Advanced Methods in Text Analytics

Recurrent Neural Networks

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Language is a Temporal Phenomenon

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Language** is essentially a **signal over time**
  - Sounds flow to form semantically meaningful messages
  - Words flow to form sentences, sentences to form paragraphs, etc.
- Different language models (LMs) represent/use time differently
  - N-gram LMs? Predict next word based on previous n - 1 words
  - FNN-based LMs? Move input window over time
- This lecture, new type of model: **recurrent neural networks (RNNs)**
  - Represent time differently (quite explicitly)
  - Addresses some issues with FNN-based LMs like Bengio's (e.g. input size)
  - New challenges, e.g. gradient flow over time
- RNNs were **essentially replaced by the transformer architecture**
  - But their modeling approach important, may become relevant again (Gu et al. 2023)
  - We look at the transformer architecture in the next lecture slides

Outline

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

1. Recurrent Neural Networks
2. Training RNNs
3. The Encoder-Decoder Architecture

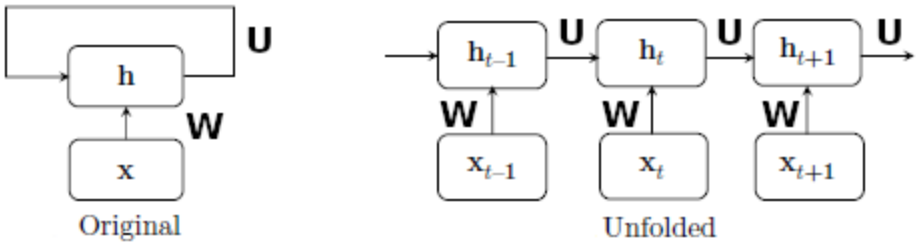
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

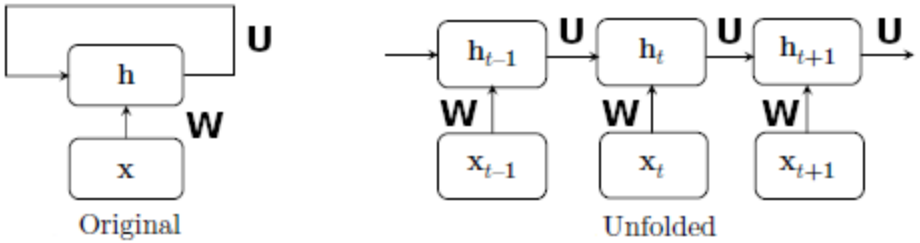
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Goal (in NLP):** learn a representation of a sequence
  - So, representation learning but for a given sequence, not a single word
  - Note a fully-connected neural network can also be used for this
- How? Let's look at *vanilla* RNNs
  - **Input:** sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ over n time steps (numbers, words, etc.)
  - **Requires:** initial hidden state $\boldsymbol{h}_0$
  - **Produces:** sequence of hidden states $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n$
- **Key difference** to fully-connected LMs like Bengio's LM (2003)
  - $\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b})$ wh. $\boldsymbol{W}, \boldsymbol{U}$ linear transformations, $\boldsymbol{b}$ bias, f is activation
  - I.e. hidden states computed from previous hidden state *and* current input

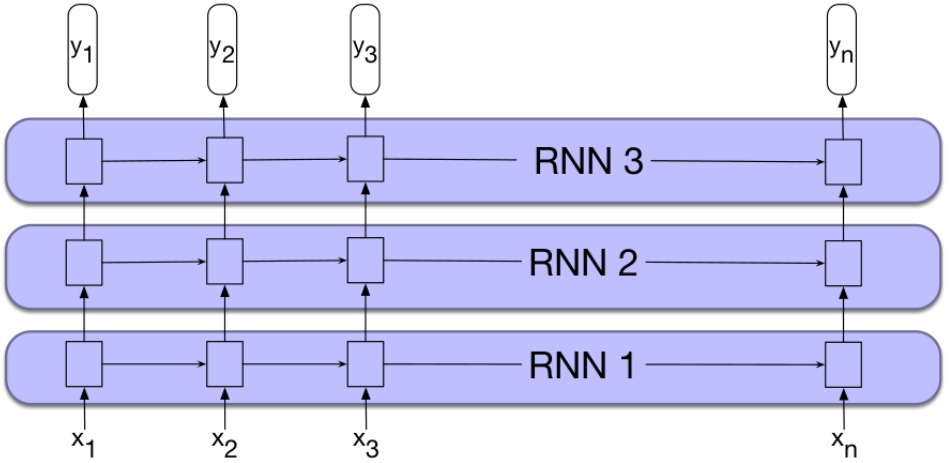


RNNs are Deep in Time

- ***W, U*** are **shared across time steps -->** RNNs are deep in time
  - Unfolded diagram *does not* show different layers! It's the same layer!



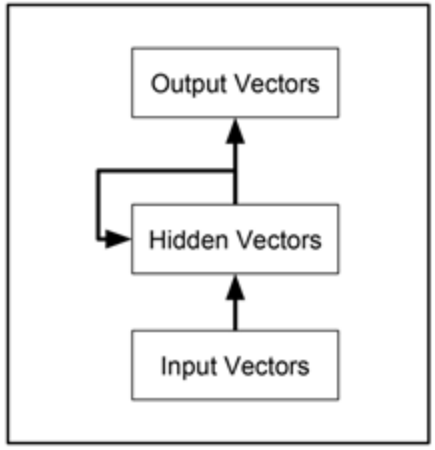
- But RNNs can also be **deep in the traditional sense** (stacked RNNs)
  - Input to layer *i* is sequence produced by layer *i – 1*
  - Weights are shared across time, not across layers

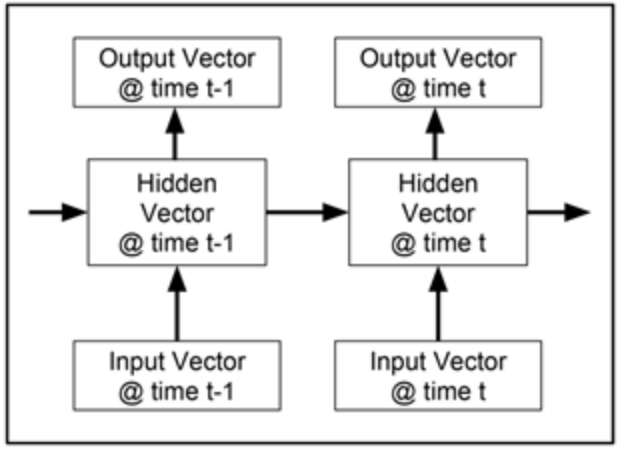


RNNs are Expressive (1)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- What about the **output** of the vanilla RNN?
  - What do I use as representation for the entire input sequence? Why?
  - In other words, how do we use an RNN for inference?
- We can design RNNs to do many tasks
  - **Design:** what units my network has, how they are computed
- For example, a network with an output unit $\boldsymbol{y}_t$ at each time step t
  - $\boldsymbol{h}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$
  - $\boldsymbol{y}_t = \sigma(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{b}_y)$





RNNs are Expressive (2)

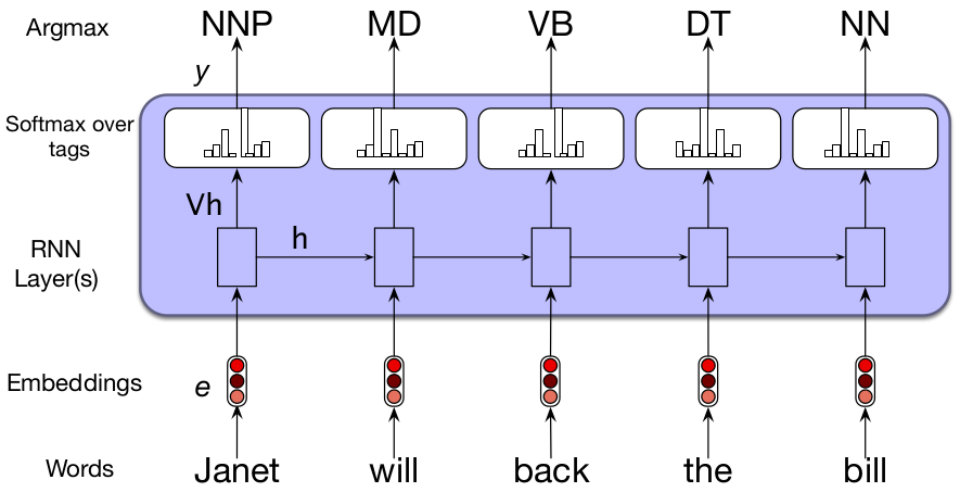
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Another type of RNN is often referred to as Jordan networks
  - $\boldsymbol{h}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{y}_{t-1} + \boldsymbol{b}_h)$
  - $\boldsymbol{y}_t = \sigma(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{b}_y)$
  - Notice the difference?
- In general, RNNs are very expressive (Chung et al. 2021)
  - Some tasks may require multiple output units, others just one
  - Some tasks may require sigmoid activations, other softmax
- Important: **weight sharing across time** acts as **model's memory/state**
  - **Intuition:** hidden state $\boldsymbol{h}_t$ encodes input sequence up until time step t
  - If $\boldsymbol{h}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$, $\boldsymbol{x}_t$ represents current input, $\boldsymbol{h}_{t-1}$ all previous ones
  - In principle can process inputs of any length (unlike LMs similar to Bengio's)
  - In practice, challenges emerge when input lengths are too long
- Let's have a look at **some RNN architectures for NLP**
  - I.e. input is sequence of words/tokens

RNNs for Token-level Classification

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

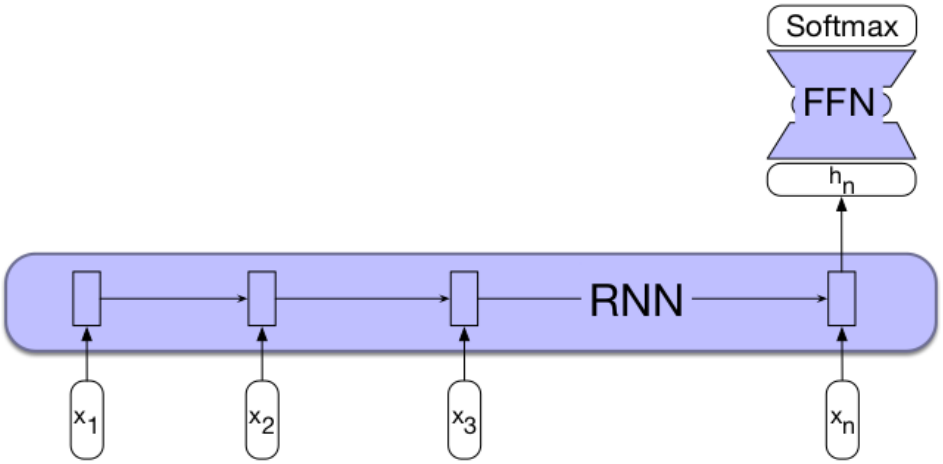
- Also referred to as sequence labeling
- **Goal:** predict a class for each token in the input sequence
  - **Named Entity Recognition (NER):** identify and classify named entities
    - **Input:** Jim bought 300 shares of Acme Corp. in 2006.
    - **Output:** $[\text{Jim}]_{\text{Person}}$ bought 300 shares of $[\text{Acme Corp.}]_{\text{Organization}}$ in $[2006]_{\text{Time}}$.
- Here's an **RNN for Part-of-Speech (POS) Tagging**
  - POS Tagging: each word can be a noun, verb, adverb, etc.
- More formally:
  - $\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$
  - $\boldsymbol{y}_t = softmax(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{b}_y)$



RNNs for Sequence-level Classification

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Also referred to as text classification.
- **Goal:** predict class for given sequence of text, e.g. a document
  - **Sentiment classification:** is this tweet's sentiment positive/negative/neutral?
  - **Spam detection:** is this email a spam or not?
- Here's an **RNN for sequence classification**



- We don't require outputs at each time step, **just one output at the end**
  - Alternatively, we could pool all hidden states together, use them as output
  - Again, no theories/principles to guide this choice, just known tricks

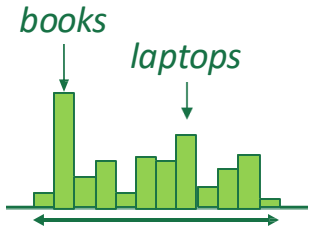
RNNs for Language Modeling (1)

