



Cairo University

Cairo University

Cairo University
Faculty of Engineering
Computer Engineering Department

Natural Language Processing

0100 0100
0110 0110
1001 1001
0100110
0100001
1101
1101

Dr. Sandra Wahid

learning processing
natural language NLP
text linguistics interaction
automatic programming technologies
understanding data evolution
linguistics science cloud automation
interaction machine learning
interaction communication
output systems
output communication
systems
output communication
systems

Motivation

RNN -> SEQUENTIAL CIRCUITS
FFNN -> COMBINATIONAL CIRCUITS

- Language is an inherently **temporal** phenomenon.
 - Normally processed sequentially.
- The primary weakness of our earlier **Markov N-gram** approaches is that it **limits the context** from which information **can be extracted**; anything outside the context window has no impact on the **decision** being made.
 - This is an issue since there are many tasks that require access to information that can be arbitrarily **distant** from the point at which processing is happening.

i **Recurrent Neural Networks(RNNs):** have mechanisms to deal directly with the sequential nature of language that allow them to handle variable length inputs without the use of arbitrary fixed sized windows, and to capture and exploit the temporal nature of language.

Recurrent Neural Networks

- A recurrent neural network (RNN) is any network that contains a **cycle** within its network connections.
 - That is, any network where the value of a unit is directly, or indirectly, dependent on its own earlier outputs as an input.

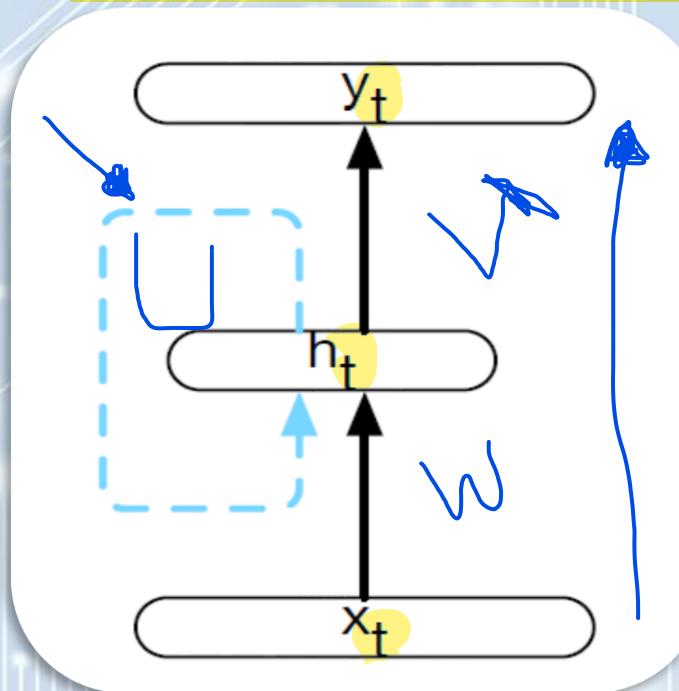
We have 3 main matrices, we should try to learn:

W: Between the input and the hidden layer

U: Between the current hidden layer and the previous hidden layer.

V: Between the hidden layer and the output layer.

zy ma fe el sequential circuits, enta mkontsh bt8yr el hardware bta3 el memory, fa dol msh byt8yro brdu



dol msh layer wahda, dol mmkn ykono 3, 6 ,12 ay 3adad 3ady, bs kolohom nfs el shakl, kol el fekra en el t bs bttghyr, fa da m3nah msln lw dkhltlk gomla zy (hello dady I am here) n2dr n3br 3nhom fe shakl sequence kaltaly : (1,Hello), (2,Dady), (t=3, I), (t=4, am), (t=5, Here), w bkda hyb2a 3ndna nfs el shakl da akno mtkrr 5 mrat, w b3den el h hya bt3br 3n hidden layers, fa homa el bysImmo b3d, lagn enta kol mara bydkhol input kelma gdeda, w enta bt13 output kelma gdeda, w el h, bt3tmd 3la el context el adem.

de aknha t layers.

