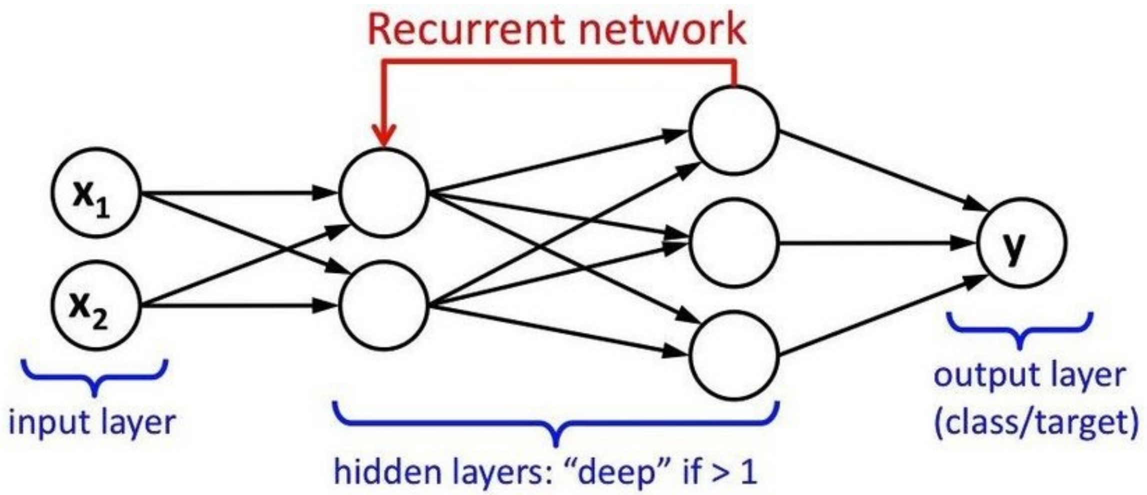
Deep Learning

Elisabetta Fersini

elisabetta.fersini@unimib.it

Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) are a form of neural networks specialized for processing sequential data $x^{(1)}, \ldots, x^{(T)}$.



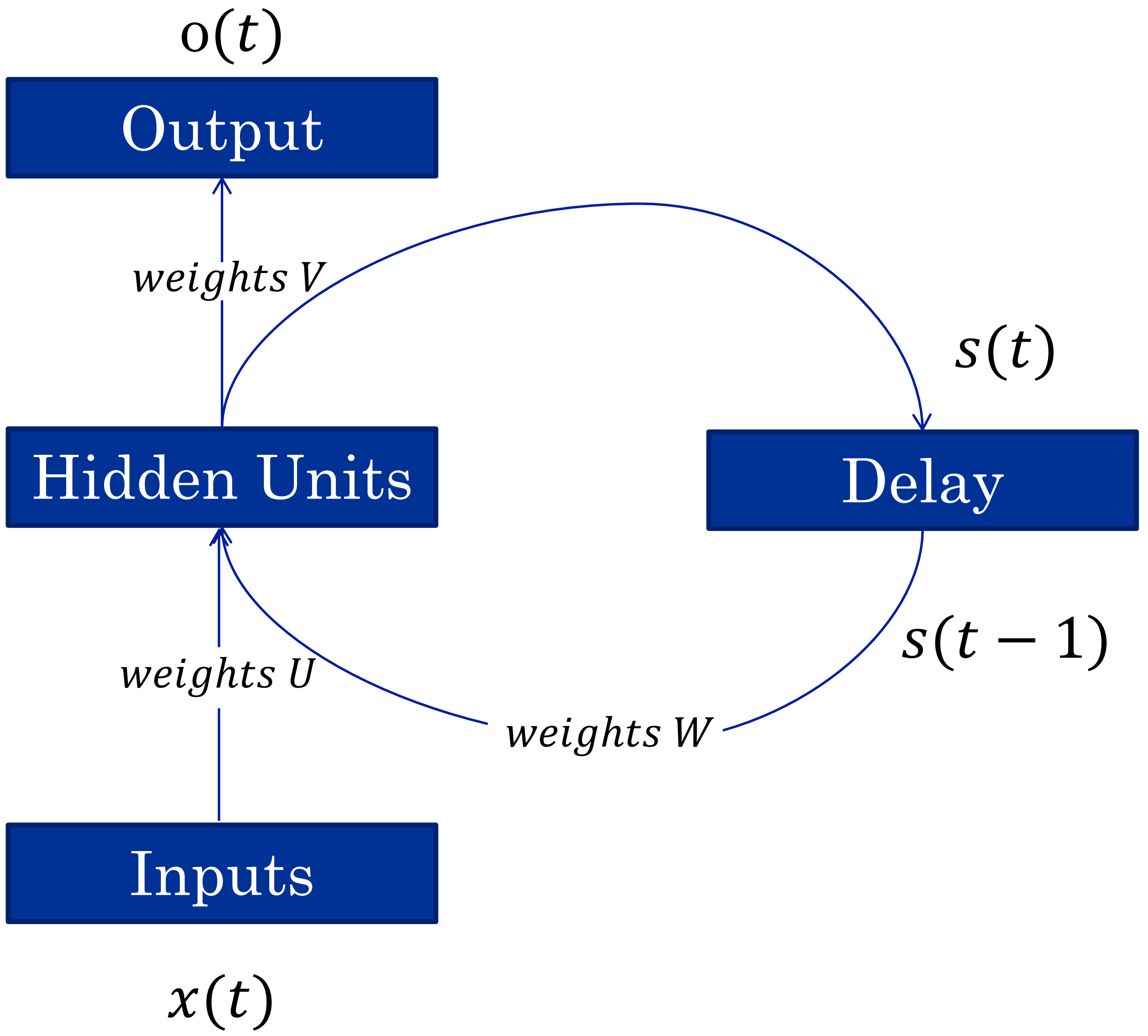
- For moving from multi-layer networks to RNNs, we need to take advantage of the idea of sharing parameters across different parts of a model.