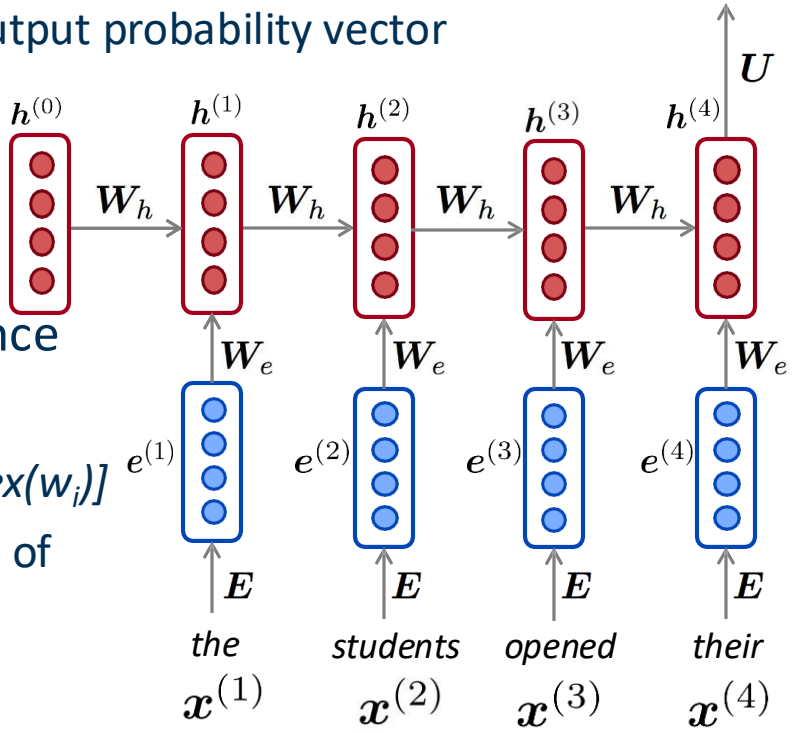
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Let's design an RNN for language modeling
  - **Recall the task:** *p("fish"|"Thanks for all the")*
  - Such a model can compute $p(w_{1:n}) = \Pi_i\, p(w_i|w_{<i})$
- As with Bengio's LM, we represent **each word** by a ***learned*** **embedding**
  - For vocabulary *V*, let $\boldsymbol{x}_i \in R^{|V| \times 1}$ be the one-hot vector for word *i*
  - We learn embeddings of size *d*, stack them in matrix $\boldsymbol{E} \in R^{|V| \times d}$
  - Then, embedding for word *i* is obtained as follows: $\boldsymbol{e}_i = \boldsymbol{E}^T\boldsymbol{x}_i \in R^{d \times 1}$
- We can define the following **RNN-based LM**:
  - $\boldsymbol{h}_t = f(\boldsymbol{W}_e\boldsymbol{e}_t + \boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_h)$
  - $\boldsymbol{y}_t = softmax(\boldsymbol{U}\boldsymbol{h}_t + \boldsymbol{b}_y)$
- If hidden states are of size *h* and ***U*** projects to vocabulary space, what are the sizes of $\boldsymbol{W}_e$, $\boldsymbol{W}_h$ and ***U***?
  - $\boldsymbol{W}_e \in R^{h \times d}$, $\boldsymbol{W}_h \in R^{h \times h}$, $\boldsymbol{U} \in R^{|V| \times h}$
  - Thus, $\theta = \{\boldsymbol{E}, \boldsymbol{W}_e, \boldsymbol{W}_h, \boldsymbol{U}, \boldsymbol{b}_h, \boldsymbol{b}_y\}$ (typically size of *d* is the same as size of *h*)

RNNs for Language Modeling (2)

- Visually:
  - $\boldsymbol{x}_i \in R^{|V|}$ is one-hot vectors of word *i*
  - $\boldsymbol{e}^{(i)} = \boldsymbol{E}\boldsymbol{x}^{(i)}$ are word embeddings
  - $\boldsymbol{h}^{(i)} = f(\boldsymbol{W}_e\boldsymbol{e}^{(i)} + \boldsymbol{W}_h\boldsymbol{h}^{(i-1)} + \boldsymbol{b}_h)$ is hidden state at time step *i*
  - $\boldsymbol{y}^{(i)} = softmax(\boldsymbol{U}\boldsymbol{h}^{(i)} + \boldsymbol{b}_y) \in R^{|V|}$ is output probability vector
- Probability that word *k* follows at time step *i* is then given by *i*-th component of $\boldsymbol{y}^{(i)}$, *i.e.* $\boldsymbol{y}^{(i)}[k]$
- The probability of an entire sequence can be computed as follows:
  - $p(w_{1:n}) = \Pi_i\, p(w_i | w_{1:i-1}) = \Pi_i\, \boldsymbol{y}^{(i)}[index(w_i)]$ where $index(w_i)$ indicates position of word *i* in vocabulary *V*





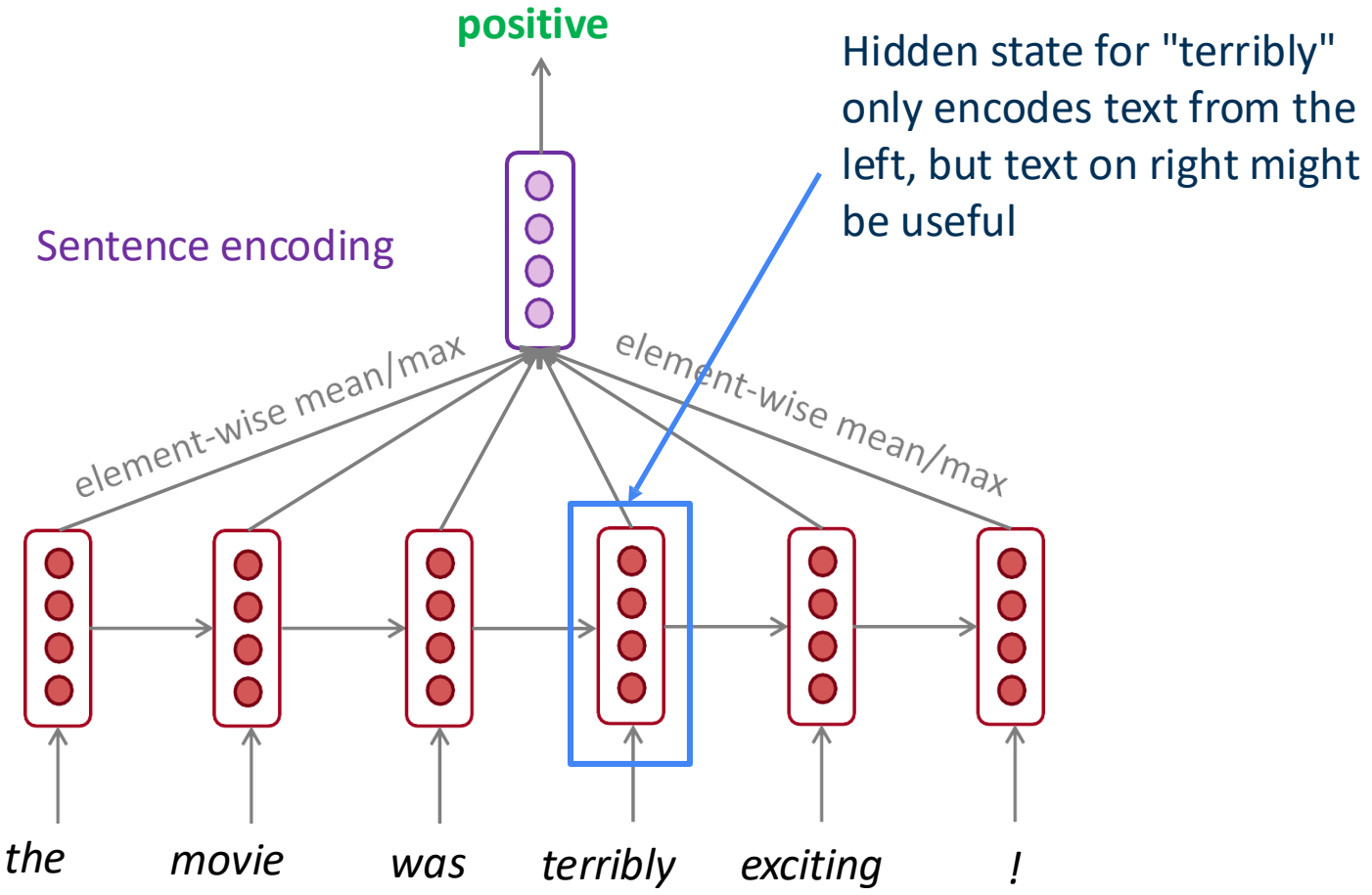
RNNs for Language Modeling (3)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Advantages** of RNN LMs:
  - Can process any input length
  - Model size independent of input length (unlike Bengio's LM and similar)
  - Shared weights across time allow information sharing from many steps back
- **Disadvantages**
  - RNN computations are slow, can't be generally parallelized ($\boldsymbol{h}_i$ depends on $\boldsymbol{h}_{i-1}$, which in turn depends on $\boldsymbol{h}_{i-2}$, etc.)
  - Information sharing across many steps back becomes difficult as input length grows larger
  - Challenges more clearly seen when looking at training approach
  - But first...
- Another important type of RNN
  - Bidirectional RNNs

Bidirectional RNNs (1)

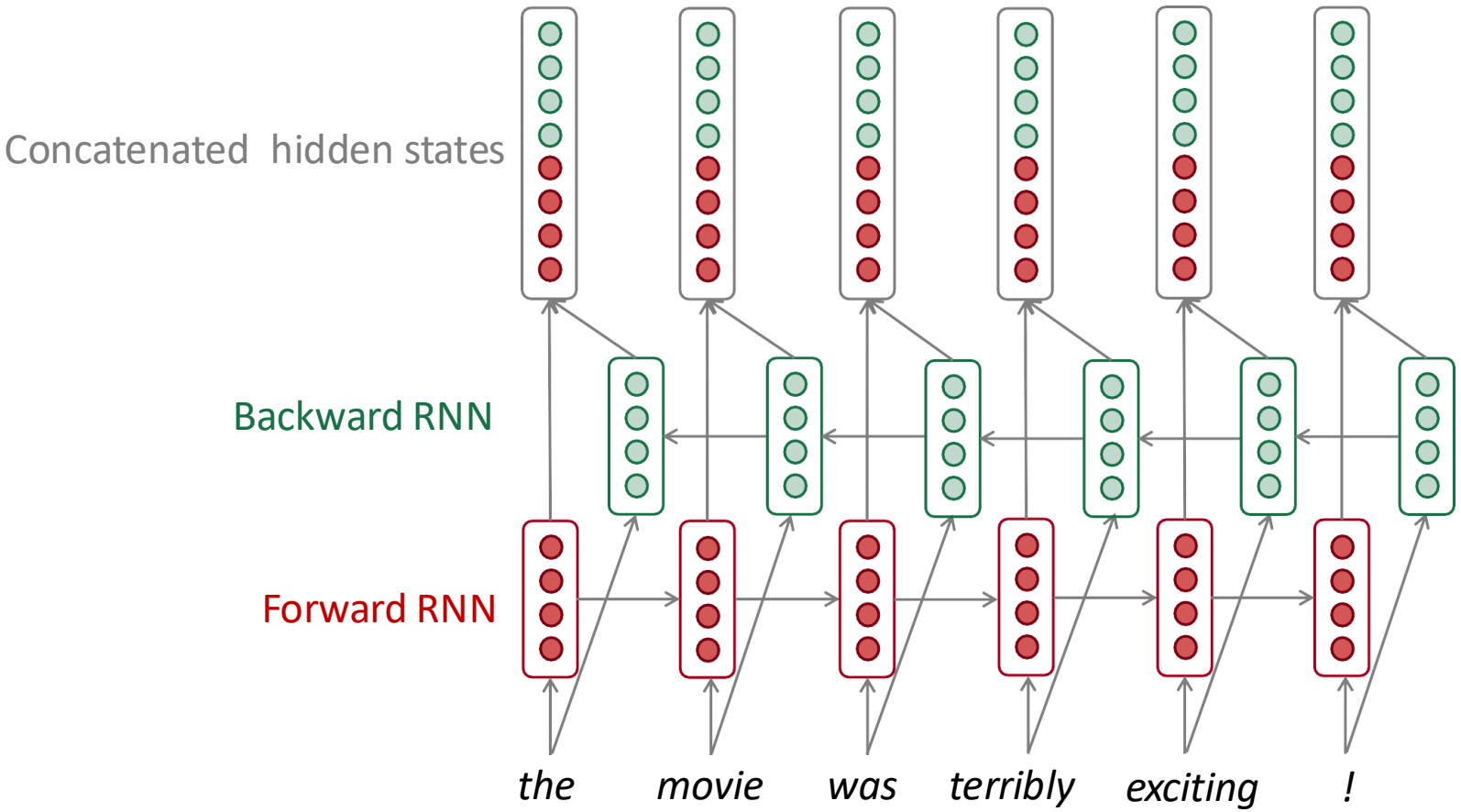
- They "read" input from left to right *and* right to left
- **Why?**



Bidirectional RNNs (2)

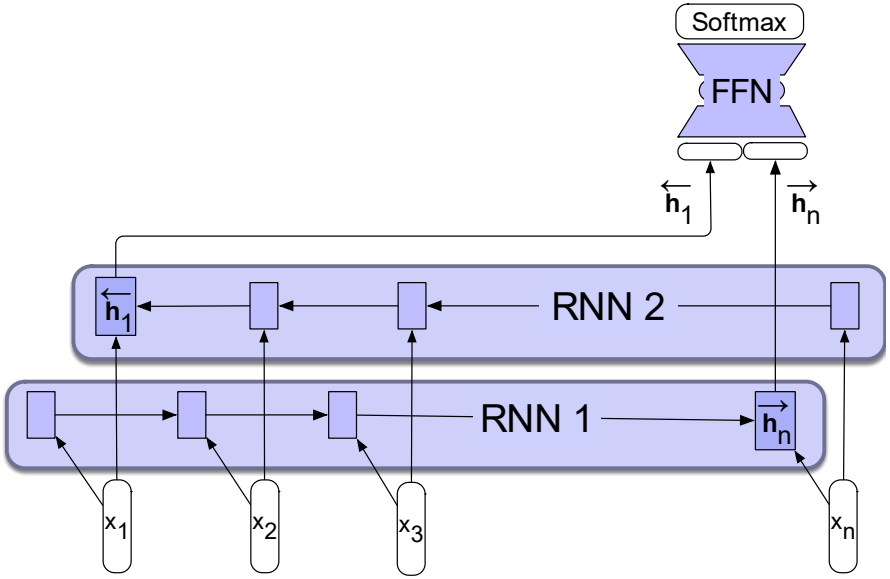
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **How?**
  - Keep another hidden state for new direction: $\boldsymbol{s}^{(i)} = f(\boldsymbol{W}_e \boldsymbol{e}_t + \boldsymbol{W}_s \boldsymbol{s}_{t+1} + \boldsymbol{b}_s)$