- x_t :an **input vector** representing the current input.
- x_t is multiplied by a **weight matrix** and then passed through a **non-linear activation function** to compute the values for a **layer of hidden units**.
- This hidden layer is then used to calculate a corresponding **output: y_t**

Recurrent Neural Networks

- Sequences are processed by presenting **one item at a time** to the network.
- The **key difference** from a **feedforward network** lies in the **recurrent link** shown in the figure with the dashed line →**temporal dimension**
- This link augments the **input** to the computation at the hidden layer with the **value of the hidden layer from the preceding point in time**.
- The hidden layer from the previous time step provides a **form of memory**, or context, that encodes earlier processing.
- Critically, this approach does **not impose a fixed-length limit** on this prior context
 - the context embodied in the previous hidden layer includes information extending back to the **beginning of the sequence**.

compute dimensions (d elly ray7lo x d elly gy meno)

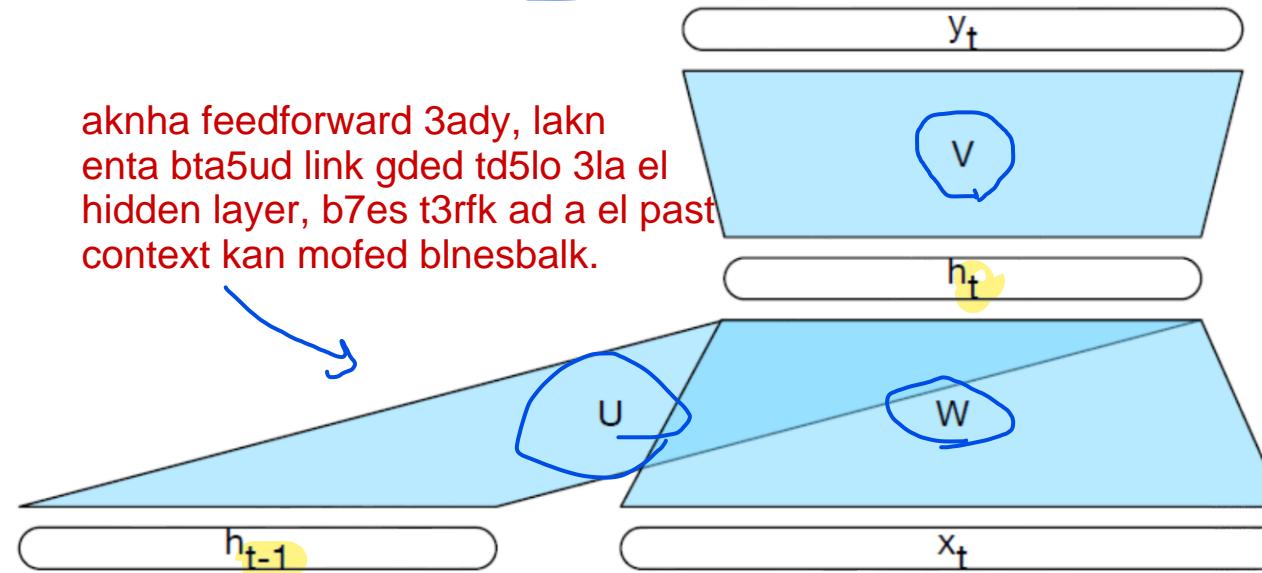
Recurrent Neural Networks

mt7fzsh 7aga, efhm bs el denya mashya ezay w kolo hyb2a sahl.
compute dimensions: $W(dhxdin)$, $U(dhxdh)$, $V(doutxdh)$

el U bt3alemak ezay el past
bta3k bysa3dk 34an tkawen
el present.

El values bta3t el matricies
btt8yr emta? fl learning phase
lakn fl testing phase wl production
khlas baa el U, W, V have
constant values, el bytghyr baa
homa el inputs wl outputs.

aknha feedforward 3ady, lagn
enta bta5ud link gded td5lo 3la el
hidden layer, b7es t3rfk ad a el past
context kan mofed blnesbalk.



U -> the weights between
two hidden layers.

W -> the weights between
the inputs and the hidden
layers

V -> this is the weights
between the hidden layers
and the outputs.

dh
din
dout

- The most significant change lies in the new set of weights: U that connect the hidden layer from the previous time step to the current hidden layer.
- These weights determine how the network makes use of past context in calculating the output for the current input.
- As with the other weights in the network, these connections are trained via backpropagation.

