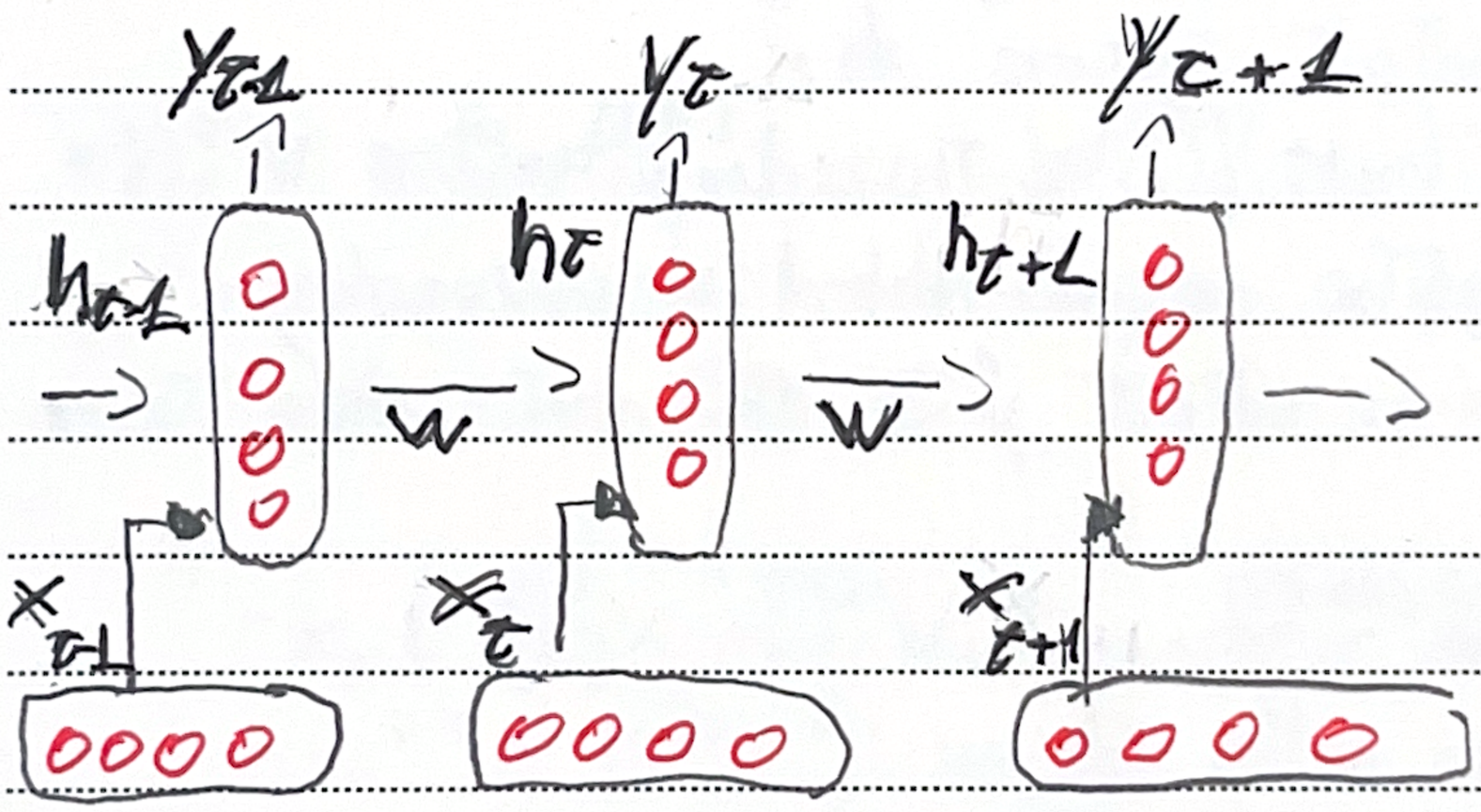
Date:

Recurrent Neural Network (RNN)

- Special Type of NN adapted to work for Time series data or data that works in sequences.

- Capable of conditioning the model on all previous inputs.

A RNN with Three Time Steps

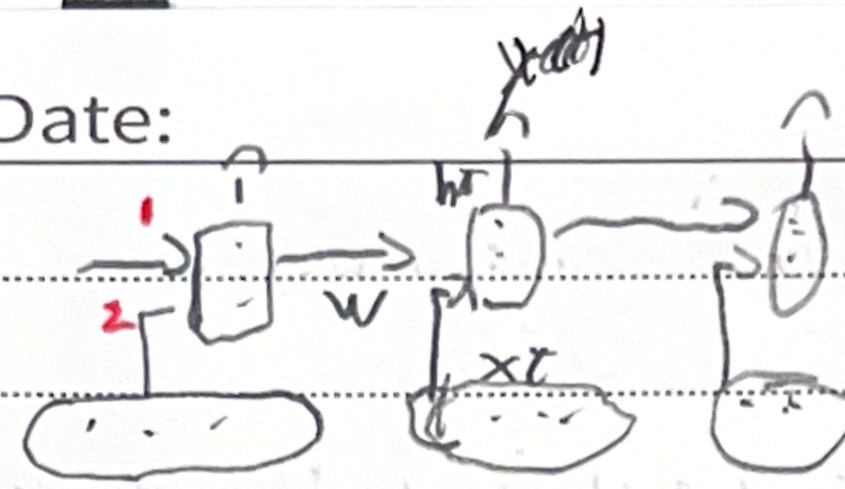


- each vertical rectangular box is a hidden layer at time-Step t.

- Each layer holds a number of neurons, each of which Performs a liner matrix operation on its inputs Followed by a non-liner operation (eg. Tanh or ReLU)

- At each time step there are 2 inputs.