Bidirectional RNNs (3)

- Note that in previous slide, hidden states are concatenated to produce single representation of input sequence.
  - In practice, can aggregate them in any way, e.g. element-wise addition
- **How do we produce an output with the resulting hidden state?**
  - That depends on what we need for our task
  - E.g. for sequence classification



Summary: RNNs

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- They are: **designed to represent sequences**
  - Either a representation for each step, or for the entire sequence
- They are: **deep in time**
  - Hidden states at time step *t* encodes sequence until *t*
  - Shared parameters across time steps
  - Thus, hidden states act as model memory
- They are: **flexible/expressive**
  - Sequence labeling: one output per time step
  - Sequence classification: one output for each sequence
- They can be: **bidirectional**
  - Encode sequences based on previous and subsequent steps
- **Main advantage** for language modeling
  - Can encode infinitely long sequences
  - In practice, difficult to encode long sequences (more later)

Training RNNs

Training RNNs (1)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- As with other deep networks in NLP: **self-supervision + backprop**
- However, RNNs are deep in time
  - $\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1})$
  - That is, we produce a *representation for a different input at each time step*
- Thus, **we can compute a loss value at each time step!**
- Specifically, the cross-entropy at time step *t* is given by:

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

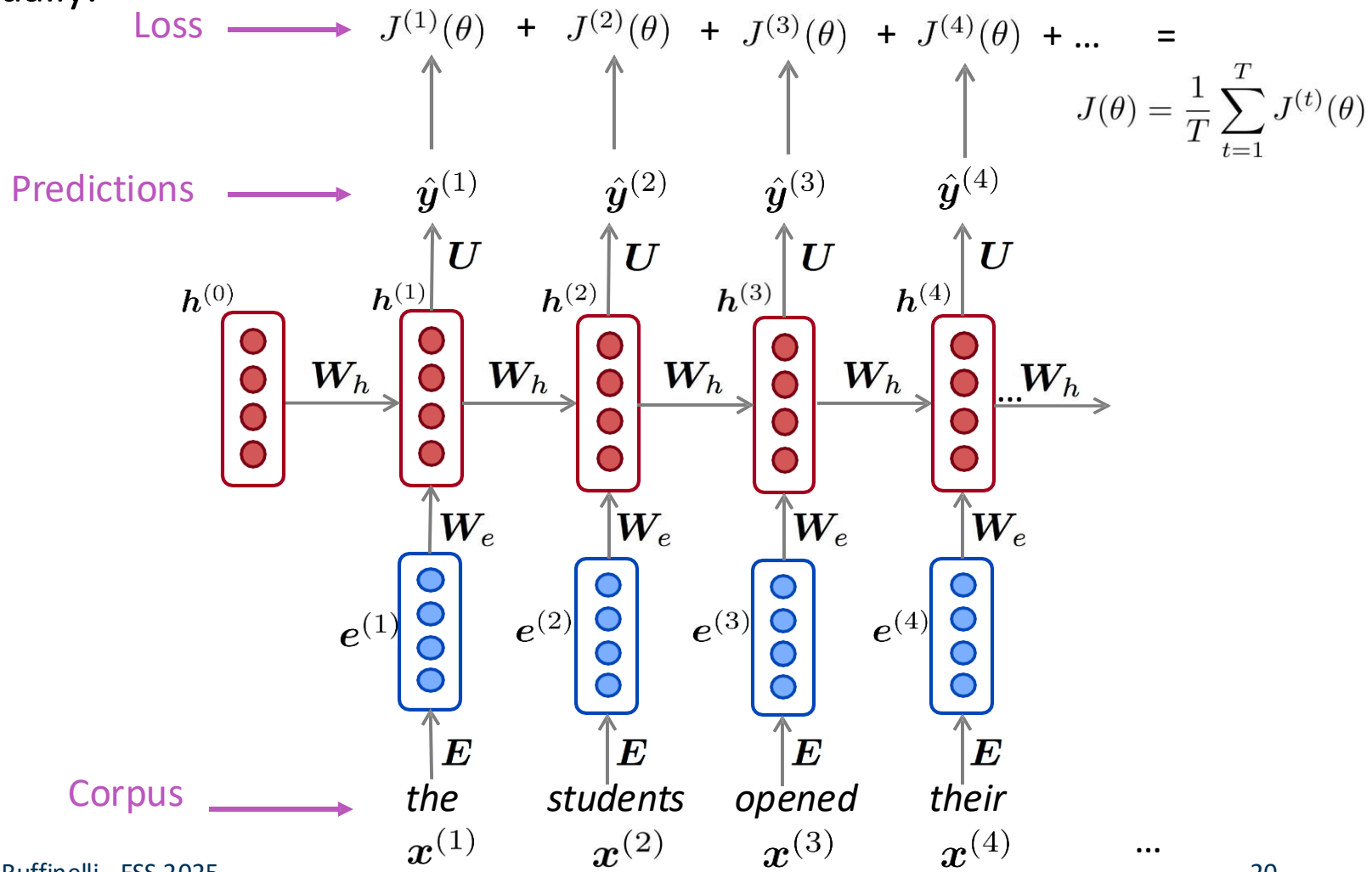
where $\mathbf{y}^{(t)}$ is probability of true next word at time step t, and $\hat{\mathbf{y}}^{(t)}$ is predicted probability for that next word

- We then average these losses over entire input/batch/corpus, denoted *T*

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

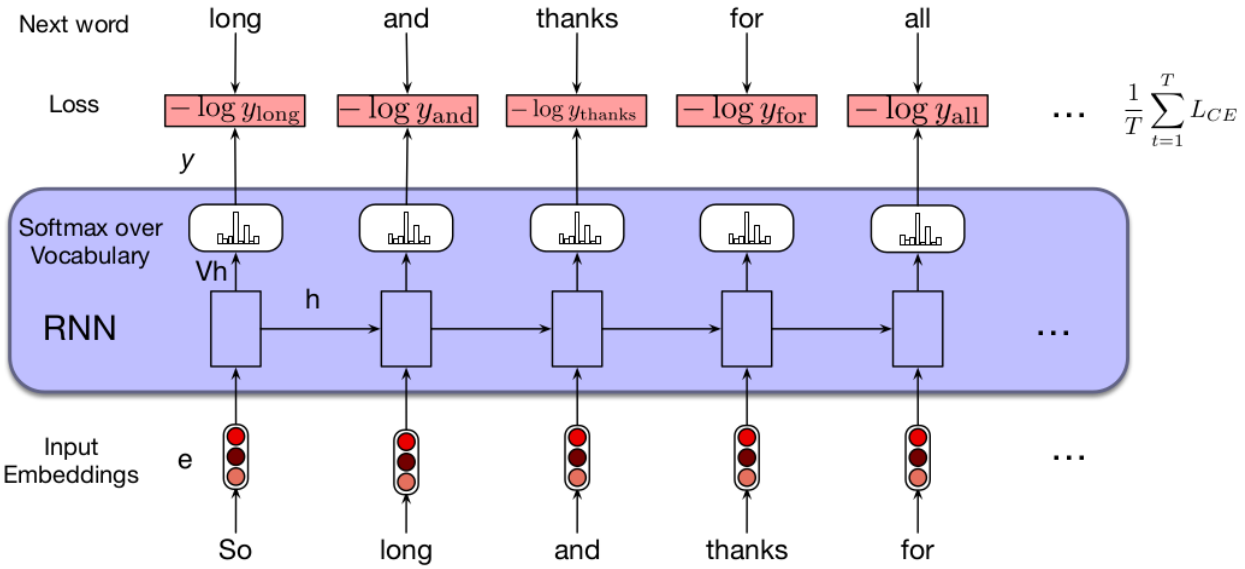
Training RNNs (2)

- Visually:



Training RNNs (3)

- What is the **correct/target word at each time step?**
  - We have this thanks to **self-supervision**
  - This is known as **teacher forcing:** feed model true next word from history



- This in contrast to **using word previously predicted by model**
  - That is what's done for **text generation** (more in later tutorials/lectures)

Weight Tying

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Take a standard RNN:
  - $\boldsymbol{x}_i \in R^{|V|}$ is one-hot vector of word i
  - $\boldsymbol{e}^{(i)} = \boldsymbol{E}\boldsymbol{x}^{(i)}$ are word embeddings where $\boldsymbol{E} \in R^{|V| \times d}$
  - $\boldsymbol{h}^{(i)} = f(\boldsymbol{W}_e\boldsymbol{e}^{(i)} + \boldsymbol{W}_h\boldsymbol{h}^{(i-1)} + \boldsymbol{b}_h)$ is hidden state at time step i
  - $\boldsymbol{y}^{(i)} = softmax(\boldsymbol{U}\boldsymbol{h}^{(i)} + \boldsymbol{b}_y) \in R^{|V|}$ is output probability vector
- Note that $\boldsymbol{E}$ and $\boldsymbol{U}$ are of the same size: $R^{|V| \times d}$ where d is embedding size
- **Weight tying** (Press et al. 2016, Inan et al. 2016)
  - Have $\boldsymbol{E}$ and $\boldsymbol{U}$ be the same matrix
  - I.e., same set of parameters are used both as word embeddings and for projection to final vocabulary space
- This **was observed to improve LM performance**
  - Added benefit: **number of parameters considerably reduced**
  - Remember $|V|$ is usually in the tens to hundreds of thousands
  - Thus, this is commonly done

The Challenge of Long-term Memory

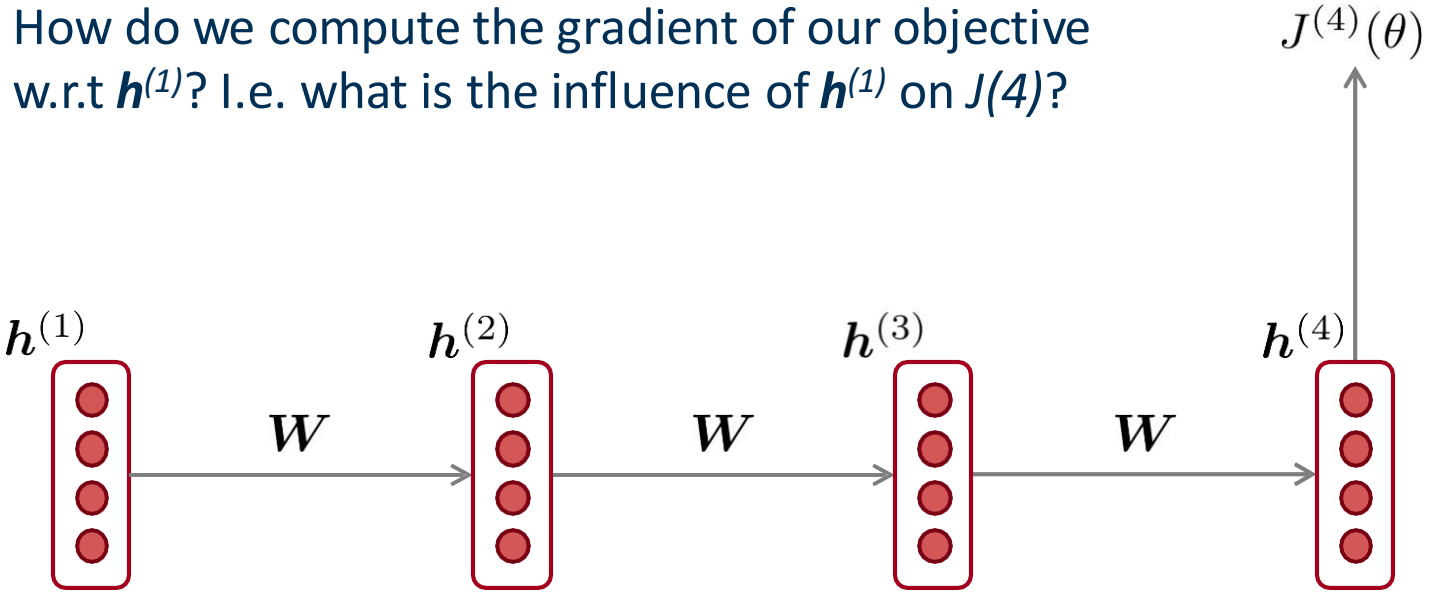
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Let's have a look at a **high-level example** for some intuition
- Take the following sentence:
  - *Daniel was the one who told us about the greatness of Michael I. Jordan.*
- Imagine an RNN encodes this sentence in its final hidden state $\boldsymbol{h}_t$
  - I.e. $\boldsymbol{h}_t$ is the representation of the sentence
- Ideally, $\boldsymbol{h}_t$ encodes enough information about the sentence to answer a variety of questions
  - Who told you about Michael I. Jordan?
  - How does Daniel feel about Michael I. Jordan?
- Note the distance in time between *Michael I. Jordan* and *Daniel*.
  - At last time step, $\boldsymbol{h}_t$ has to **"remember" Daniel was part of the sentence**
- Why can this be difficult?
  - The **vanishing gradient** problem, present in many deep models (including transformers), but nicely illustrated with RNNs (via Stanford slides)