Recurrent Neural Networks

- For instance consider the two sentences:
  - "I went to Nepal in 2009"
  - "In 2009, I went to Nepal."

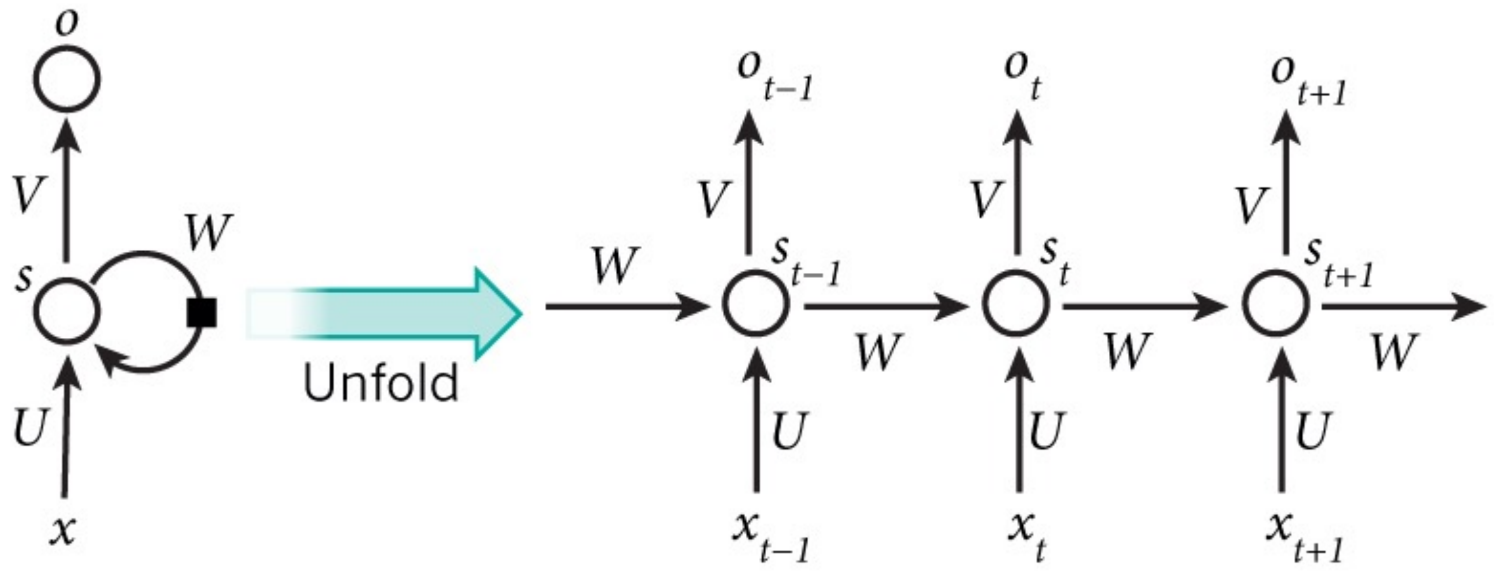
- Let's assume that we have trained a feedforward network **A** that processes sentences of fixed length.
  - **A** would have separate parameters for each input feature, so it would need to learn all of the rules of the language separately at each position in the sentence. By comparison, RNNs share the same weights across several time steps.