Date:



- At each time step, we have 2 inputs To the hidden layer

1 - the output of the previous layer h_{t-1}

2 - The input of that timestep x_t

3 - The former input is multiplied by a weight matrix $W^{(hh)}$
- And the latter by a weight matrix $W^{(hx)}$ to produce output features h_t

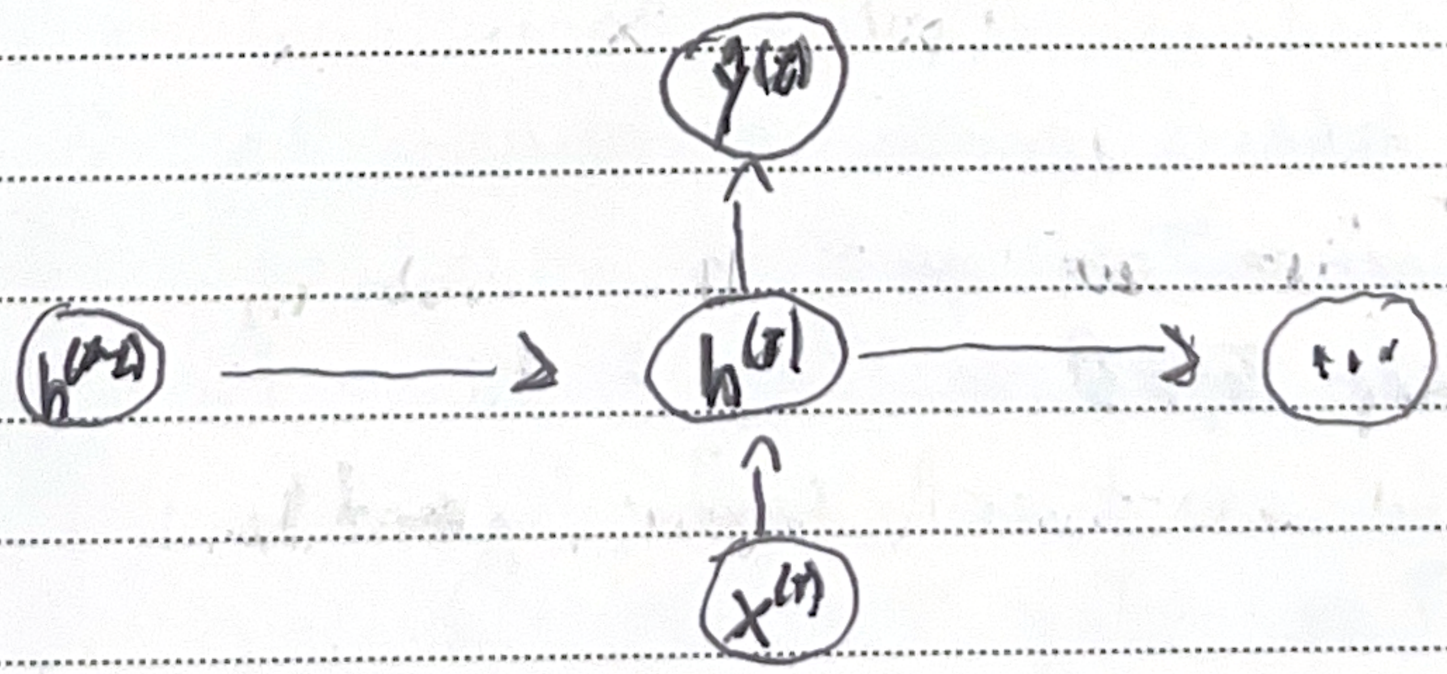
4. - These are multiplied with a weight matrix $W^{(s)}$ and through a softmax over the vocabulary

- This produces output $\hat{y}$ of the next word.

3. $h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]})$

4. $\hat{y}_t = \text{softmax}(W^{(s)} h_t)$

- Inputs and outputs of each decoder illustrated here:



Date:

• The same weights $W^{(hh)}$ and $W^{(hx)}$ are applied repeatedly at each time step:

- number of parameters the model has to learn is less
- independent of the length of the input sequence.
- i.e. defeating the curse of dimensionality.

Details of each procedure in the network

• $X_1, \ldots, X_{t-1}, X_t, X_{t+1}, \ldots X_T$: The word vectors corresponding to a corpus with T words