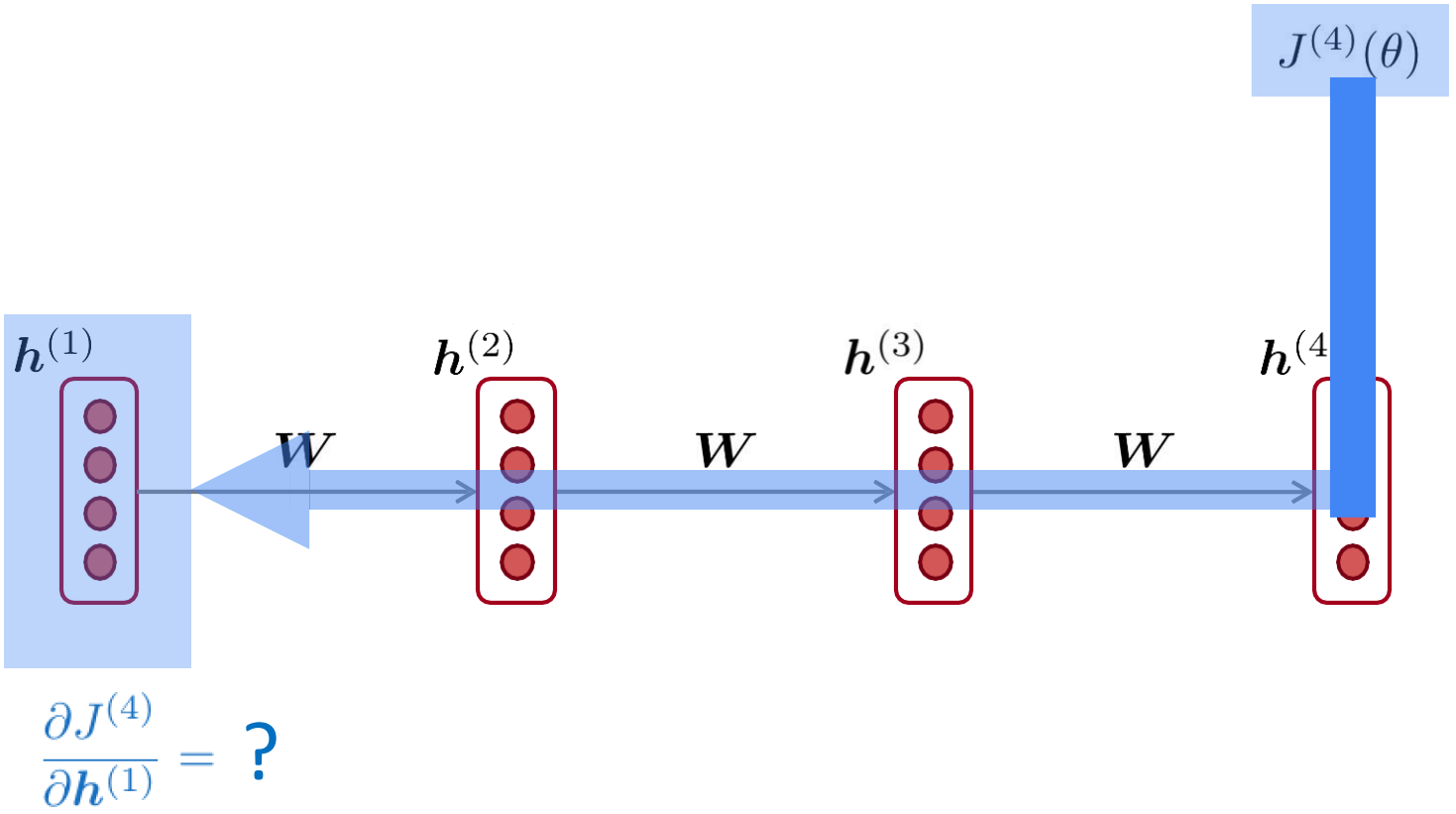
Vanishing Gradient Intuition

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

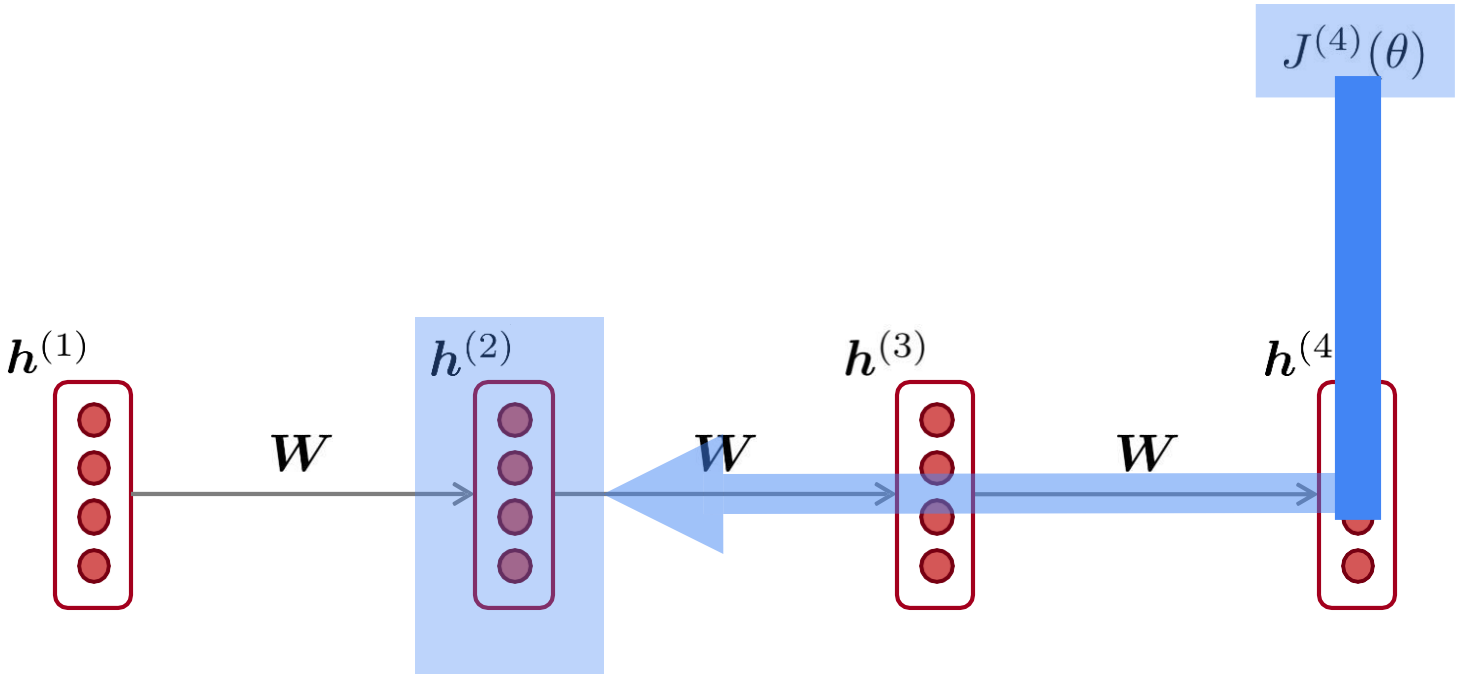
- Let *J(θ)* be our training objective
- We need to do backpropagation through time
- How do we compute the gradient of our objective w.r.t $\boldsymbol{h}^{(1)}$? I.e. what is the influence of $\boldsymbol{h}^{(1)}$ on *J(4)*?



Vanishing Gradient Intuition



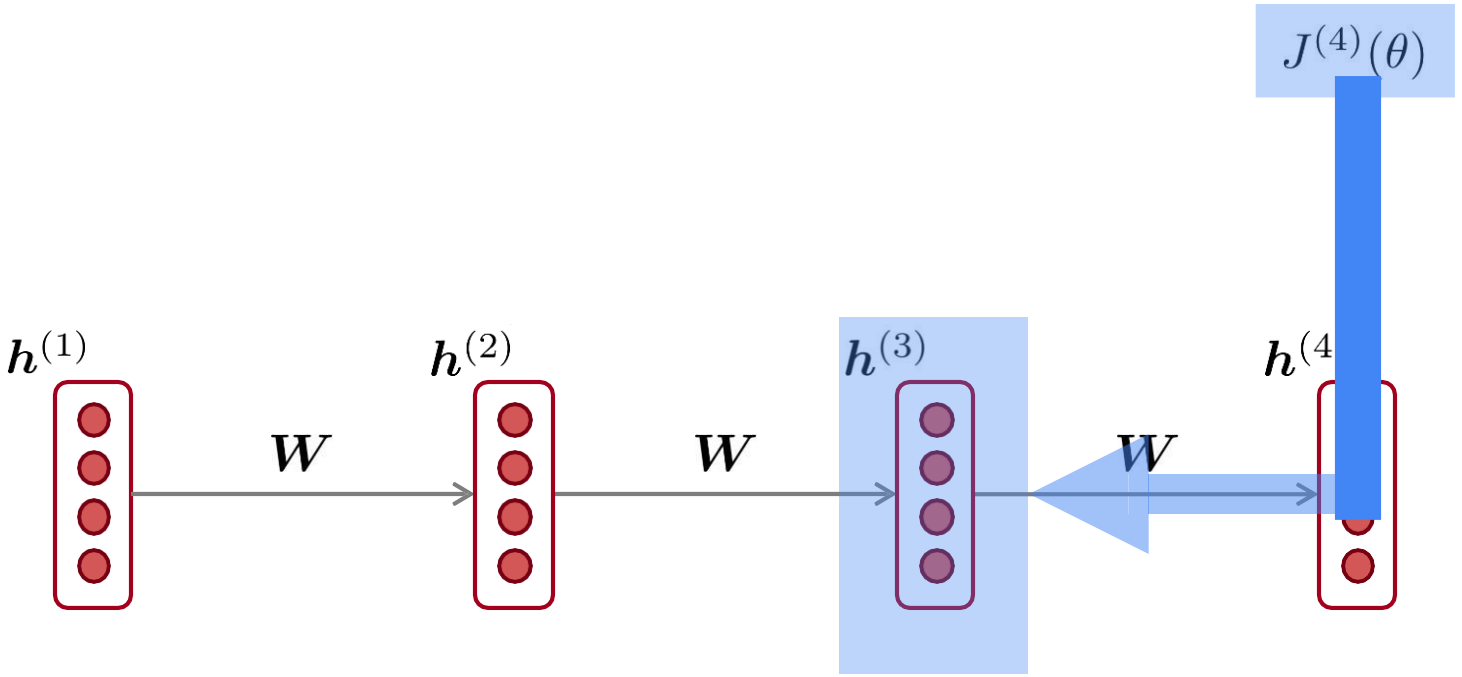
Vanishing Gradient Intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(2)}}$$

chain rule!

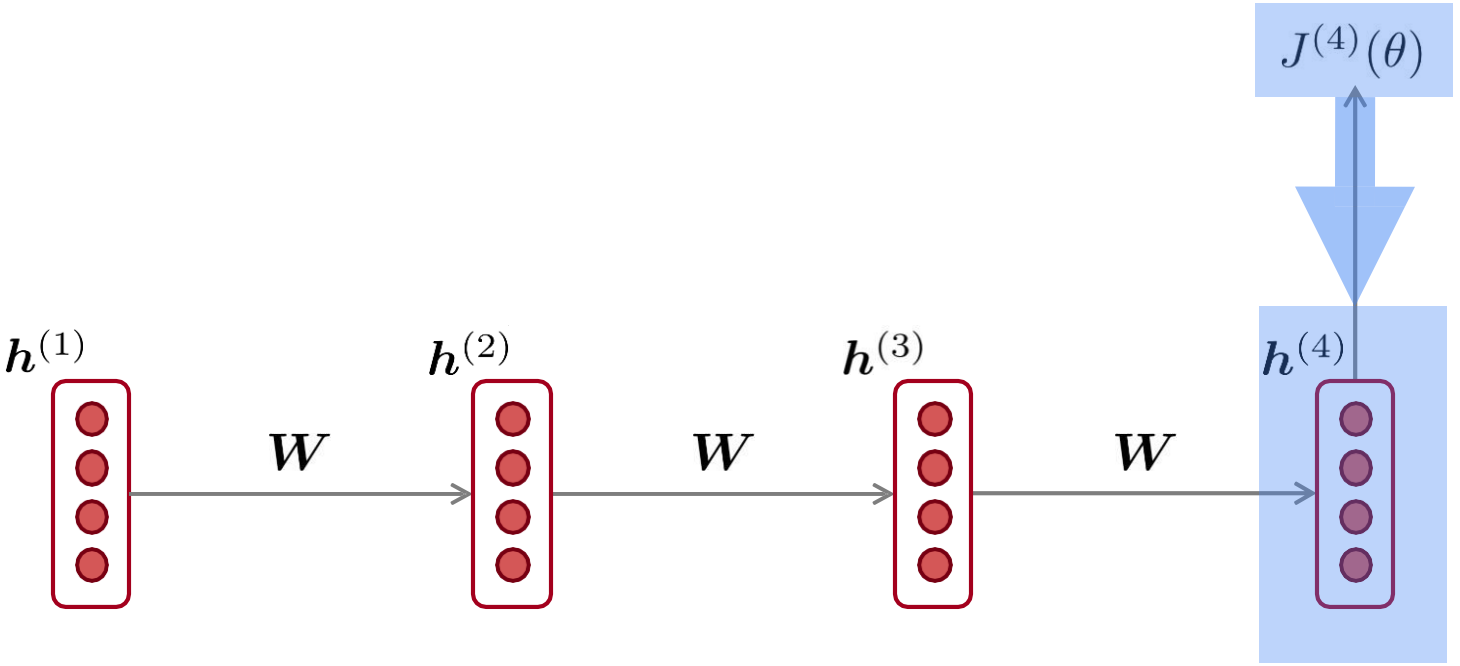
Vanishing Gradient Intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(3)}}$$

chain rule!