RNN Architecture



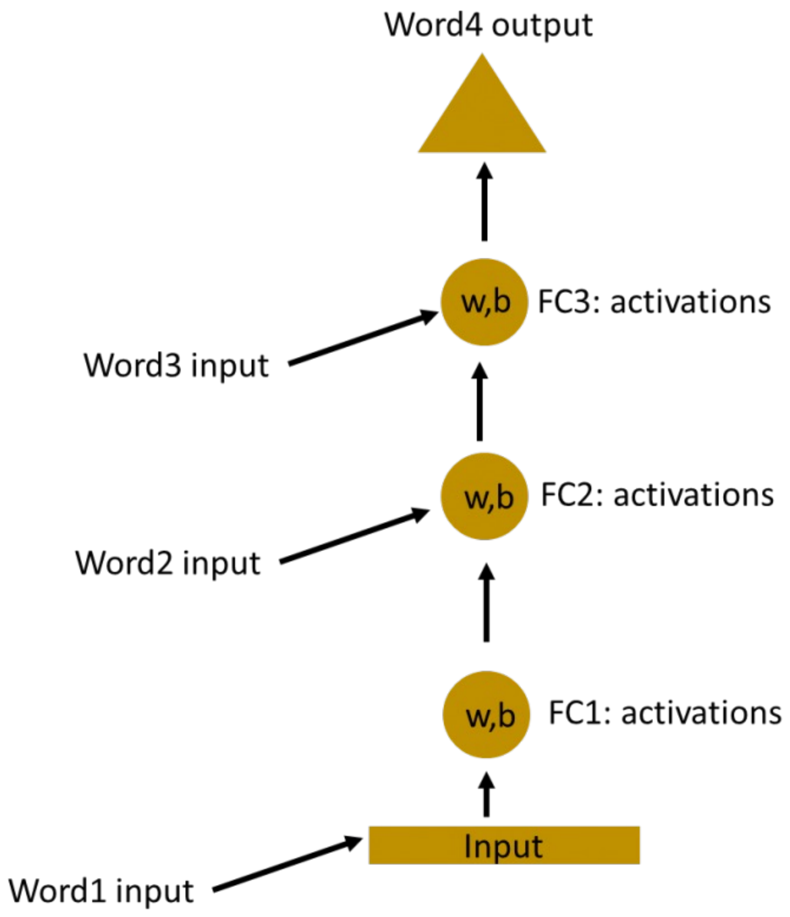
RNN Architecture

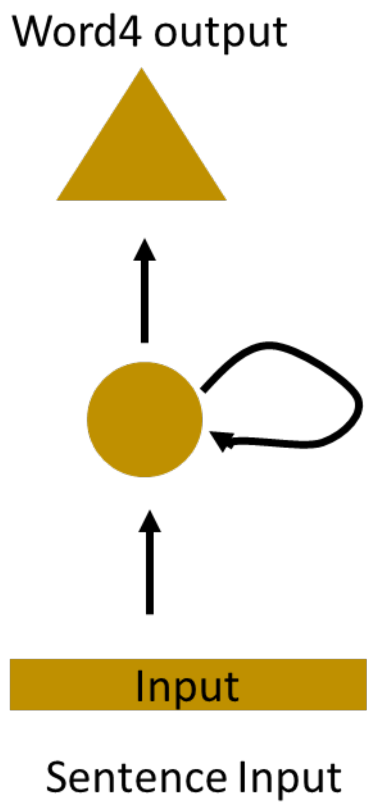
- Unfolding RNNs



RNN - EXAMPLE

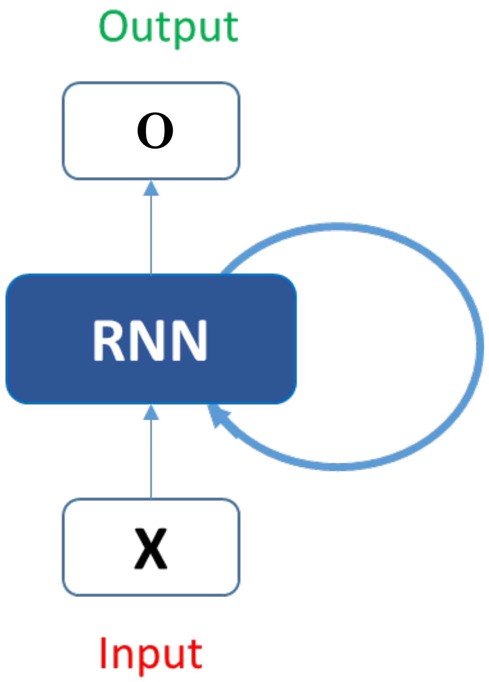
- What we need is to have the same parameters independently on the lenght of the data





RNN - EXAMPLE

- Let's take a character level RNN where we have a word "Hello". So we provide the first 4 letters i.e. h,e,l,l and ask the network to predict the last letter i.e.'o'. So here the vocabulary of the task is just 4 letters {h,e,l,o}.



RNN - EXAMPLE

- In this case, the letter “h” has nothing preceding it, let’s take the letter “e”. So at the time the letter “e” is supplied to the network, a recurrence formula is applied to the letter “e” and the previous state which is the letter “h”.

- The *recurrence formula* is applied to both “e” and “h” to obtain the new state:

$$s_t = f(s_{t-1}, x_t)$$

where s_t is the new state, s_{t-1} is the previous state while x_t is the current input.

- Basically we have a state that represents the previous input and the input itself.

RNN - EXAMPLE

- Taking the simplest form of a recurrent neural network, let's say that the activation function is tanh, the weight at the recurrent neuron is W and the weight at the input neuron is U, we can write the equation for the state at time t as:

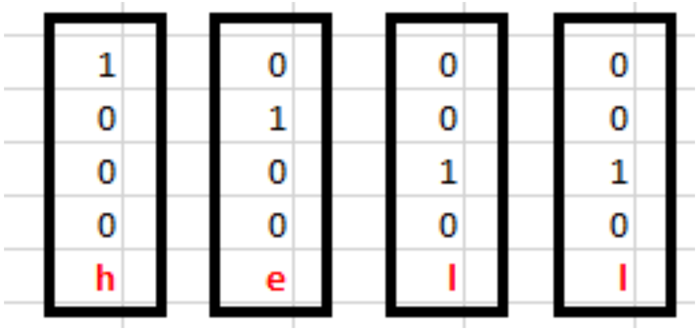
$$s_t = \tanh(W s_{t-1} + U x_t)$$

- Once the final state is computed we can produce the output:

$$o_t = V s_t$$

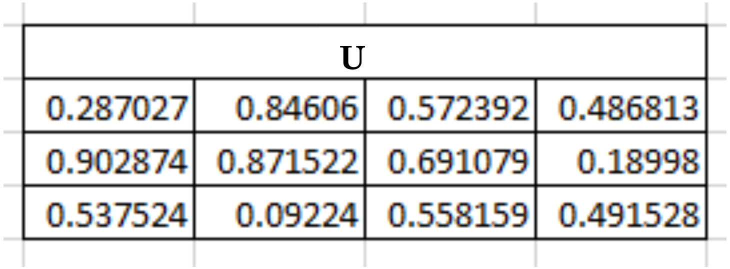
RNN - EXAMPLE

- Let’s start.. The inputs are one hot encoded. Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.



1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

- Now the input neuron would transform the input to the hidden state using the weight matrix U (initially randomly initialized)

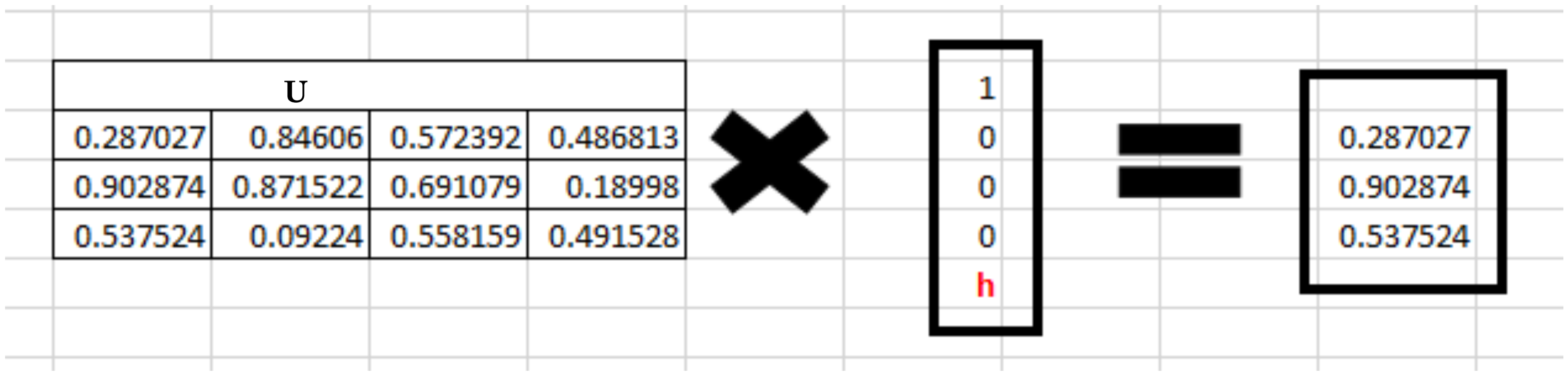


U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

E. Fersini

RNN - EXAMPLE

- Now for the letter “h”, for the the hidden state we would need $U*x_t$.

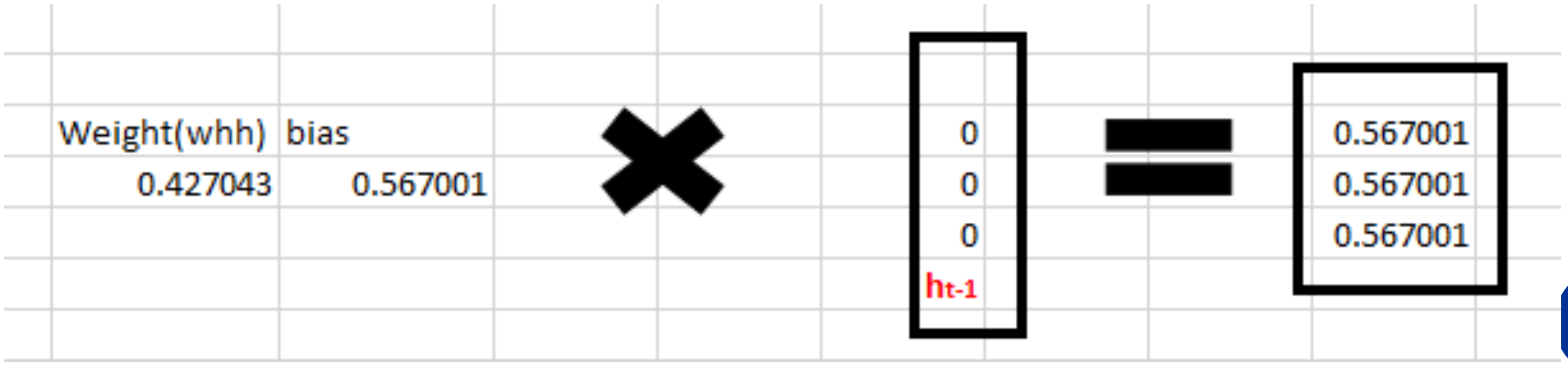


$$\begin{array}{|c|c|c|c|} \hline \multicolumn{4}{|c|}{\mathbf{U}} \\ \hline 0.287027 & 0.84606 & 0.572392 & 0.486813 \\ \hline 0.902874 & 0.871522 & 0.691079 & 0.18998 \\ \hline 0.537524 & 0.09224 & 0.558159 & 0.491528 \\ \hline \end{array} \times \boxed{\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ \mathbf{h} \end{array}} = \boxed{\begin{array}{c} 0.287027 \\ 0.902874 \\ 0.537524 \end{array}}$$