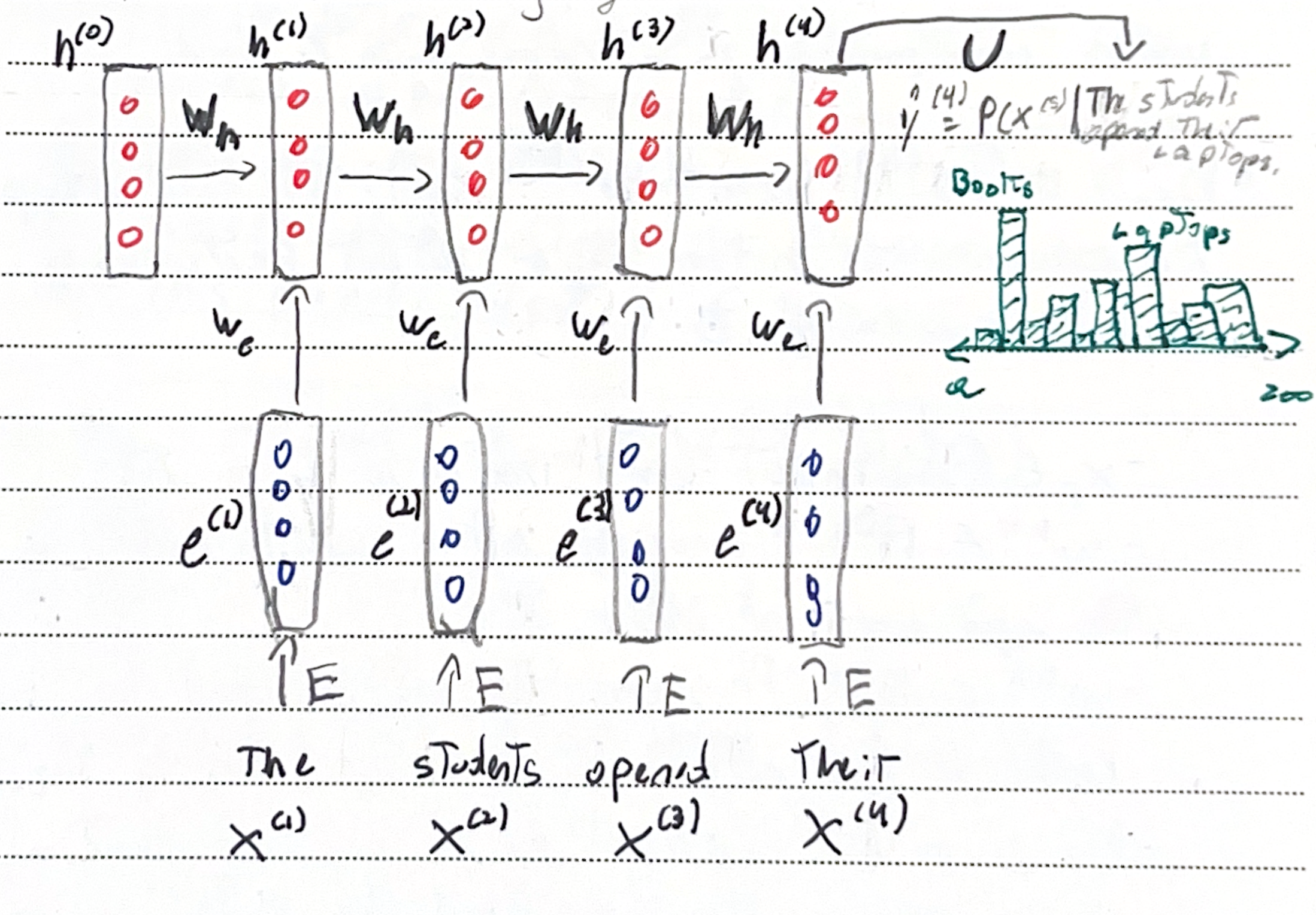
• $h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} X_t)$: the relationship to compute the hidden layer output features at each time step t

- $X_t \in \mathbb{R}^d$: input word vector at time t.
- $W^{hx} \in \mathbb{R}^{D_h \times d}$: weights matrix used to condition the input word of vector, X_t
- $W^{hh} \in \mathbb{R}^{D_h \times D_h}$: weights matrix used to condition the output of the previous time-step, h_{t-1}
- $h_{t-1} \in \mathbb{R}^{D_h}$: output of the non-linear function of the previous time-step, $t-1$. $h_0 \in \mathbb{R}^{D_h}$ is an initialization vector for the hidden layer at time-step $t = 0$.
- $\sigma()$: The non-linearity function (sigmoid here).

Date:

- $\hat{y}_t = \text{Softmax}(W^{(s)} h_t)$: The output probability distribution over the vocabulary at each time step t. Essentially, $\hat{y}_t$ is the next predicted word given the document context score so far (i.e. h_{t-1}) and the last observed word vector $x^{(t)}$. Here, $W^{(s)} \in \mathbb{R}^{|V| \times Dh}$ and $\hat{y} \in \mathbb{R}^{|V|}$ where $|V|$ is the vocabulary.

Example of RNN Language Model



here, the equivalent of $W^{(hh)}$ is W_h, $W^{(ht)}$ is W_e and $W^{(s)}$ is U. E converts words inputs into $x^{(t)}$ to word embeddings $e^{(t)}$. The final softmax over the vocabulary shows us the probability of various options for token $x^{(5)}$, conditioned on all previous tokens. The input could be much longer than 4-5 tokens.

Date:

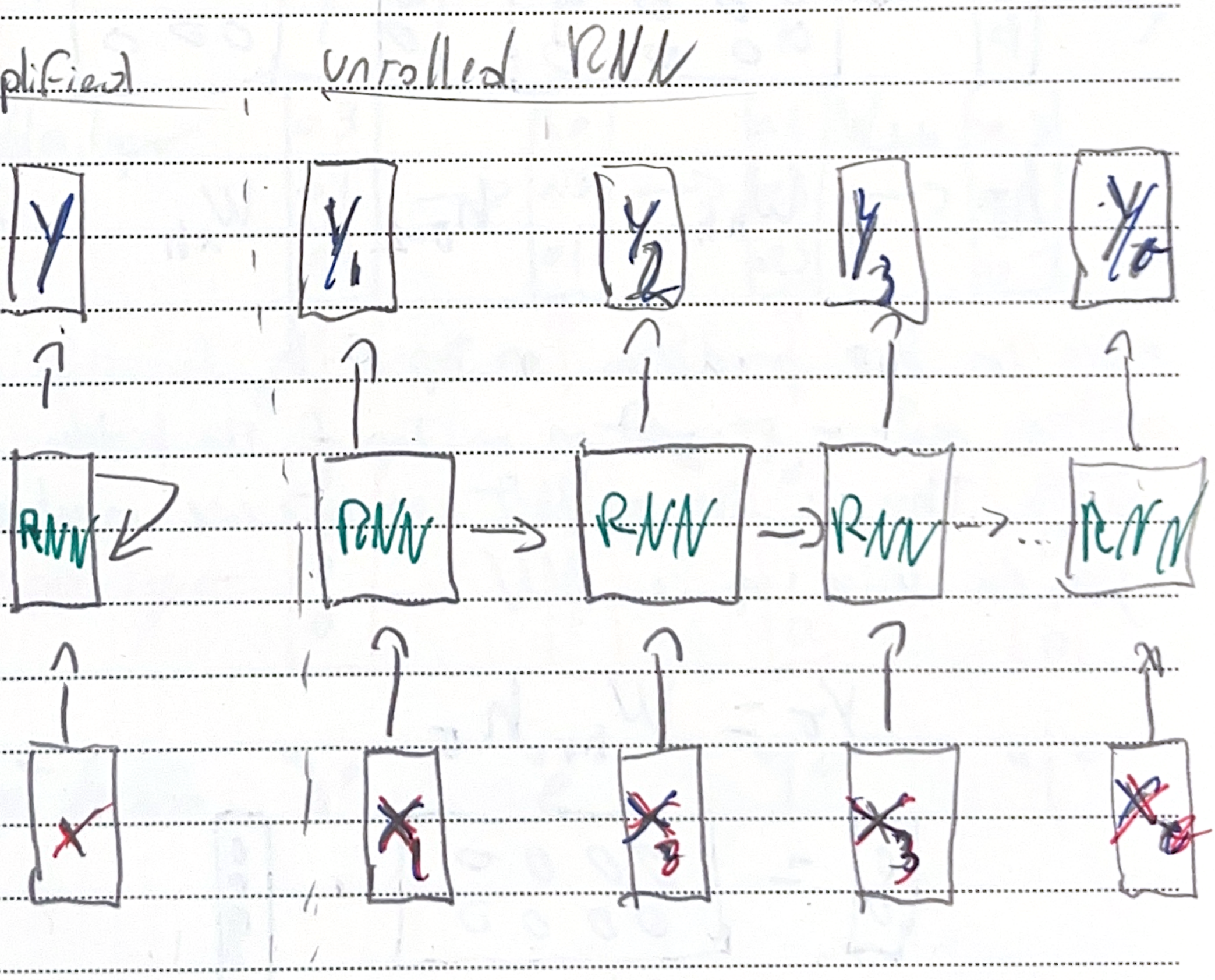
RNN loss and perplexity

- the loss function used in RNNs is often the cross entrophy error.
- Sum of entire vocabulary at time step t

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

END To END Example.

Simplified



unrolled RNN

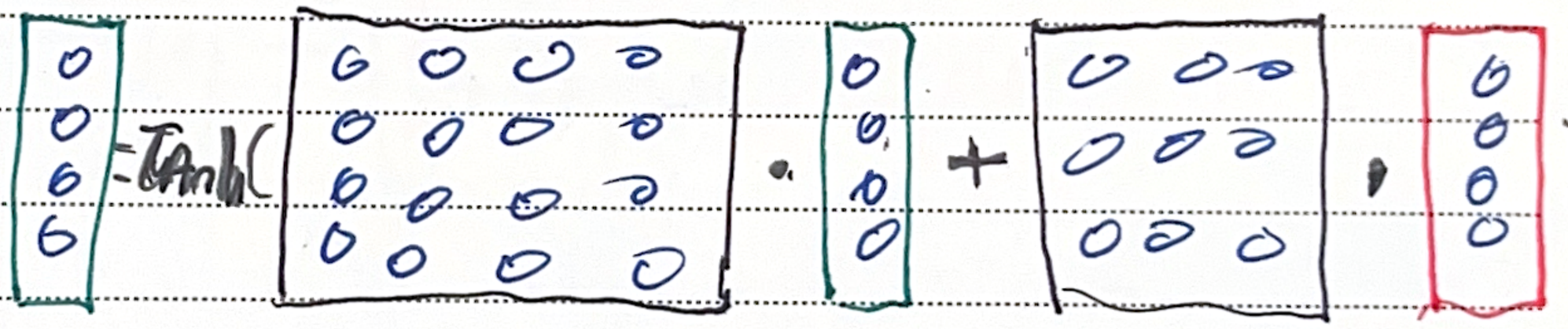
Date: Example RNN high level overview.

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

recurrent formula

- Simplest RNN (vanilla RNN), single hidden state h where we use a recurrent formula that tells us how we should update our hidden state h.

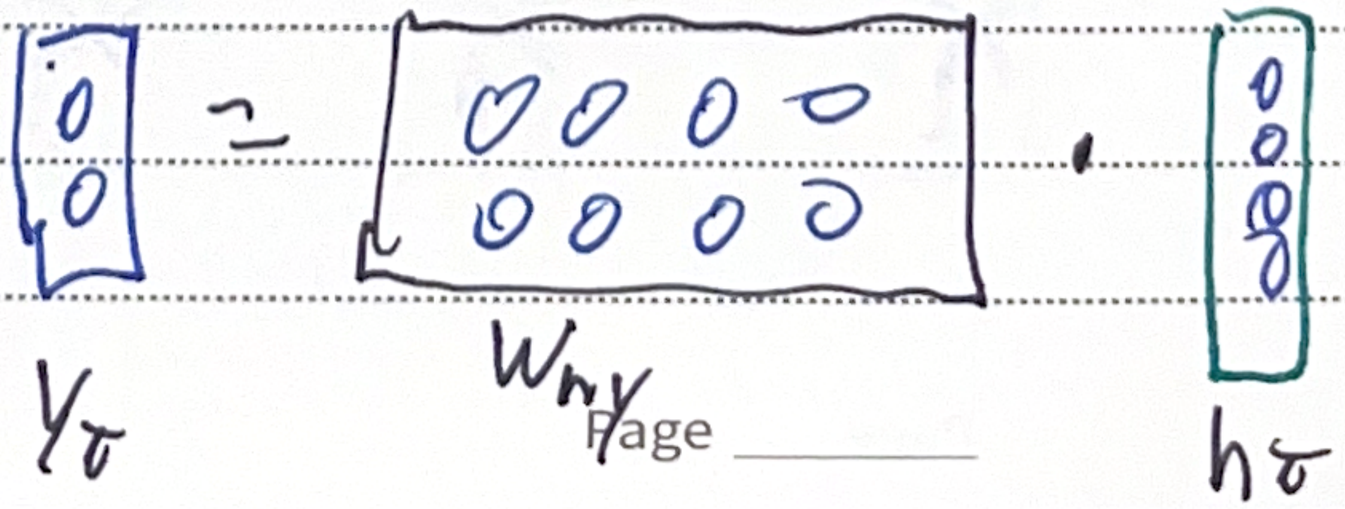
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$