Vanishing Gradient Intuition



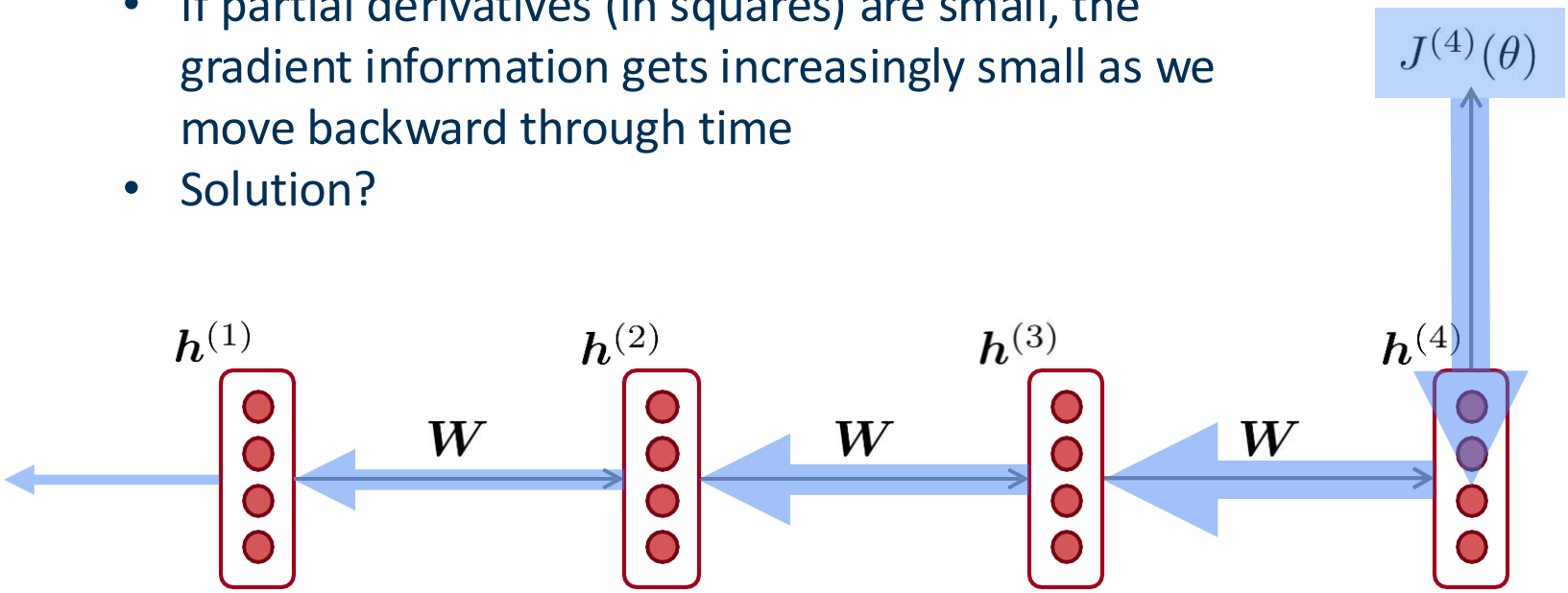
$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

chain rule!

Vanishing Gradient Intuition

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Recall: each operator has impact on gradient flow
- If partial derivatives (in squares) are small, the gradient information gets increasingly small as we move backward through time
- Solution?



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

Long Short-Term Memory (LSTM)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Most **common unit to build RNNs** (Hochreiter and Schmidhuber, 1997)
  - Though often "an LSTM" refers to an RNN with LSTM units
- Designed to allow network ***be able to*** **retain information from distant time steps**
- Divided this **"context management"** issue into two subproblems:
  - Removing information no longer needed from context
  - Adding information likely to be needed at later time steps
- **Importantly, LSTMs allow model to *learn* to manage context!**
- How?
  - They designed a **new unit** that is much more involved
  - It essentially **includes a new (context) layer**
  - It also includes **new parameters**, **gated operations**
  - It's a good example of how deep learning is about designing algebraic circuits for information flow

Intuition behind Gating Mechanisms

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Suppose you were adding two quantities, *a* and *b*, but you wanted to control how much of *b* gets into the sum:

$$a + \lambda b$$

- λ is a value between 0 and 1
- We say λ acts as a “switch” or a “gate”, because it **controls the amount of b that gets into the sum**
  - If λ is learnable, a model can learn how much to "open the gate"

LSTM Components

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Encodes RNN state** at time step *t* **in two vectors**: $\boldsymbol{c}_i$, $\boldsymbol{h}_i$
  - $\boldsymbol{c}_i$ is new "context layer", $\boldsymbol{h}_i$ is hidden state as before
  - So input of LSTM unit at time step *t* is $\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$ and $\boldsymbol{c}_{t-1}$
- Introduces **differentiable gating mechanisms:** smooth functions that simulate logical gates
  - **Forget gate**: decides which parts of the context should be forgotten due to new input
  - **Input/add gate**: decides how much of the current input $\boldsymbol{x}_i$ should be written to the context vector $\boldsymbol{c}_i$
  - **Output gate**: decides which parts of the context vector should be copied to the current hidden state
- Gate vectors are computed from current input $\boldsymbol{x}_i$ and previous state $\boldsymbol{h}_{i-1}$
  - Let's look at this circuit in more detail

An LSTM Unit

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

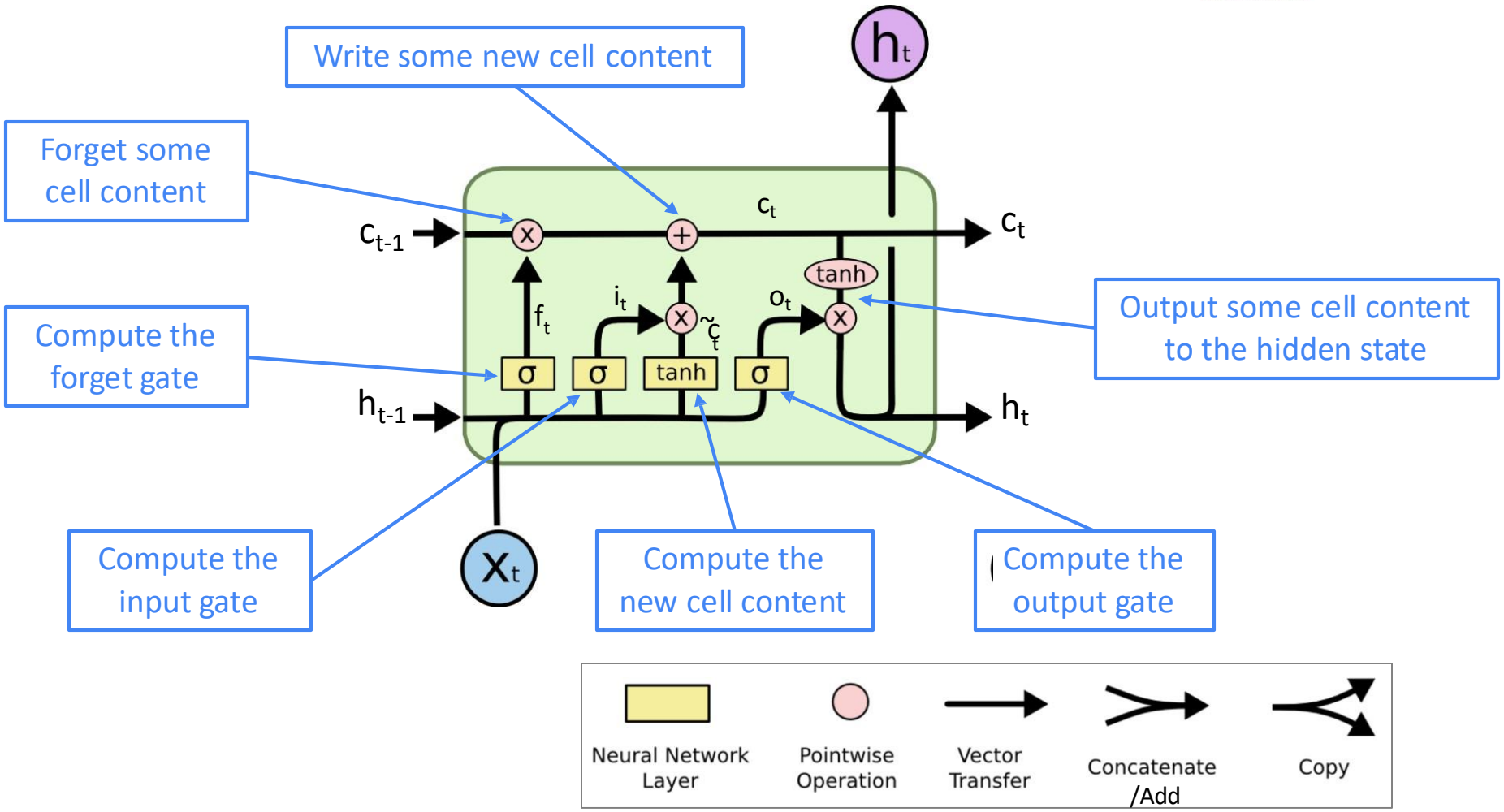


Image source

Gates: Common Design Pattern

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- All gates consist of:
  - A **linear layer**
  - A **sigmoid activation function**
  - An **element-wise (Hadamard) product** with the layer being gated
- The **sigmoid function** projects its inputs to the real interval [0, 1]
- **Combined with element-wise product**, it acts as a **soft binary mask**
  - Values in the layer being gated that align with values near 1 in the mask are *passed through nearly unchanged*
  - Values aligned with components of the mask that are closer to zero corresponding are *essentially erased*

Forget Gate

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Role:** delete information from context that is no longer needed
- How?
  - Mask $\mathbf{f}_t$ is created by adding up transformed versions of input and previous hidden step, then passed by a sigmoid function ($\boldsymbol{W}_f$, $\boldsymbol{U}_f$ are parameters)
  - Mask is then multiplied element-wise by context vector $\boldsymbol{c}_{t-1}$ to remove the information that is no longer required from context

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

Input/Add Gate

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- First we need to compute our hidden state:

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$

- **Goal of Add Gate:** select information to add to current context
- How?
  - Generate mask to select the information to add to the current context ($\boldsymbol{W}_i$, $\boldsymbol{U}_i$ are parameters)

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

  - Add this to the modified context vector to get the new context vector

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

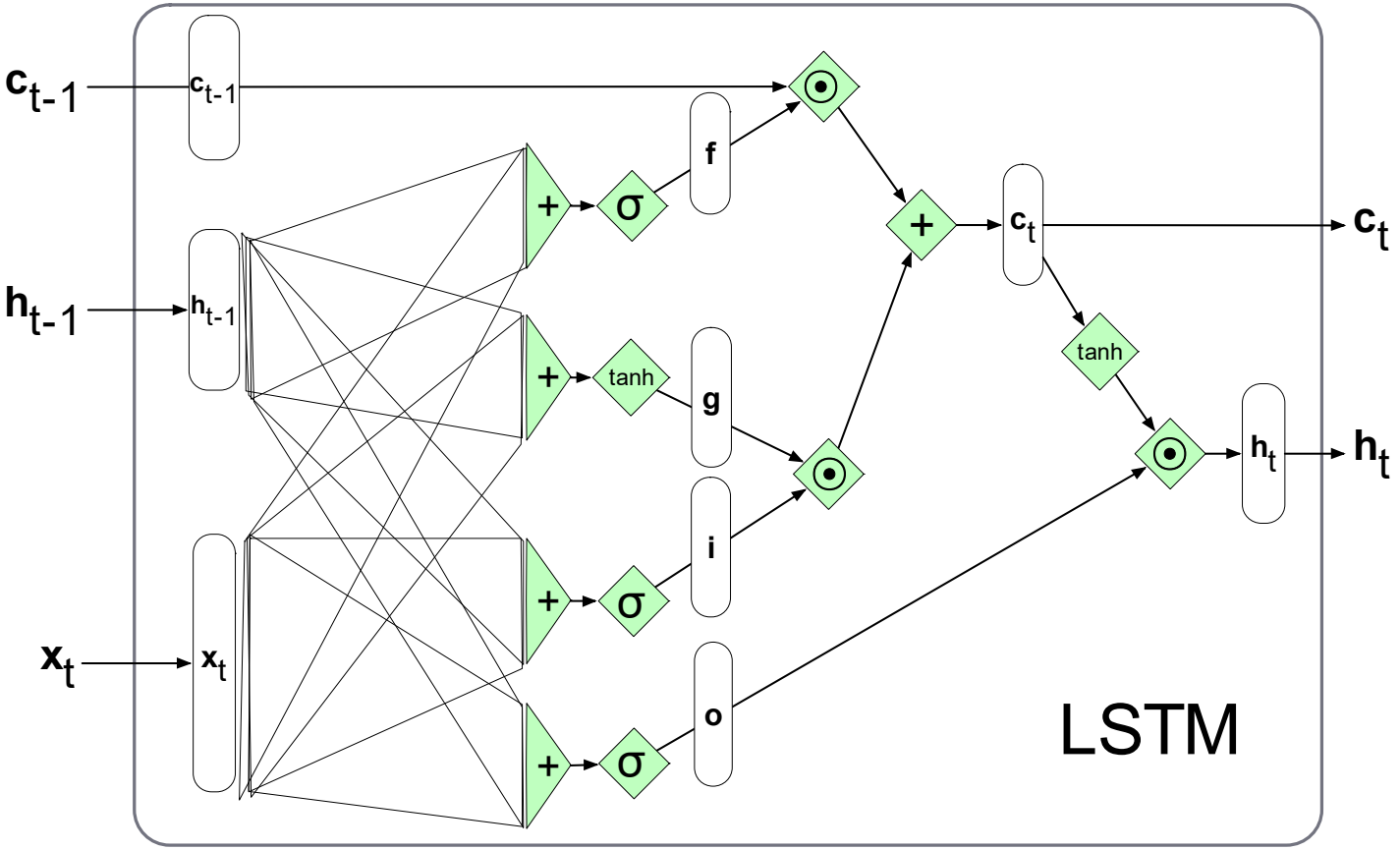
Output gate

- **Goal:** select information to use from current hidden state
- How?
  - As before, we create a mask to select information from the hidden state that is required for the current hidden state

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Bringing It All Together

- A super nice illustration of the operations!