E. Fersini

RNN - EXAMPLE

- Now moving to the recurrent neuron, we have W as the weight which is a 1*1 matrix with value 0.427043 and the bias which is also a 1*1 matrix with value 0.56700.
- For the letter "h", the previous state is [0,0,0] since there is no letter prior to it.
- We can compute $W*s_{t-1}$

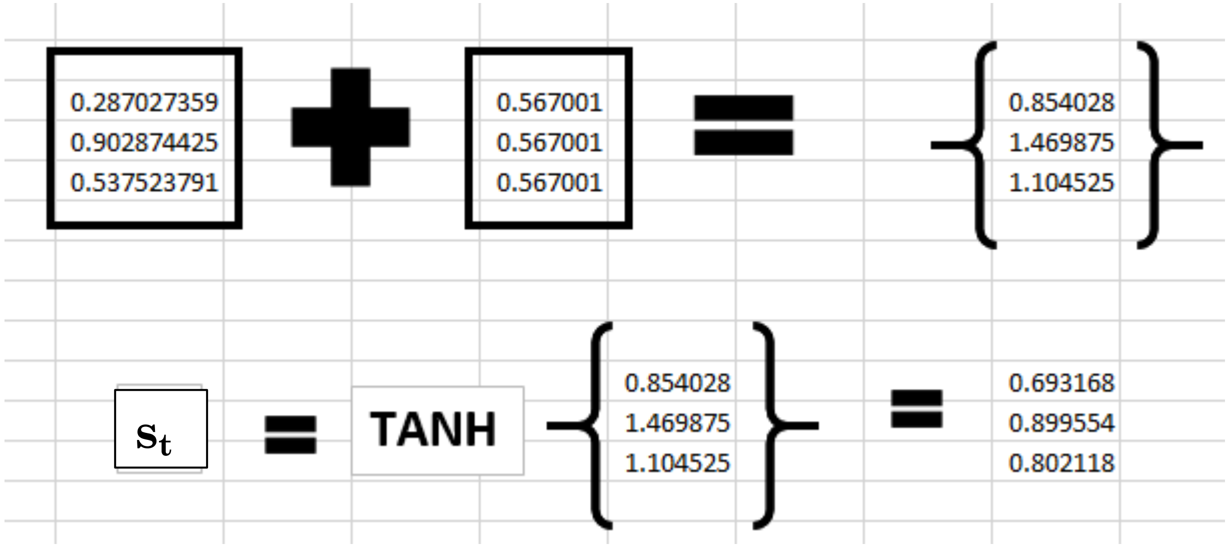

Weight(whh)
bias
0.427043
0.567001
0
0
0
ht-1
0.567001
0.567001
0.567001

whh*ht-1+bias

RNN - Example

- We can now estimate the current state s_t

$$s_t = \tanh(Ws_{t-1} + Ux_t)$$



$$\boxed{\begin{matrix} 0.287027359 \\ 0.902874425 \\ 0.537523791 \end{matrix}} + \boxed{\begin{matrix} 0.567001 \\ 0.567001 \\ 0.567001 \end{matrix}} = \left\{ \begin{matrix} 0.854028 \\ 1.469875 \\ 1.104525 \end{matrix} \right\}$$

$$\boxed{s_t} = \text{TANH} \left\{ \begin{matrix} 0.854028 \\ 1.469875 \\ 1.104525 \end{matrix} \right\} = \begin{matrix} 0.693168 \\ 0.899554 \\ 0.802118 \end{matrix}$$

RNN - EXAMPLE

- Now we can move to the next state. The letter "e" is now supplied to our RNN. The processed output of s_t, now becomes s_{t-1}, while the one hot encoded "e" is our observation x_t.
- We can now effectively estimate our current state s_t as usual:

$$s_t = \tanh(Ws_{t-1} + Ux_t)$$

- $W * s_{t-1}$ + *bias* will be:

$$\boxed{\mathbf{W*s_t+Bias}} = \boxed{0.427043} \times \begin{array}{r} 0.69316804 \\ 0.89955366 \\ 0.8021184 \end{array} + \boxed{0.567001} = \begin{array}{r} 0.863013 \\ 0.951149 \\ 0.90954 \end{array}$$

- $U * x_t$ will be:

$$\begin{array}{|c|c|c|c|} \hline \multicolumn{4}{|c|}{\mathbf{U}} \\ \hline 0.287027359 & 0.84606 & 0.572392 & 0.486813 \\ \hline 0.902874425 & 0.871522 & 0.691079 & 0.18998 \\ \hline 0.537523791 & 0.09224 & 0.558159 & 0.491528 \\ \hline \end{array} \times \begin{array}{|c|} 0 \\ 1 \\ 0 \\ 0 \\ \text{e} \\ \hline \end{array} = \begin{array}{|r|} \hline 0.84606 \\ \hline 0.871522 \\ \hline 0.09224 \\ \hline \end{array}$$