Inference in RNNs

- To compute an output y_t for an input $x_t \rightarrow$ we need the activation value for the hidden layer h_t .
- To calculate this, we multiply the input x_t with the weight matrix W , and the hidden layer from the previous time step h_{t-1} with the weight matrix U .
- We add these values together and pass them through a suitable activation function: g to arrive at the activation value for the current hidden layer h_t .

$$h_t = g(Uh_{t-1} + Wx_t)$$

- Once we have the values for the hidden layer, we proceed with the usual computation to generate the output vector.

$$y_t = f(Vh_t)$$

- In the commonly encountered case of soft classification, f is a softmax function that provides a probability distribution over the possible output classes.

$$y_t = \underline{\text{softmax}}(Vh_t)$$

Inference in RNNs

ay dimension hwa(dimension elly raye7lo * dimension elly gy meno.)



It's worthwhile here to be careful about specifying the dimensions of the input, hidden and output layers, as well as the weight matrices to make sure these calculations are correct. Let's refer to the input, hidden and output layer dimensions as d_{in} , d_h , and d_{out} respectively. Given this, our three parameter matrices are: $W \in \mathbb{R}^{d_h \times d_{in}}$, $U \in \mathbb{R}^{d_h \times d_h}$, and $V \in \mathbb{R}^{d_{out} \times d_h}$

- The fact that the computation at time t requires the value of the hidden layer from time t-1 mandates an **incremental inference algorithm** that proceeds from the start of the sequence to the end.

el output bta3na byb2a mn kol
time step.

el W, V, U sabten fe kol el iterations.

leh ? l2n enta el mfrod tkon btt3lm
nfs el 7aga, f leh t8yro?

mehtag a2ra aktur fl 7war da...

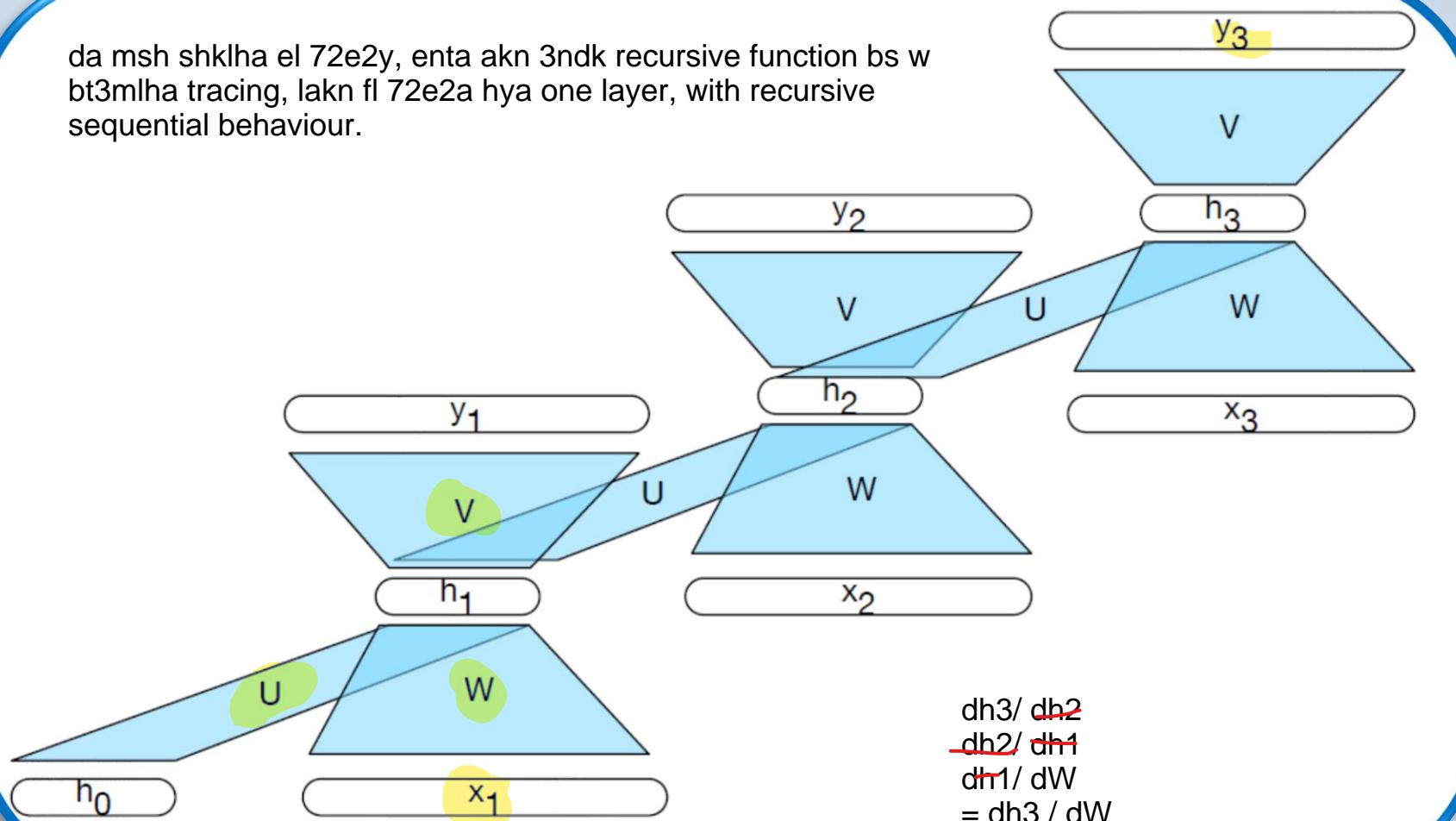
```
function FORWARDRNN( $x, network$ ) returns output sequence  $y$ 
     $h_0 \leftarrow 0$  usually byb2a b 0, w mmkn 7d yeb2a y3dlo lw l2ah bygeb better performance.
    for  $i \leftarrow 1$  to LENGTH( $x$ ) do
         $h_i \leftarrow g(U h_{i-1} + W x_i)$ 
         $y_i \leftarrow f(V h_i)$  output of each layer, w bgm3hom fl akher fl vector  $y$ 
    return  $y$  da vector contains all  $y_i$ 
```