h_t W_{hh} h_{t-1} W_{xh} x_t

- We can have prediction on top of h_t by using yet another matrix projection on top of the hidden state. This is the simplest complete case in which you can wire up a NN

$$y_t = W_{hy} h_t$$



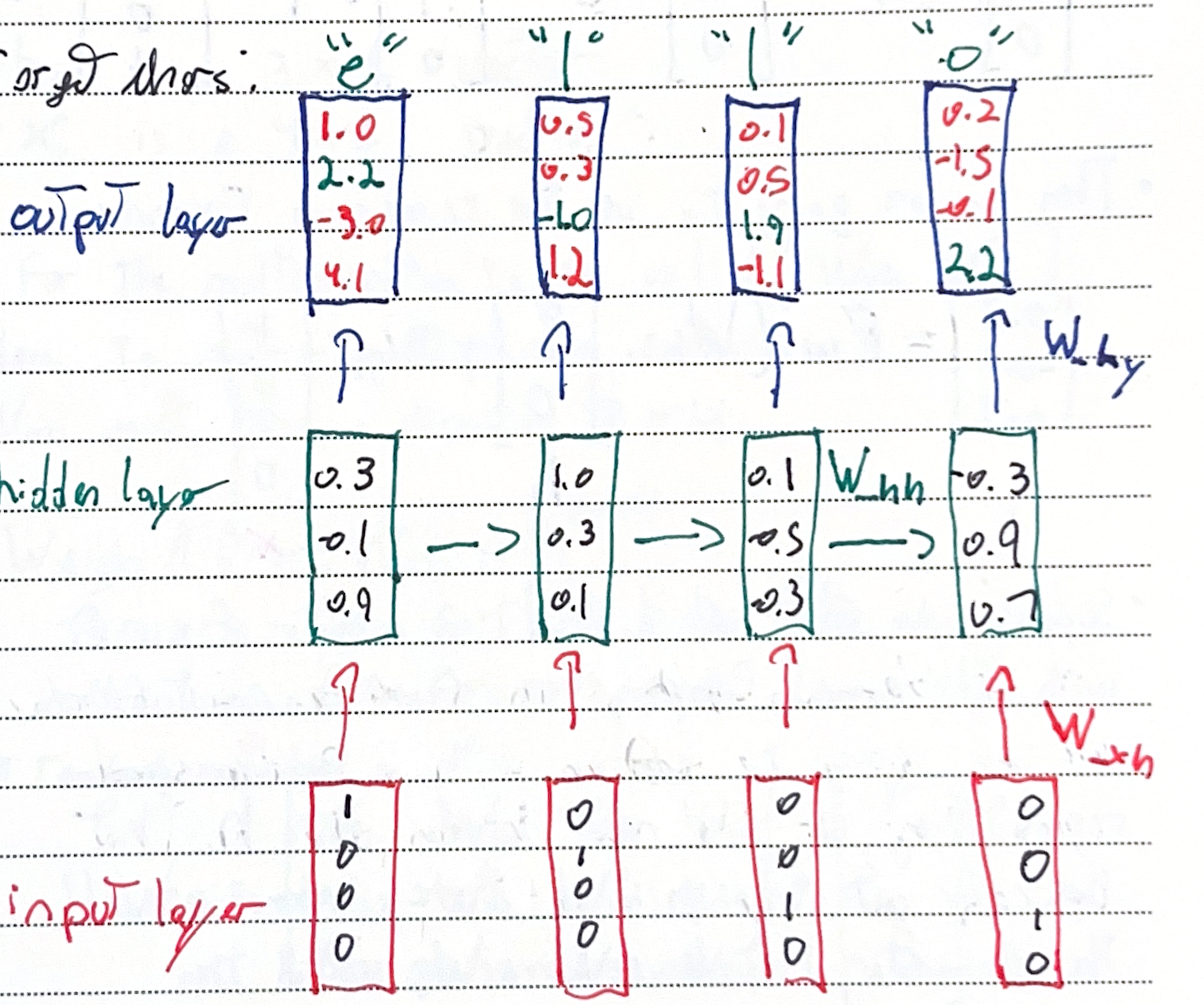
y_t W_{hy} h_t

RNN example: character-level language model

Date:

- A training sequence of just one string "hello"
and we have a vocabulary $V \in \{"h", "e", "l", "o"\}$
of 4 characters in the entire dataset.

- We are going to try to get an RNN to learn to predict
the next character in the sequence of this training set.



Date:

• As shown in the last image, we will feed our characters one time into an RNN. First "h", then "e", then "l", and finally "l". All characters are encoded in the representation of a one-hot vector, where only one unique bit of the vector is turned on. For example

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \text{"h"} \qquad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \text{"e"} \qquad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \text{"l"} \qquad \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \text{"o"}$$

• Then we are going to use the recurrent formula:

$$\begin{bmatrix}0.3\\-0.1\\0.9\end{bmatrix} = F_W\left(W_{hh}\begin{bmatrix}0\\0\\0\end{bmatrix} + W_{xh}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\right)$$

h $\qquad$ x

• Suppose we start off with h as a vector of size 3 with all zeros. By using this fixed recurrent formula we are going to end up with a 3-dimensional representation of the next hidden state h. This basically is any point in time, summarizes all the characters that have come up until then

• h (hidden state) is a 3-dimensional vector ($h \in \mathbb{R}^3$)

• The input vector x is a 4 dimensional vector ($x \in \mathbb{R}^4$)

Date:

• The recurrent formula is:

$$h_1 = \tanh(W_{hh} h_0 + W_{xh} x_1)$$

since $h_0 = 0$, we simplify T_2:

$$h_1 = \tanh(W_{xh} \cdot x_1)$$

- h_1 is a 3×1 vector
- x_1 is a 4×1 vector

• For the multiplication to be valid, $[W_{xh} \cdot x_1]$ dere to matrix multiplication constraints.
W_{xh} must have a dimension 3×4

• Weight Matrix: $W_{xh} \in \mathbb{R}^{3 \times 4}$
- 3 rows (one for each hidden state dimension)
- 4 colums (one for each input dimension)

* Randomly initialized

$$W_{xh} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\text{Row 1} = W_{11} \cdot 1 + W_{12} \cdot 0 \ldots = W_{11}$$
$$\text{Row 2} = W_{21} \cdot 1 + W_{22} \cdot 0 \ldots = W_{21}$$
$$\text{Row 3} = W_{31} \cdot 1 + W_{32} \cdot 0 \ldots = W_{31}$$

$$W_{xh} x = \begin{bmatrix} W_{11} \\ W_{21} \\ W_{31} \end{bmatrix}$$

Date:

$$h_1 = \tanh\begin{bmatrix} W_{11} \\ W_{21} \\ W_{31} \end{bmatrix}$$

Assume

$W_{11} = 0.3$

$W_{21} = -0.1$

$W_{31} = 1.5$

- We randomly initialize weights only during first iter.

- Then we apply our activation function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

1. $\tanh(0.3)$
2. $\tanh((-0.1)$
3. $\tanh(1.5)$

$$\textcircled{1} = \frac{e^{0.3} - e^{-(0.3)}}{e^{0.3} + e^{-(0.3)}}$$

1. $\tanh(0.3) = 0.3$
2. $\tanh(0.1) = -0.1 \rightarrow \begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} = h_1$
3. $\tanh(1.5) = 0.9$

Time Step 2

$$h_2 = \tanh\left(W_{hh}\begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} + W_{xh}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right)$$

- W_{hh} is a 3×3 matrix that is also randomly initialized at step 1. in this example

W_{hh} is all 0's $= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Summary of Forward pass (Time Steps:)

$$h_1 \begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} = \tanh\left(\begin{bmatrix} 000 \\ 000 \\ 000 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.3, 3.0, 0.1, -0.3 \\ -0.1, 0.3, 0.5, 1.5 \\ 1.5, 0.1, -0.3, 0.9 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right)$$

$(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$

Recurrence relation:

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

$$h_2 = \tanh\left(W_{hh} \begin{bmatrix} 0.3 \\ -0.1 \\ 0.9 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} = h_2$$

$$h_3 = \tanh\left(W_{hh} \cdot \begin{bmatrix} 1.0 \\ 0.3 \\ 0.1 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.1 \\ -0.5 \\ -0.3 \end{bmatrix} = h_3$$

$$h_4 = \tanh\left(W_{hh} \cdot \begin{bmatrix} 0.2 \\ -0.5 \\ -0.3 \end{bmatrix} + W_{xh} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} -0.3 \\ 0.9 \\ 0.7 \end{bmatrix} = h_4$$

Date:

- After computing the hidden states (h_t) using our recurrence relation, we can compute y_t

- Step 2: Compute output Logits (y_t)

* Matrix multiplication

$$y_t = W_{hy} h_t$$

- This gives logits for predicting the next character.
- We can do this for all logits at once

- We use the hidden to output weight matrix W_{hy} 4×3
W_{hy} is a learnable parameter ; it is first randomly initialized

Assume

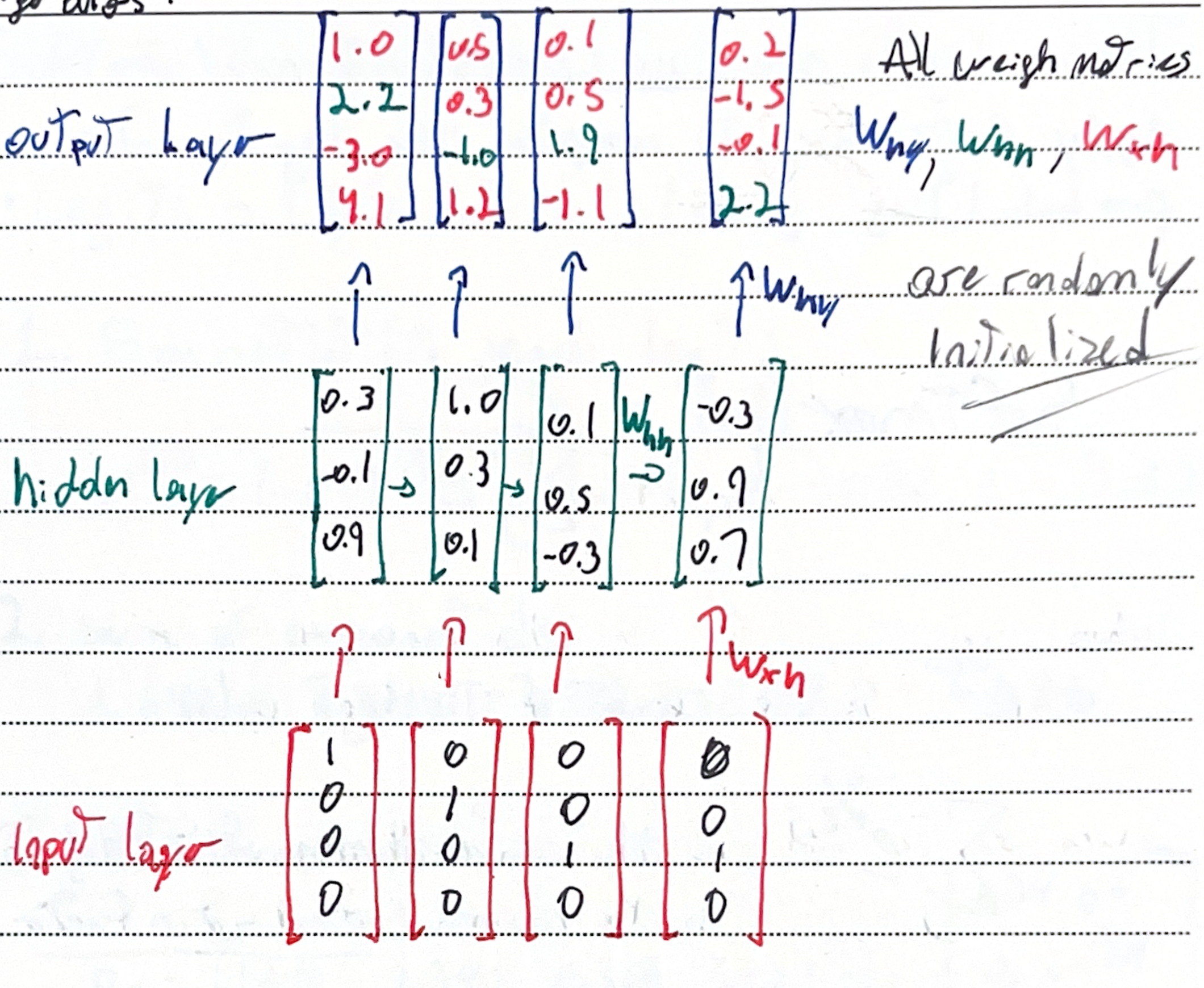
$$W_{hy} = \begin{bmatrix} 0.52 & -0.43 & 0.91 \\ 0.92 & -2.58 & 1.82 \\ -1.21 & 1.62 & -2.75 \\ 1.14 & -1.83 & 4.19 \end{bmatrix} \cdot h_t$$