Summary: Training RNNs

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Backpropagation through time**
  - Each time step depends on previous time steps
  - We can compute a loss value at each time step
  - We average losses over time
  - Then backpropagate w.r.t. different time steps, not different layers
- **Long-term memory** is known to be **a challenge**
  - Illustrated by **vanishing gradient problem**
  - Addressed with different designs, e.g. LSTMs
- **LSTM:** long short-term memory
  - Adds new context layer to RNNs
  - Designed based on gating mechanisms
  - Allows model to **learn to use different context at each time step**
  - Recently shown to achieve comparable performance when scaled to the size of LLMs

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

The Encoder-Decoder Architecture

So far...

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **LMs** with **fully-connected neural network**
  - Sees entire input at same time, but limited to fixed maximum input size
  - Why? Model size (i.e. number of parameters) grows with input size
- **RNNs** for various **NLP tasks**, e.g. token-level classification, LMs
  - Sees input one (time) step at a time, allows arbitrarily long input sequences
  - Shares parameters across time, so model size independent of input size
  - Hidden states encode sequence "seen so far"
  - In practice, difficult to "remember" what was seen long ago
- Important aspect of architecture/tasks seen so far: **fixed-size outputs**
  - **Token-level:** for each input token, models produce *corresponding* output, e.g. labels, next word, etc.
  - **Sequence-level:** for entire input sequence, produce *single* output, e.g. spam or not
- What about tasks where **output** must be sequence of **arbitrary length**?
  - Can we design an RNN for such a task?

Sequence-to-sequence Tasks

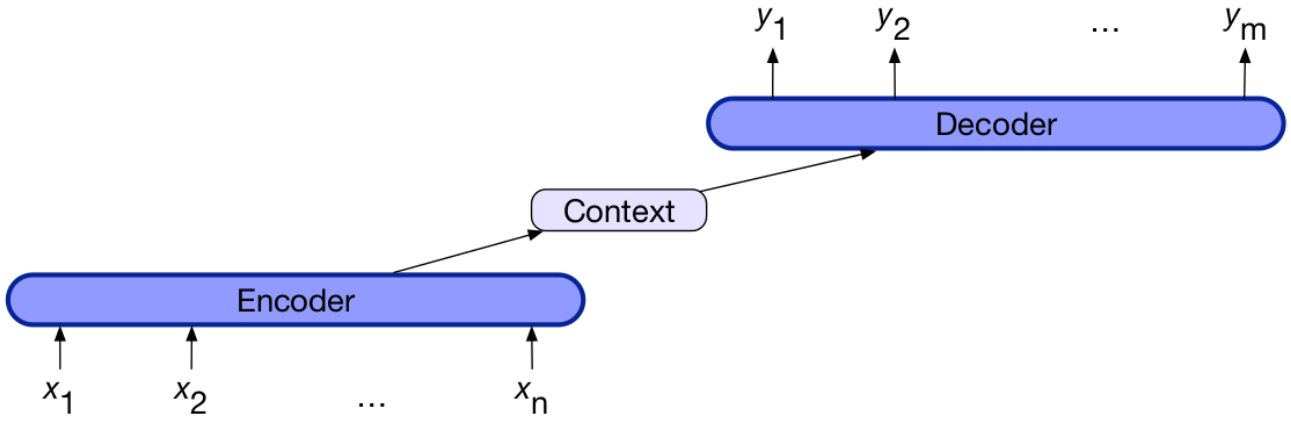
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Machine translation**
  - **Input:** J'aimerais apprendre le français un jour
  - **Output:** I would like to learn French one day
- **Summarization**
  - **Input:** I asked Google Gemini to summarize the starting note from Google's CEO found in this article into a single sentence
  - **Output:** "The author believes the transition to AI is the most profound technological shift ever, with vast potential to improve lives and accelerate progress."
- **Dialogue Systems/Question Answering**
  - **Input:** I then said to Gemini "That was nice! How did you learn to do that?"
  - **Output:** "I learn and improve in a few ways! Here are the key ones:
    1. Massive Dataset Training: …
    2. Deep Learning Algorithms: …
    3. Feedback and Refinement: … "

Sequence-to-sequence Models

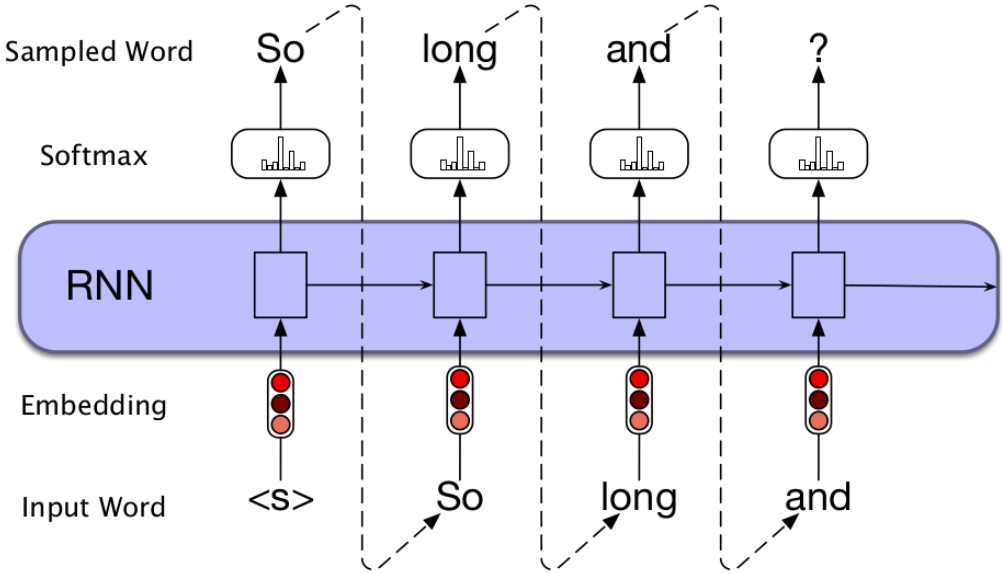
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Often abbreviated **seq2seq**
  - Input sequence and output sequence can have different lengths
  - Mapping between input and output tokens can be very indirect
- **Common architecture for seq2seq: encoder-decoder**
  - **Encoder:** from input seq. to seq. of *contextualized representations*
  - **Context vector:** it is the vector that represents entire input sequence given sequence of contextualized representations
  - **Decoder:** given context vector, generates sequence of hidden states, from which corresponding sequence of output states can be obtained



Text Generation with RNNs

- To discuss seq2seq with RNNs, we first need to discuss text generation
  - The **decoder** part of a seq2seq model **generates sequences** (of tokens/text)
- **Autoregressive generation:** generate sequence of tokens by repeatedly sampling the next token *conditioned on previously predicted tokens*
- **Can be broken down into three steps:**
  1. Sample word from model (softmax) distribution with start-of-sentence token <s> as input
  - Use sampled word in previous step as input, sample next word as in Step 1
  - Repeat Steps 1 and 2 until end-of-sentence token </s> is sampled.

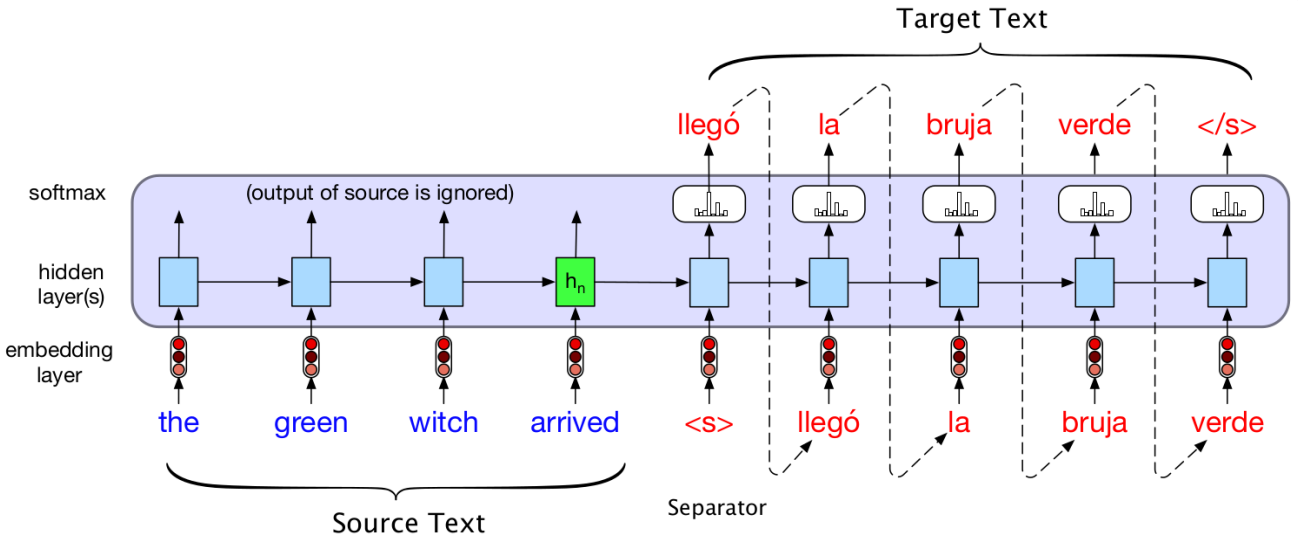


seq2seq with RNNs (1)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Recall: hidden state $\boldsymbol{h}_t$ at time step *t* represents all *t –1* steps seen so far
  - So, take sentence *"So long and thanks for all the fish"*
  - $\boldsymbol{h}_1$ encodes *"<s>"*, $\boldsymbol{h}_2$ encodes *"<s> so"*, $\boldsymbol{h}_3$ encodes "<s> so long", etc.
- Let's do **machine translation**, a common seq2seq task
  - **Input:** sequence of tokens in source language
  - **Output:** corresponding sequence of tokens in target language
- How to train an RNN for machine translation (seq2seq task)?
  - Use a **sentence separator token**, commonly denoted by [SEP]
  - **Concatenate input and target sequences**, feed it to model
- Let's see how this formatting of the machine translation task for RNNs looks like!