RNN - EXAMPLE

- The activation function will add non-linearity:

$$\boxed{s_t} = \boxed{\text{TANH}} \left\{ \begin{bmatrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{bmatrix} + \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix} \right\} = \begin{matrix} 0.93653372 \\ 0.94910403 \\ 0.76234056 \end{matrix}$$

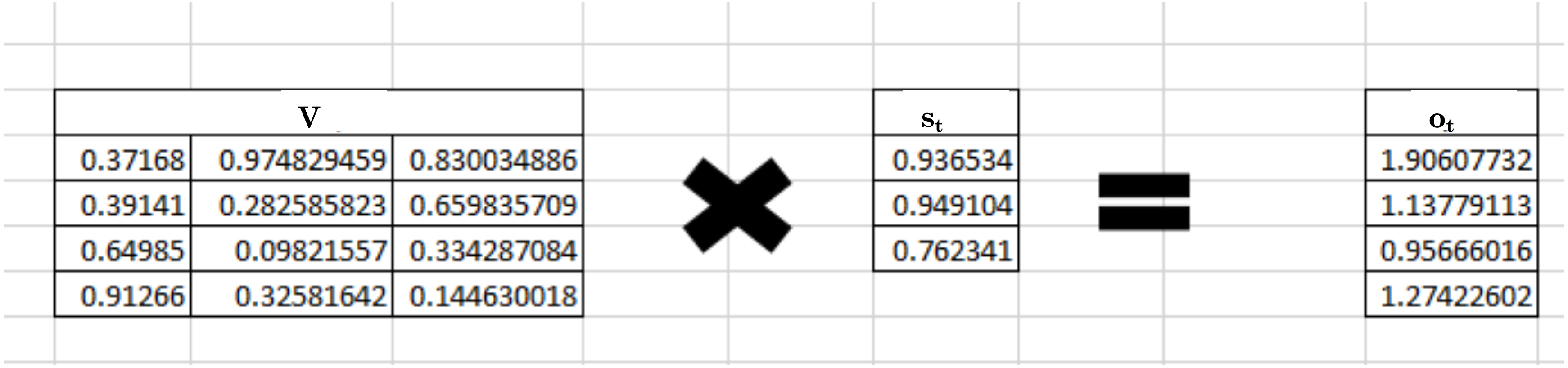
- Now this would become s_{t-1} for the next state and the recurrent neuron would use this along with the new character to predict the next one.
- At each state, the RNN would produce an output o_t as well:

$$o_t = V s_t$$

RNN - EXAMPLE

- At each state, the RNN would produce an output o_t as well:

$$o_t = Vs_t$$



V		
0.37168	0.974829459	0.830034886
0.39141	0.282585823	0.659835709
0.64985	0.09821557	0.334287084
0.91266	0.32581642	0.144630018

×

s_t
0.936534
0.949104
0.762341

=

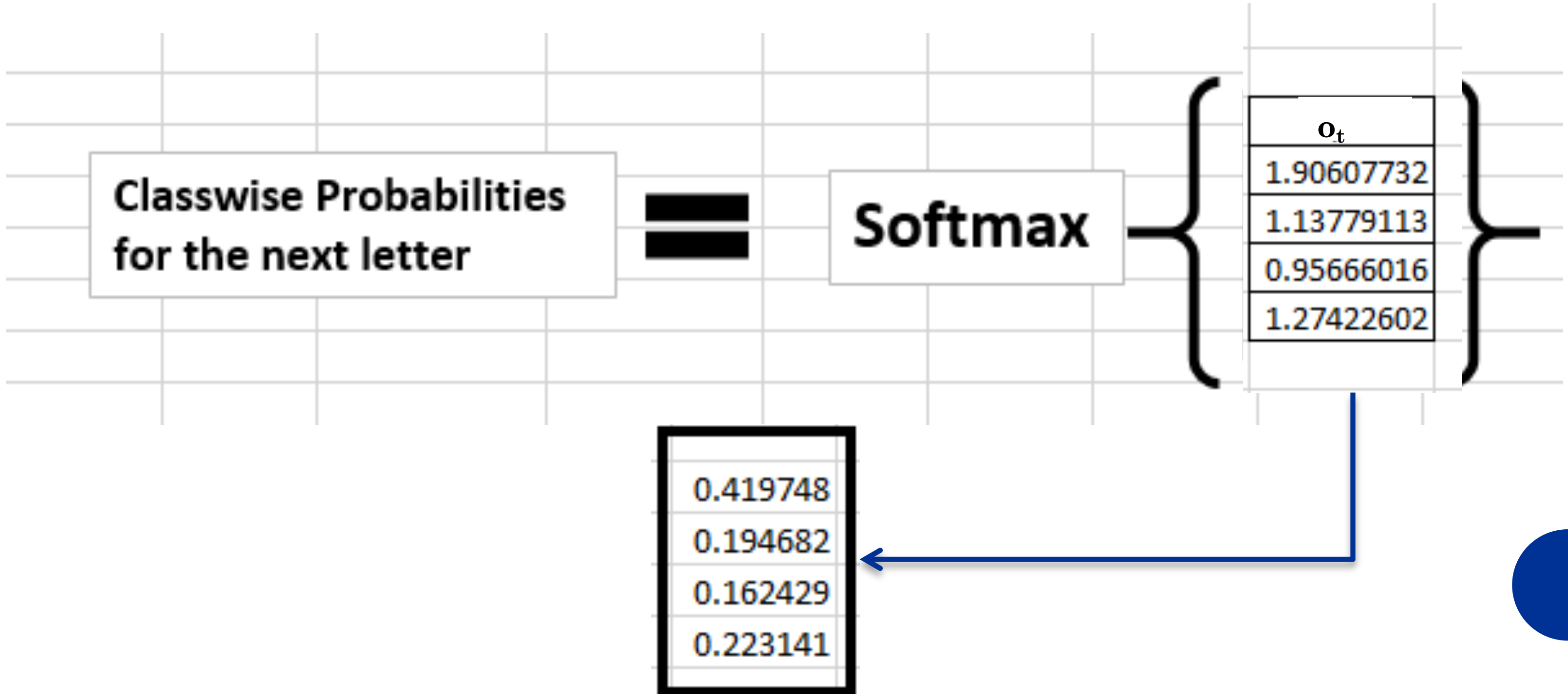
o_t
1.90607732
1.13779113
0.95666016
1.27422602