The matrices **U, V and W are shared across time**, while new values for **h and y are calculated with each time step**.

Inference in RNNs

- The sequential nature of simple recurrent networks can also be seen by unrolling the network in time:

da msh shklha el 72e2y, enta akn 3ndk recursive function bs w bt3mlha tracing, lagn fl 72e2a hya one layer, with recursive sequential behaviour.



$$\begin{aligned} dh_3 / dh_2 \\ dh_2 / dh_1 \\ dh_1 / dW \\ = dh_3 / dW \end{aligned}$$

Training

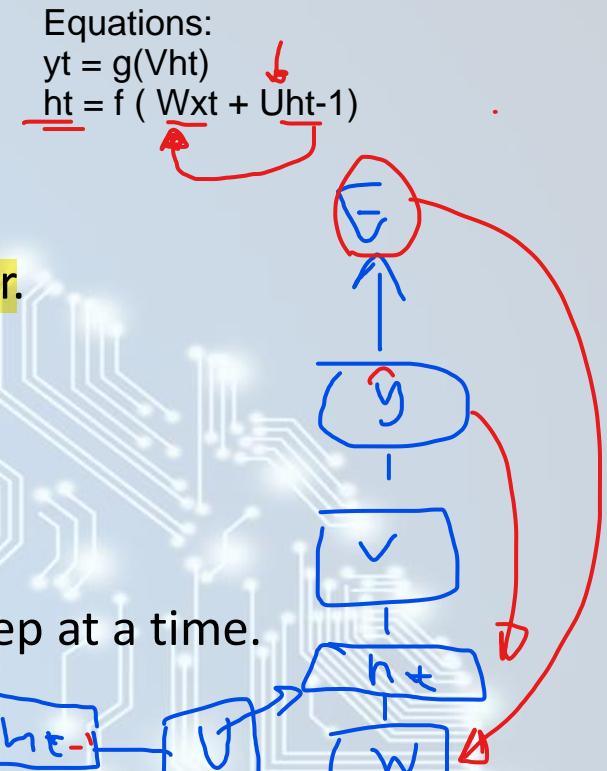
- A neural network is an instance of **supervised** machine learning.
 - We know the **correct output y** for each **observation x** .
 - The system generates **output \hat{y}** .
- The goal of the **training procedure** is to learn parameters such as W that makes \hat{y} as close as possible to the true y .
 - First, we'll need a **loss function** that models the **distance between** the system output and the **gold output**.
 - Second, to find the parameters that minimize this loss function, we'll use the **gradient descent optimization** algorithm.