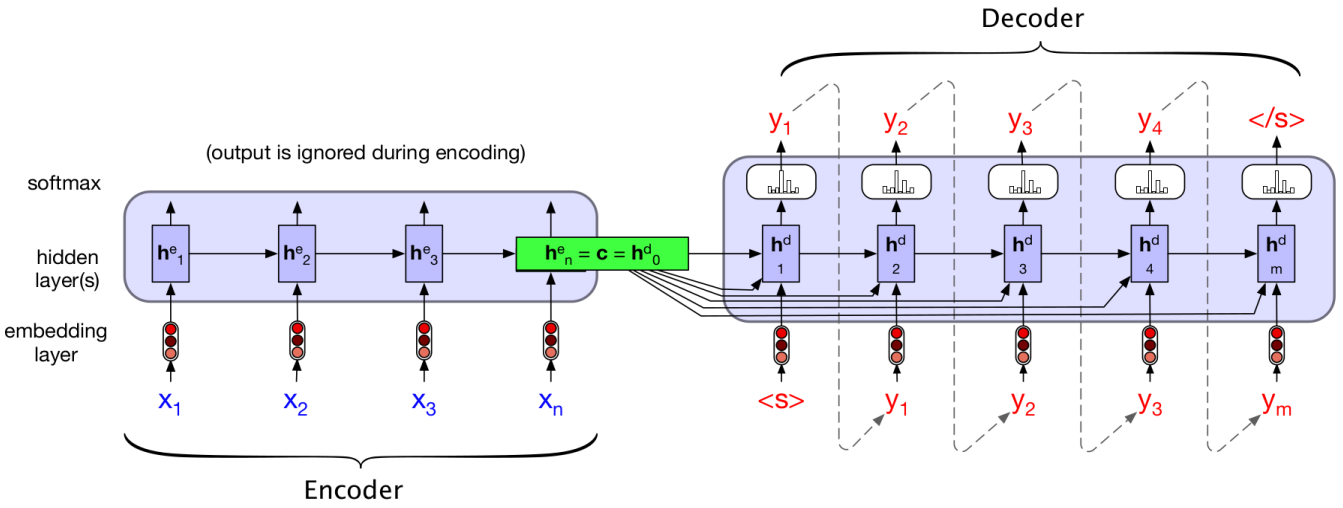
seq2seq with RNNs (2)



- Note: we use a **single RNN**
  - *<s>* acts as sentence separator in image above
  - Must be different from other special tokens, e.g. start-of-sentence
- **Question:** is this an encoder-decoder architecture?
  - Yes, $\boldsymbol{h}_n$ encodes entire input sequence, **acts as context vector *c***
  - It is then fed as input to "rest of network", which acts as decoder
  - Special token *<s>* marks start of decoding, autoregressive generation

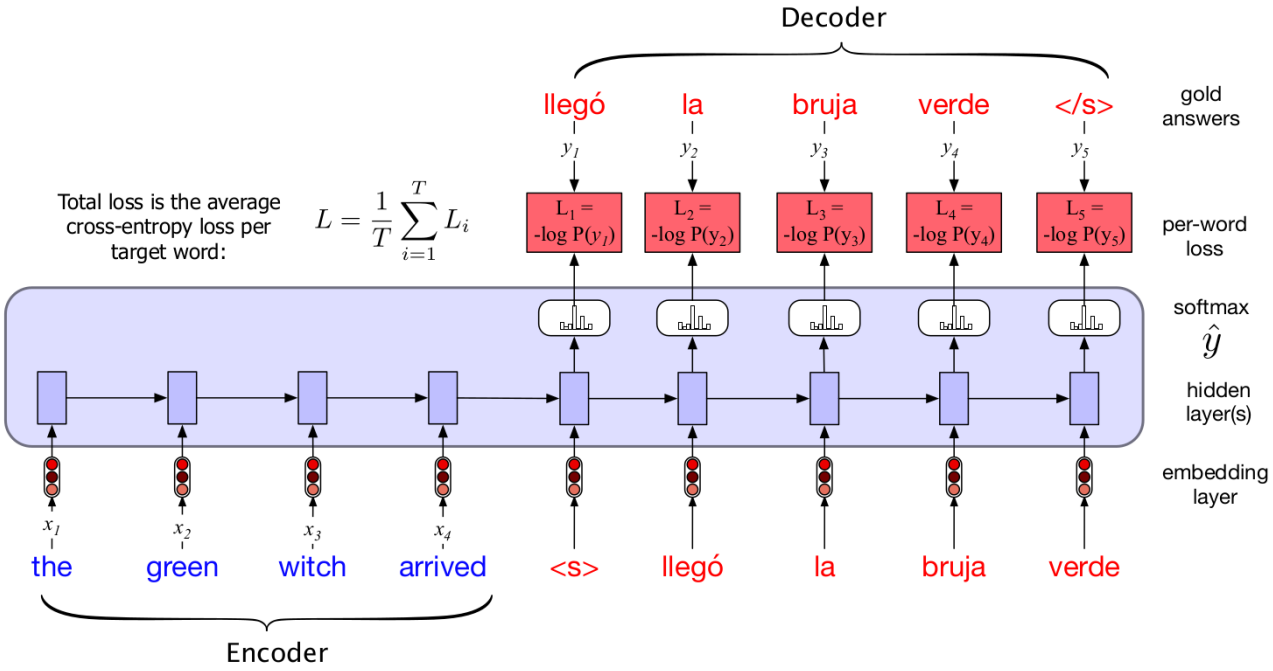
seq2seq with RNNs (3)

- In more detail:
  - $\boldsymbol{h}^e_t$ denotes encoder hidden states at time step *t*, $\boldsymbol{h}^d_t$ decoder hidden states
  - Context vector ***c*** *can be* added to all decoder hidden states
  - That is: $\boldsymbol{h}^d_t = f(y_{t-1}, \boldsymbol{h}^d_{t-1}, \boldsymbol{c})$
- **In practice**:
  - Stacked RNNs commonly used, encoder/decoder can be different networks
  - **Encoder** can be a biLSTM (**bidirectional LSTM**), reading input in both directions is useful for encoding, **not possible for decoding**



Training an Encoder-Decoder RNN

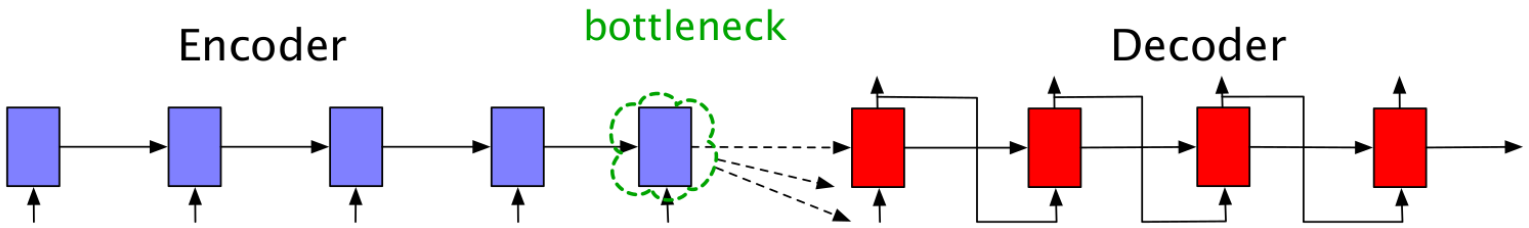
- Examples: (source sequence, target sequence)
  - So, supervised, not just self-supervised
- Training then as before, except:
  - We focus on average loss of target words (i.e. output must be correct)
  - We apply **teacher forcing** to train the decoder



Limitations with Encoder-Decoder

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Context vector ***c*** must encode entire input sequence
  - For a given architecture, ***c*** is of a given fixed size
  - But inputs can be of any length, ***c*** must encode them regardless
  - Hidden states are known to encode more of recent context than farther
- Thus, ***c* acts as a bottleneck**
  - It can be shown that for some tasks/input length, ***c*** can't encode input well



- The **attention mechanism** was **designed to address this**
  - Key to transformer architecture (next lecture)

Attention

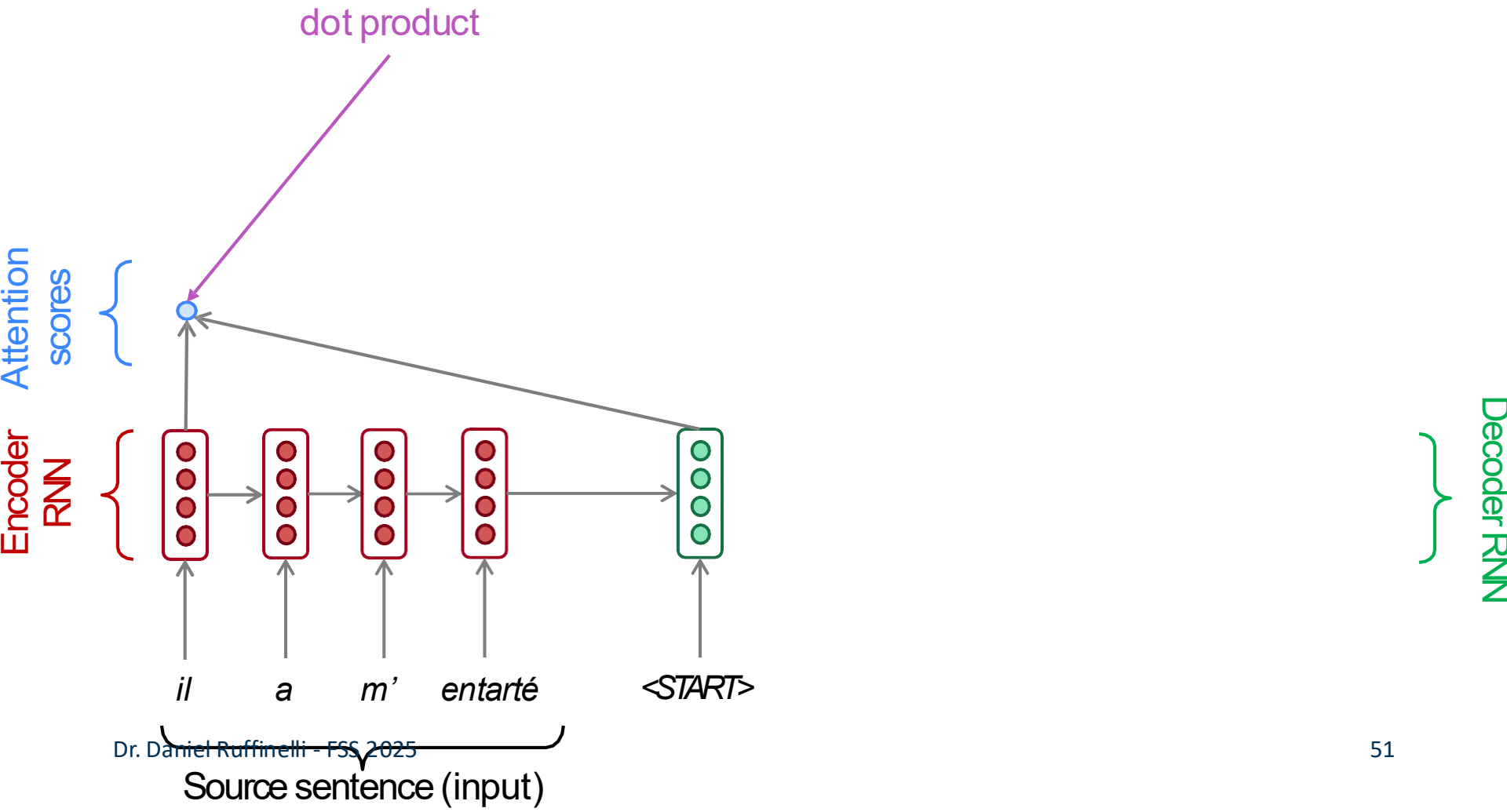
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Question:** why not allow *decoder* to access *encoder* hidden states directly?
  - Would this be cheating in some way? No.
- This **would allow decoder** to get **information from all time steps**
  - It was accomplished by redefining context vector ***c***
- Formally, let $\boldsymbol{c} = f(\boldsymbol{h}^e_1, \ldots, \boldsymbol{h}^e_t)$ be a function of all encoder hidden states
  - Most commonly, *f* is a weighted sum of encoder hidden states
  - Weights determine how much to "attend to" each hidden state
- **How we compute weights** usually determines **type of attention**
  - Most common approach: **dot-product attention**
  - **Attention scores:** dot-product between encoder and decoder hidden state
  - **Attention weights:** normalized attention scores

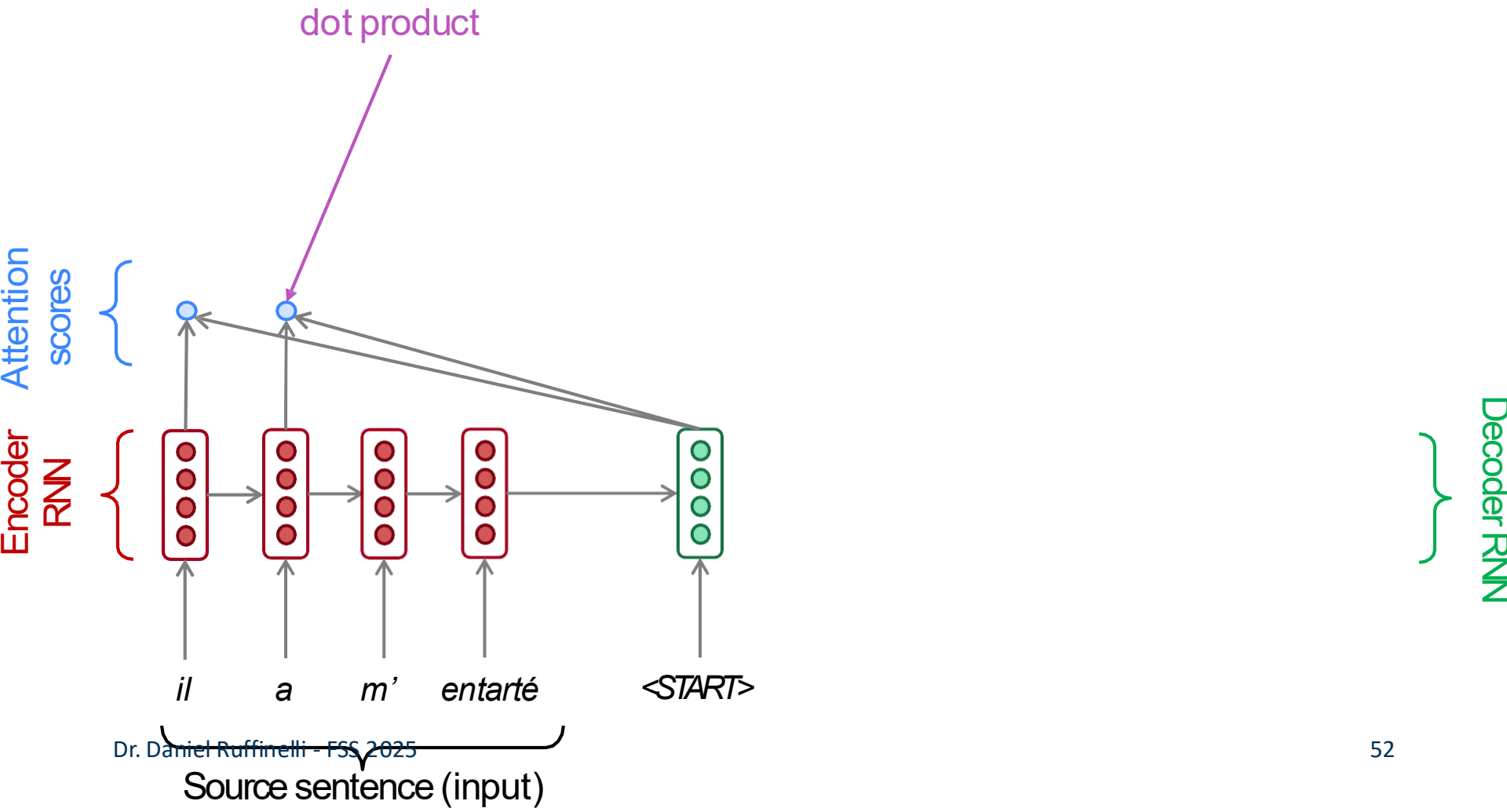
$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)) \\ &= \frac{\exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))} \end{aligned}$$