E. Fersini

RNN - EXAMPLE

- The probability for a given letter from the vocabulary can be estimated by using the softmax function:

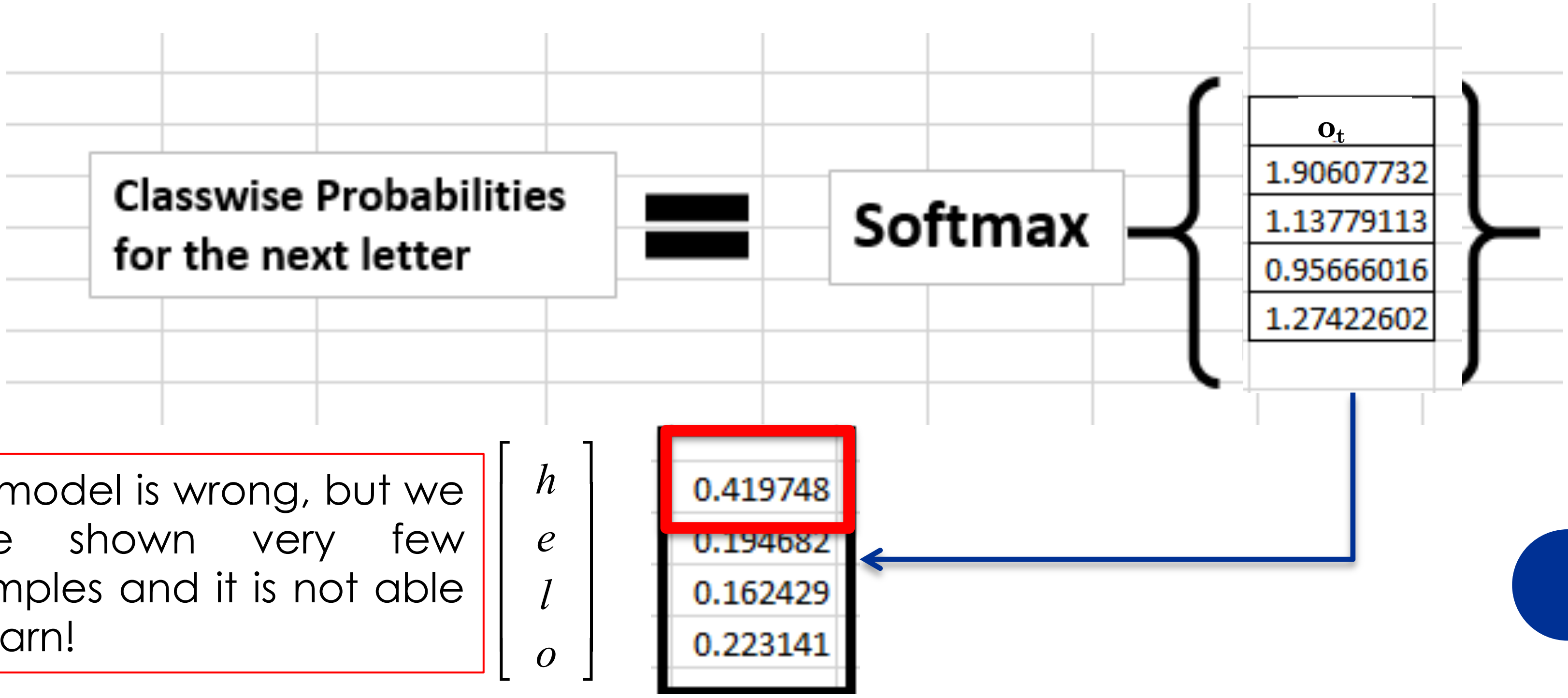
$$P(o_t) = softmax(o_t)$$



RNN - EXAMPLE

- The probability for a given letter from the vocabulary can be estimated by using the softmax function:

$$P(o_t) = soft\max(o_t)$$



The model is wrong, but we have shown very few examples and it is not able to learn!