Forward propagation: Computing the error.

Backward propagation: Computing the partial derivatives and updating the parameters.

Training

- In RNN, there are 3 sets of weights to update:
 - W**, the weights from the input layer to the hidden layer.
 - U**, the weights from the previous hidden layer to the current hidden layer.
 - V**, the weights from the hidden layer to the output layer.



- RNNs are trained using **Backpropagation Through Time (BPTT)**:

- BPTT begins by unrolling a recurrent neural network in time.
- Propagates the error backward over the entire input sequence, one timestep at a time.
- Equations to calculate the error gradients:
 - V**: depends only on the **current timestamp**.
 - W and U**: depend on **current timestamp and previous** ones.
 - Using Chain rule at each timestep:

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial V}$$

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

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

E 1

h_{t-1}

h_t

y_t

W

x_t

NOTE THIS
EQUATION:

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

Ex: at timestep 3:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial W} + \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W}$$

- Total Error: is the sum of errors for all timesteps

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

enta msh m7tag te7fz el equation, enta me7tag ttab2 el chain rule fe dmaghk, w htla2ek btws1 lel sequence el s7. h3 * h2 , h2 * h1, h1 * h0 and so on...

Training

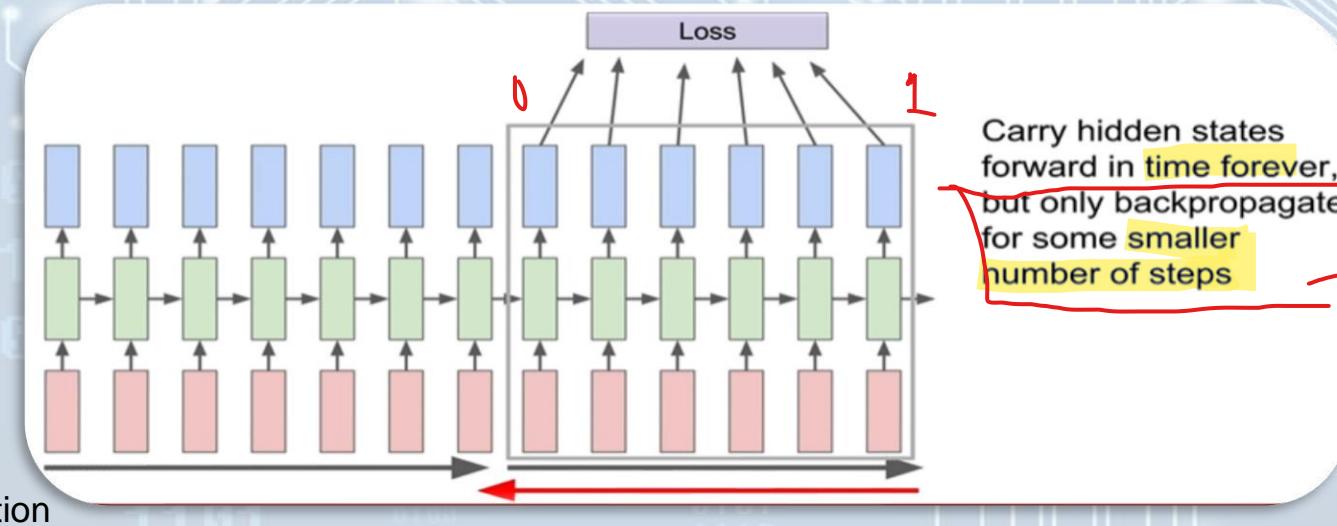
- For applications that involve **long input sequences**, such as speech recognition, character-level processing, or streaming of continuous inputs, unrolling an entire input sequence may not be feasible and BPTT becomes very expensive and problematic (*check BPTT disadvantages with long sequences*).

- One solution is to use **Truncated Backpropagation Through Time**.

- Choose a **number of timesteps** to use as input → split up the long input sequences into subsequences.
- A modification of BPTT to limit the number of timesteps used on the backward pass and in effect estimate the gradient used to update the weights rather than calculate it fully.

