

$\frac{\partial L_t}{\partial W_x} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial W_x}$ but note, that within h_t, h_{t-1} also contains W_x, thus we need to recurisvely chain rule h_{t-1} until we reach h_0

$$\frac{\partial L_t}{\partial W_x} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial W_x} + \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_x} + \ldots + \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-n}} \frac{\partial h_{t-n}}{\partial W_x} = \sum_{k=0}^{n} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_x} \quad (3)$$

Gradient for W_X

$$\frac{\partial L_{total}}{\partial W_X} = \sum_{t=1}^{n} \sum_{k=0}^{n} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_x} \quad (4)$$

Reference: Ben Khuong, https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

Task 1

- Write partial derivative equations using chain rule for W_h. They will be similar to equations for W_x on previous slide. Submit on Google classroom

Reading

- Chapter 9, Speech and Language Processing. Daniel Jurafsky & James H. Martin. Third edition

  https://web.stanford.edu/~jurafsky/slp3/9.pdf