$$Y = W_{hy} \cdot \begin{bmatrix} 0.3 & 1.0 & 0.1 & -0.3 \\ -0.1 & 0.3 & -0.5 & 0.9 \\ 0.9 & 0.1 & -0.3 & 0.7 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1.0 & 0.5 & 0.1 & 0.2 \\ 2.2 & 0.3 & 0.5 & -1.5 \\ -3.0 & -1.0 & 1.9 & -0.1 \\ 4.1 & 1.2 & -1.1 & 2.2 \end{bmatrix}$$

Target chars: "e" "l" "l" "o"

Date:



All weight matrices W_{hy}, W_{hh}, W_{xh} are randomly initialized

input chars "h" "e" "l" "l"

After one-hot

- Recap of where we are, we just finish calculating our logits (output)

- In the very first time step we fed in h, where the RNN thinks the next character "h" is 1.0 likely

Logits: [1.0, 2.2, -3.0, 4.1] → ["h", "e", "l", "o"]

To come next, "e" is 2.2 likely to come next... and so on

step 1

- RNN thinks "o" should come next! So it guessed wrong!

Date:

- These logits are raw scores, we need to apply Softmax to compute the logits into probabilities

$$\text{Softmax}(y_\tau)_i = \frac{e^{y_{\tau,i}}}{\sum_j e^{y_{\tau,i}}}$$

Where: $y_{\tau,i}$

- e is the exponent of the logit value
- $\sum_j e^{y_{\tau,j}}$ is the sum of all exponentiated logits in the column (normalized in Softo)

- Given logits.

$$y = \begin{bmatrix} 1.0 & 0.5 & 0.1 & 0.2 \\ 2.2 & 0.3 & 0.5 & -1.5 \\ -3.0 & -1.0 & 1.9 & -0.1 \\ 4.1 & 1.2 & -1.1 & 2.2 \end{bmatrix}$$

T_1

- Compute softmax for each time-step

- Softmax For $T_1 = [1.0, 2.2, -3.0, 4.1]$

1) exponentiate each logit:

$e^{1.0} = 2.718$, $e^{2.2} = 9.025$, $e^{-3.0} = 0.050$, $e^{4.1} = 60.34$

2) sum exponentials: $2.718 + 9.025 + 0.050 + 60.34 = 72.13$

3) Softmax Probabilities $= \left[\frac{2.718}{72.13}, \frac{9.025}{72.13}, \frac{0.050}{72.13}, \frac{60.34}{72.13}\right] = [.037, .12, .007, 0.83]$

Date:

Softmax For $\tau = 2$

- Logits = $[0.5, 0.3, -1.0, 1.2]$

1. Exponentiate each logit:

$e^{0.5} = 1.649$ $e^{0.3} = 1.349$, $e^{-1.0} = 0.368$, $e^{1.2} = 3.320$

2. Sum of exponentials:

$1.649 + 1.349 + 0.368 + 3.320 = 6.686$

3. Softmax probabilities:

$$p_2 = \left[\frac{1.649}{6.686}, \frac{1.349}{6.686}, \frac{0.368}{6.686}, \frac{3.320}{6.686}\right] = \begin{bmatrix} 0.2466 \\ 0.2019 \\ 0.0550 \\ 0.4965 \end{bmatrix}$$

- Repeat for $t = 3$ and $t = 4$

- Final softmax probability matrix:

$$P = \begin{bmatrix} 0.0377 & 0.2466 & 0.1131 & 0.1074 \\ 0.1251 & 0.2019 & 0.1687 & 0.0196 \\ 0.0007 & 0.0550 & 0.6841 & 0.0796 \\ 0.8365 & 0.4965 & 0.0341 & 0.7935 \end{bmatrix}$$

- each colum sum To 1, verifying the softmax Transfor

Date:

- Compute the gradients of Loss w.r.t logits

$$\frac{\partial L}{\partial y_t} = p_t - \text{target}_t$$

- Where:
  - p_t is the softmax probability matrix
  - target_t is the one hot encoding of characters
- Given Targets

one of Training on the sequence "hell" -> T

t	target character	one-hot-vector
1	"e"	[0, 1, 0, 0]
2	"l"	[0, 0, 1, 0]
3	"l"	[0, 0, 1, 0]
4	"o"	[0, 0, 0, 1]