Ana msh fahem el 7eta de awy, lakn el ana fahmo, enha tre2t approximation, laknha btgeb accuracy kwysa.
w kman ehna msh bnrg3 lel akher, l2 e7na bno2af 3nd 7eta mo3yna fl nos.

w kman el ana fahmo, en ana msh b7sb mn n : 1
laa e7na bn7sb mn n: k
w sa3tha da byb2a approximation



hwa hena b3d ma brg3 n msln,
ezay brbot el errors bb3d?

ehna bdl ma bn5ly el darb l7d el 1, byb2a
l7d el n, search aktur fl 7war da.

RNNs as Language Models

- RNN language models process the input sequence **one word at a time**, attempting to **predict the next word** from **the current word and the previous hidden state**.

i RNNs **don't have the limited context** problem that **n-gram** models have, since the hidden state can in principle represent information **about all of the preceding words** **all the way back to the beginning of the sequence**.

- The input sequence: $\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_t; \dots; \mathbf{x}_N]$ consists of a series of **word embeddings** each represented as a **one-hot vector** of size $|\mathcal{V}| \times 1$.
- The output prediction: \mathbf{y} is a vector representing a **probability distribution** over the vocabulary.

- \mathbf{E} the word embedding matrix: used to retrieve the embedding for the current **word**.

RNNs as Language Models

we2fna hena.

- At time t:

$$\begin{aligned}\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)\end{aligned}$$

- The vector resulting from $\mathbf{V}\mathbf{h}$ can be thought of as a **set of scores** over the vocabulary given the evidence provided in \mathbf{h} .
- Passing these scores through the softmax normalizes the scores into a **probability distribution**.
- The probability that a particular word i in the vocabulary is the next word is represented by $\mathbf{y}_t[i] \rightarrow$ the **ith component of \mathbf{y}_t** :

$$P(w_{t+1} = i | w_1, \dots, w_t) = \mathbf{y}_t[i]$$

RNNs as Language Models

- The probability of an entire sequence is just the product of the probabilities of each item in the sequence.
- $y_i[w_i]$ means the probability of the true word w_i at time step i

$$\begin{aligned} P(w_{1:n}) &= \prod_{i=1}^n P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^n y_i[w_i] \end{aligned}$$

- Training an RNN as a language model:
 - we use a **corpus of text** as training material
 - the model **predict the next word** at each time step t .
 - we **minimize the error** in predicting the true next word in the training sequence, using **cross-entropy** as the loss function.

Cross Entropy Loss

- Measures the difference between a predicted probability distribution and the correct distribution.
- For example, in the case of Binary Classification, cross-entropy is given by:

$$l = -(y \log(p) + (1 - y) \log(1 - p))$$

- p is the predicted probability
- y is the indicator (0 or 1 in the case of binary classification)
- For example, if the correct indicator is $y=1$

$$l = -(1 \times \log(p) + (1 - 1) \log(1 - p))$$

$$l = -(1 \times \log(p))$$

- Therefore, our loss function will reward the model for giving a correct prediction (high value of p) with a low loss. However, if the probability is lower, the value of the error will be high and therefore it penalizes the model for a wrong outcome. → **best when loss=0** (the probability of the correct class=1)
- Extension for a Multi-Classification (N classes):

$$-\sum_{c=1}^N y_c \log(p_c)$$

RNNs as Language Models

- The cross-entropy loss for language modeling is determined by the probability the model assigns to the **correct next word**.
- So at time t the **CE** loss is the negative log probability the model assigns to the next word in the training sequence.