RNN Training

- Training a RNN means estimate the parameter W, U and V :

1. A single time step of the input is supplied to the network
2. We compute s_t using a combination of the current input and the previous state
3. The current h_t becomes s_{t-1} for the next time step
4. We can go as many time steps as required
5. Once all the time steps are completed the final current state is used to compute the output o_t
6. The output is then compared to the actual output and the error is estimated
7. The error is backpropagated to the network

RNN Training

- If o_t is the predicted value $\bar{o}_t$ is the actual value, the error is computed as a cross entropy loss

$$L_t = -\bar{o}_t \log(o_t)$$

- And therefore, for all time stamps:

$$L = -\sum \bar{o}_t \log(o_t)$$

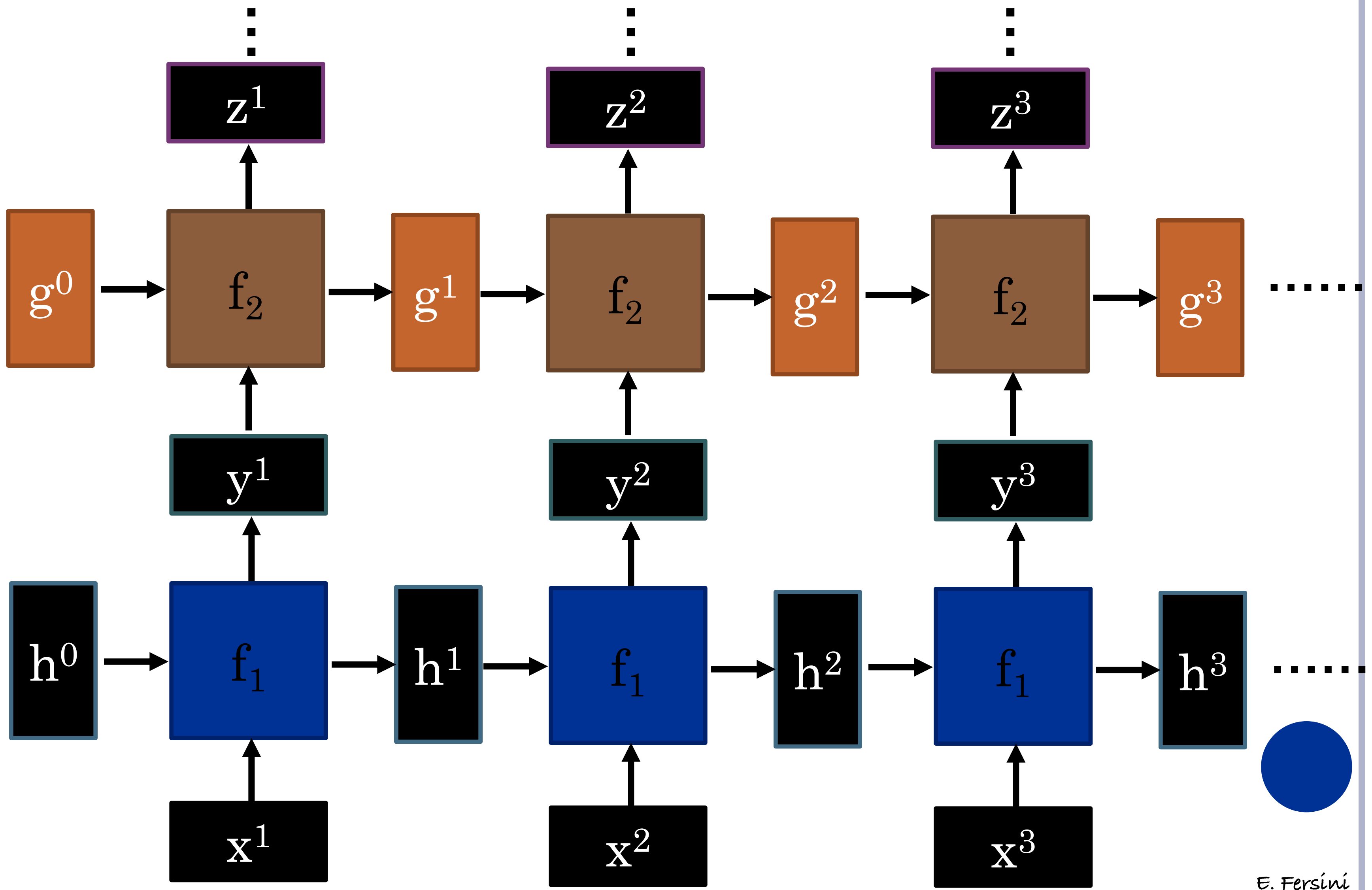
RNN Training

- The weights U, W and V are the same at each time step. However, when doing the backpropagation:

  1. The cross entropy error is first computed using the current output and the actual output (for all the time steps)
  2. For the unrolled network, the gradient is computed for each time step with respect to the weight parameter
  3. Since the weights rethe same for all the time steps the gradients can be combined together for all time steps
  4. The weights are then updated.

Deep RNN

$h', y = f_1(h, x),\ g', z = f_2(g, y)$ …



E. Fersini

Problems with RNN

- When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a “memory” to it.
- Vanishing gradient problem.

- Solution: LSTM

READINGS

Gao, Yuan, and Dorota Glowacka. "Deep gate recurrent neural network." Asian conference on machine learning. 2016.