- Compute $\frac{\partial L}{\partial y_t}$ or gradients with Target Matrix T

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Date:

• To compute $\frac{\partial L}{\partial Y}$ we subtract the Target matrix from the softmax probability matrix

$$\frac{\partial L}{\partial Y} = P - \text{Target}$$

$$= \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 \end{matrix}$$

$$\begin{bmatrix} 0.0377 & 0.2466 & 0.1131 & 0.1074 \\ 0.1251 & 0.2019 & 0.1687 & 0.0196 \\ 0.0007 & 0.0550 & 0.6841 & 0.0796 \\ 0.8365 & 0.4965 & 0.0341 & 0.7935 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{matrix}$$

$$\frac{\partial L}{\partial Y} = \begin{bmatrix} 0.0377 & 0.2466 & 0.1131 & 0.1074 \\ -0.8749 & 0.2019 & 0.1687 & 0.0196 \\ 0.0007 & -0.9450 & -0.3159 & 0.0796 \\ 0.8365 & 0.4965 & 0.0341 & -0.2065 \end{bmatrix}$$

$\tau_1 \quad \tau_2 \quad \tau_2 \quad \tau_2$

• This is our output layer error.

• H is our 3×4 matrix of hidden states (row = 3 dim hidden, columns = Time step 1...4)

$$H = \begin{bmatrix} 0.3 & 1.0 & 0.1 & -0.3 \\ -0.1 & 0.3 & -0.5 & 0.9 \\ 0.9 & 0.1 & -0.3 & 0.7 \end{bmatrix}$$

• Next is the gradient formula for the hidden-to-output weights $W_{hy} \in \mathbb{R}^{4\times 3}$ at each step t.

Gradient Formula For hidden-To-output weights

$$\frac{\partial L}{\partial W_{hy}} = \sum_{\tau=1}^{4} \left(\frac{\partial L}{\partial y_\tau} \cdot h_\tau^T \right) = \sum_{\tau=1}^{4} (4\times 1)\times(1\times 3) = 4\times 3$$

$\frac{\partial L}{\partial y_\tau} \in R^4$ $\quad h_\tau \in R^{1\times 3}$

- By collecting all 4 colums of $\frac{\partial L}{\partial y}$ (and H) into a single matrix, we can do just a single multiplication
- Transpose H since h is a 3×4 is Transpose is 4×3

$$H^T = \begin{bmatrix} 0.3 & -0.1 & 0.9 \\ 1.0 & 0.3 & 0.1 \\ 0.1 & -0.5 & -0.3 \\ -0.3 & 0.9 & 0.1 \end{bmatrix}$$

Thus

$$\frac{\partial L}{\partial W_{hy}} = \left(\frac{\partial L}{\partial y}\right) \times H^T$$

- Multiply row by row.
  - we wont a final 4×3 result.

Denoted by : $[\Delta_{(1,:)}, \Delta_{(2,:)}, \Delta_{(3,:)}, \Delta_{(4,:)}]$

Date:

- Taking row 1 of $\frac{\partial L}{\partial y}$ which is

$$[0.0377, 0.2466, 0.1131, 0.1074]$$

(we are making a 1×4 vector and multiplying it by $H^{(1)}$)

$$\Delta_{(1,:)} = [0.0377, 0.2466, 0.1131, 0.1074] \cdot \begin{bmatrix} 0.3 & -0.1 & 0.9 \\ 1.0 & 0.3 & 0.1 \\ 0.1 & -0.5 & 0.3 \\ -0.3 & 0.8 & 0.2 \end{bmatrix}$$

- computing each of the 3 entries,
  - column 1 (the "(1,1)" element),

$$= 0.0377 \cdot 0.3 + 0.2466 \cdot 1.0 + 0.1131 \cdot 0.1 + 0.1074 \cdot (-0.3)$$

$$= 0.01131 + 0.2466 + 0.01131 - 0.03222$$

$$\approx 0.2370.$$

- Column 2 ≈ 0.1103
- Column 3 ≈ 0.0998

- Now row 1 The gradient is

$$\Delta_{(1,:)} \approx [0.2370, 0.1103, 0.0998]$$

- Row 2
- Row 3 — Similar for the following rows,
- Row

Date:

Final Gradient $\frac{\partial L}{\partial W_{hy}} \simeq$

$$\begin{bmatrix} 0.237 & 0.1103 & 0.0998 \\ -0.0496 & 0.0814 & -0.8041 \\ -1.003 & -0.0540 & 0.0566 \\ 0.8128 & -0.1376 & 0.6477 \end{bmatrix}$$

- Next we calculate the gradient with respect to h_t

- Once we have $\frac{\partial L}{\partial y_t}$, we get $-\frac{\partial L}{\partial h_t} = W_{hy}^T \left(\frac{\partial L}{\partial y_t} \right)$

- This is another straight forward matrix multiplication

- Then, we would compute the gradients w.r.t W_{hh}

- From $\frac{\partial L}{\partial h_t}$ via back propagation through the recurrent connection. (Since our W_{hh} matrix is 0 we don't)

$$\frac{\partial L}{\partial W_{hh}} = \sum_{\tau} \left[\left(\frac{\partial L}{\partial h_\tau} \odot \frac{d \tanh(z_t)}{d z_t} \right) h_{\tau-1}^T \right]$$

- Similarly, we compute the gradients w.r.t W_{xh}

- From $\frac{\partial L_i}{\partial h_t}$ via back prop to the inputs to hidden weights

$$\frac{\partial L}{\partial W_{xh}} = \sum_{\tau} \left[\left(\frac{\partial L}{\partial h_\tau} \odot \frac{d \tanh(z_\tau)}{d z_\tau} \right) x_\tau^T \right]$$