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

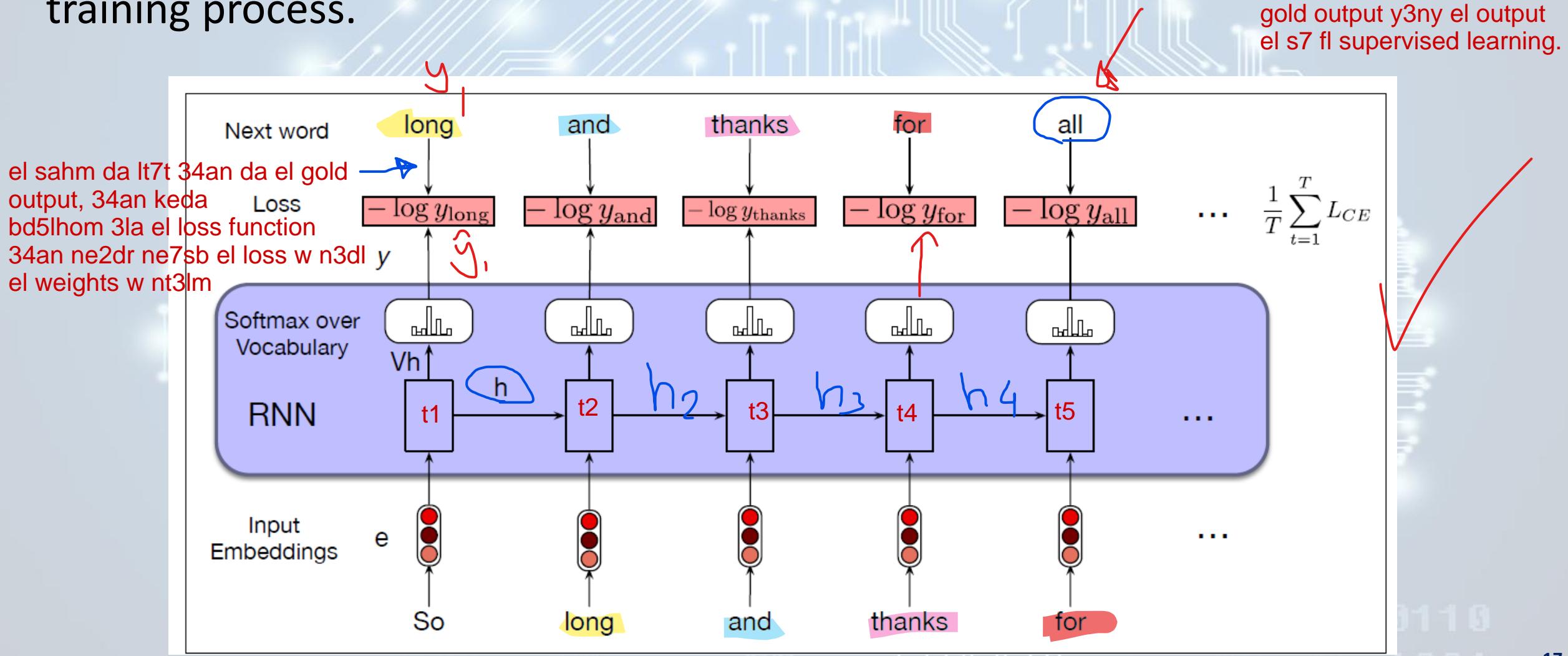
• Teacher Forcing

At each word position t of the input, the model takes as input the **correct sequence of tokens $\mathbf{w}_{1:t}$** and uses them to compute a probability distribution over possible next words and computes the model's loss for the next token w_{t+1} . Then we move to the next word, we ignore what the model predicted in the previous time step and instead use the correct sequence of tokens $\mathbf{w}_{1:t+1}$ to estimate the probability of token w_{t+2} .

The idea is that we always give the model the correct history sequence to predict the next word (rather than feeding the model its best case from the previous time step)

RNNs as Language Models

- The weights in the network are adjusted to minimize the average CE loss over the training sequence via gradient descent. The following figure illustrates the training process.



RNNs as Language Models

- **Weight tying:** is a method that dispenses redundancy and simply uses a single set of embeddings at the input and softmax layers.
 - We dispense with V and use E in both the start and end of the computation.

$$\begin{aligned}\mathbf{e}_t &= \mathbf{Ex}_t \\ \mathbf{h}_t &= g(\mathbf{Uh}_{t-1} + \mathbf{We}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{E}^{\text{transpose}} \mathbf{h}_t)\end{aligned}$$

Dimension of the hidden layer dh = Dimension of embeddings

- This provides **improved model perplexity**, and significantly **reduces the number of parameters** required for the model.



Thank You

0100 0100

0110 0110

1001 1001

0100 0110

0100 001

1101 0101

1010
0110

1001
0101

0100
1101

1101
0101

0110
1001

0101
0101

0100
0011

0110

1001