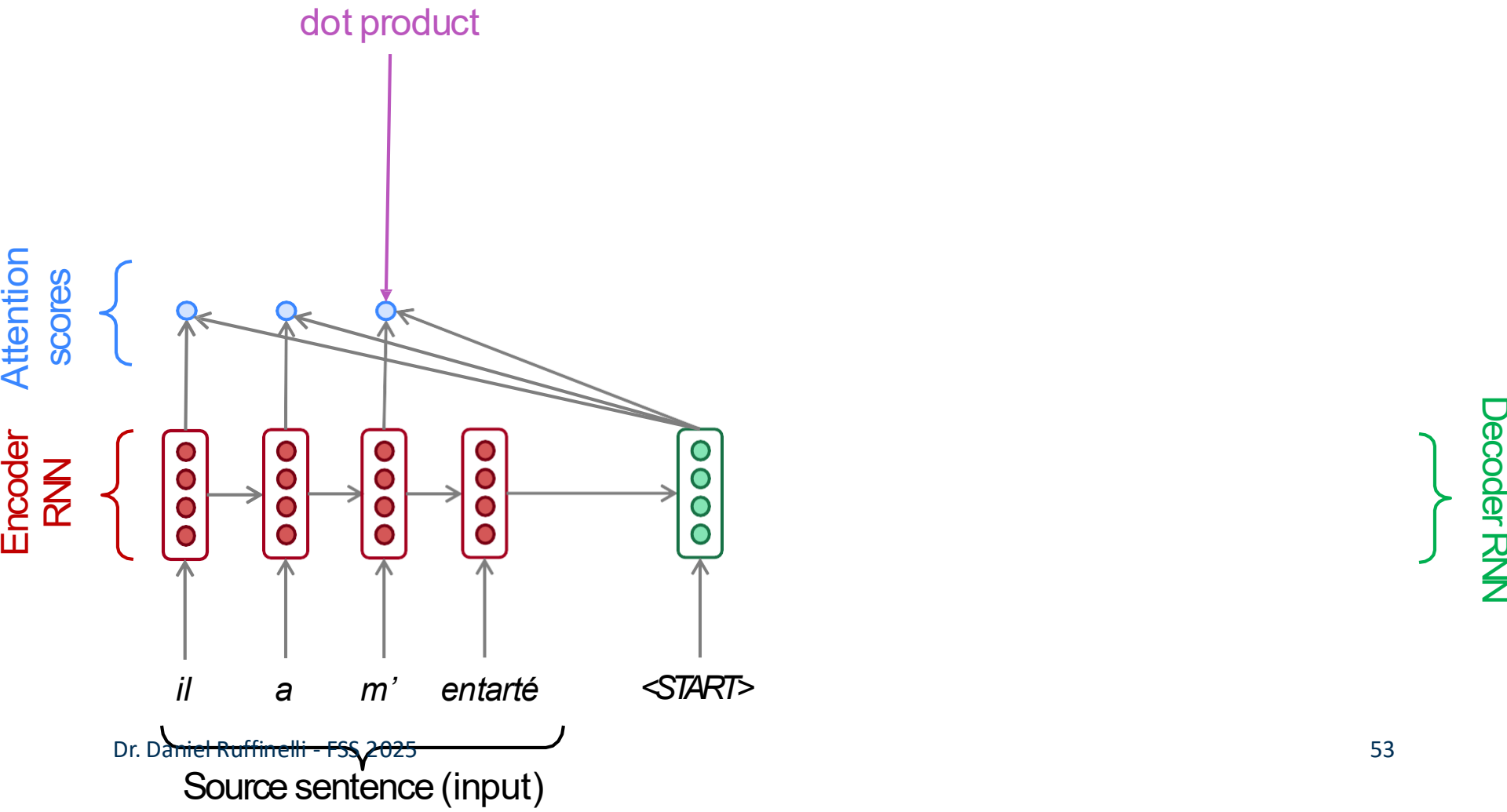
Encoder-decoder with Attention



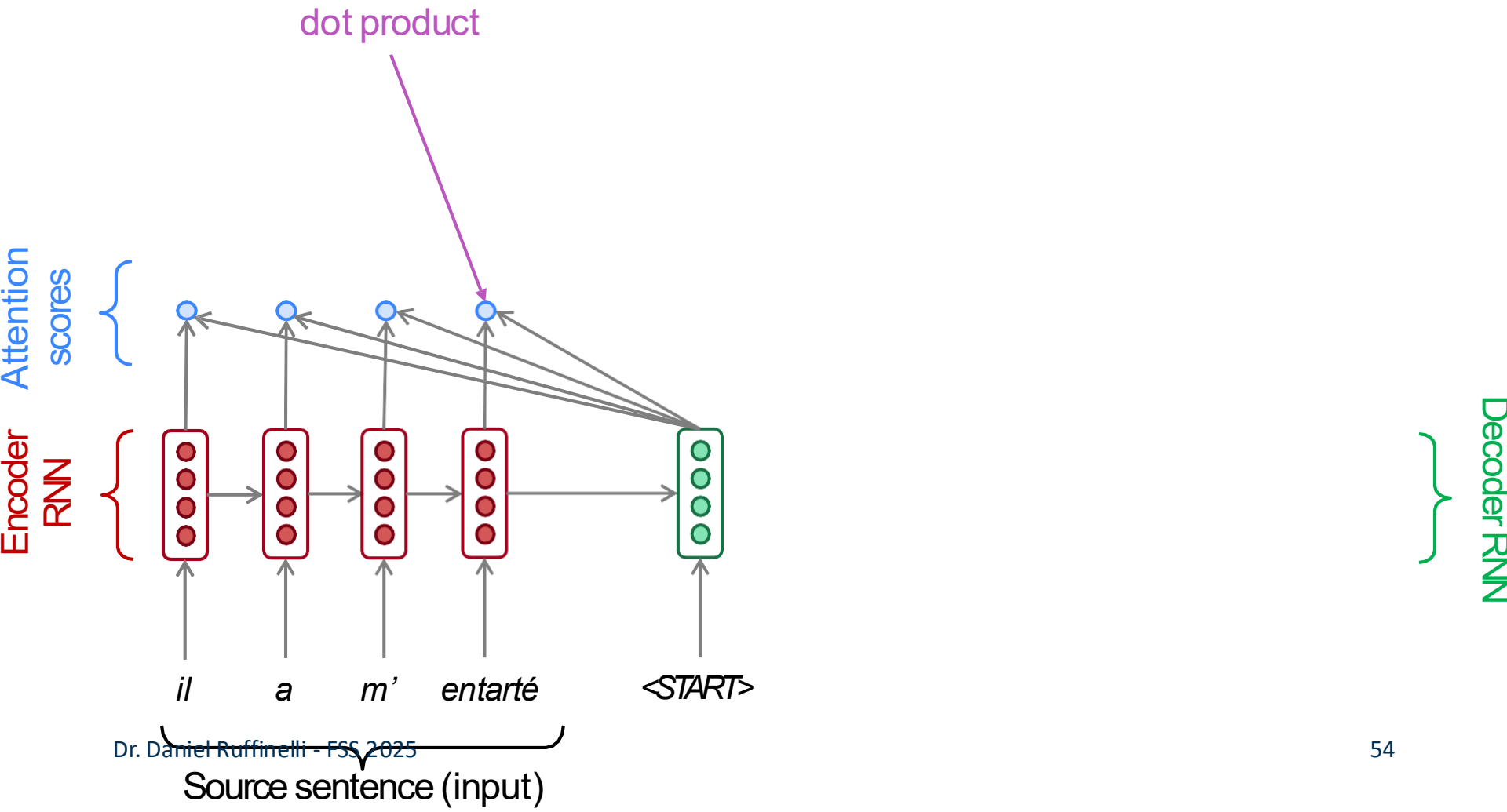
Encoder-decoder with Attention



Encoder-decoder with Attention

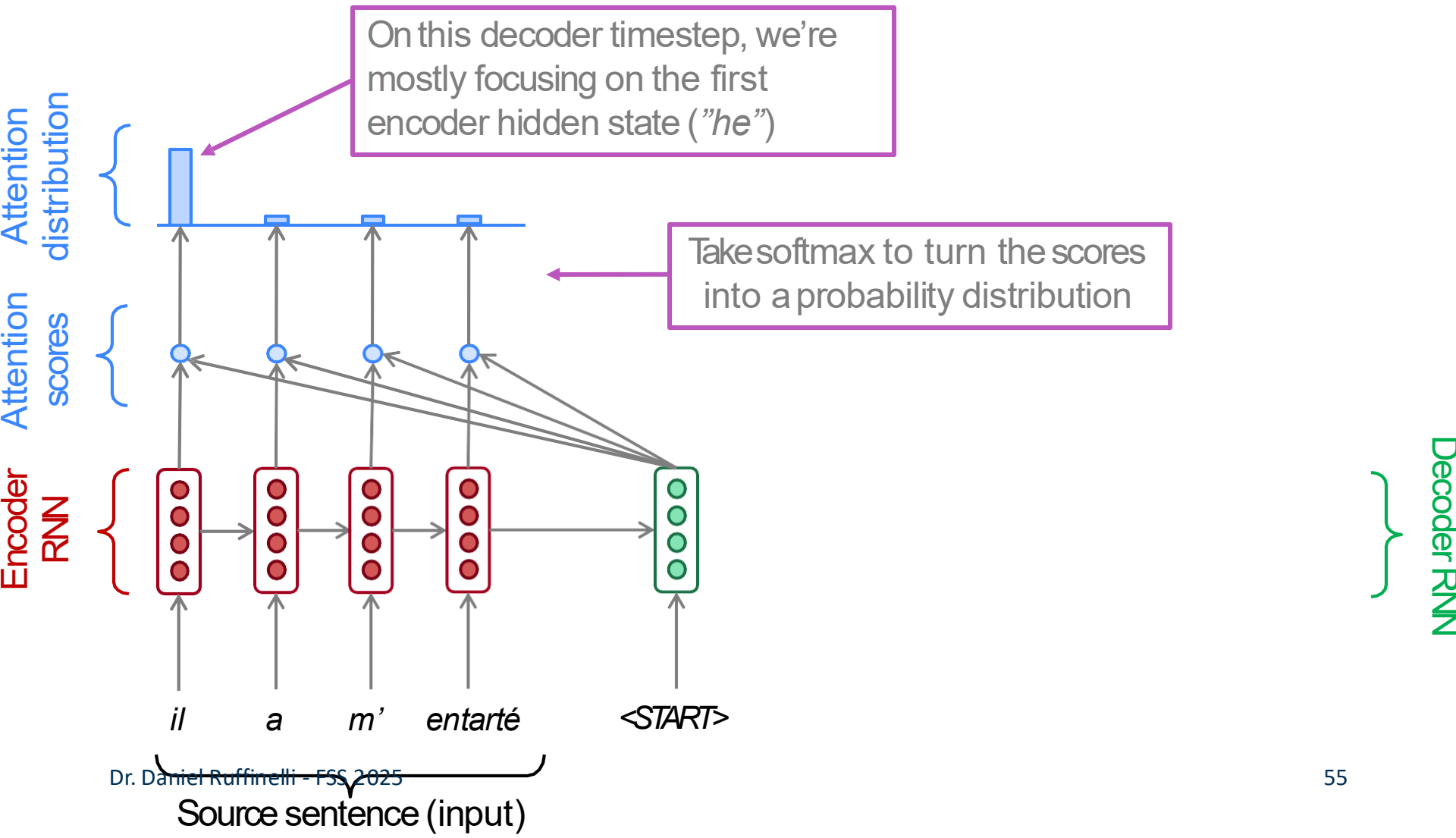


Encoder-decoder with Attention

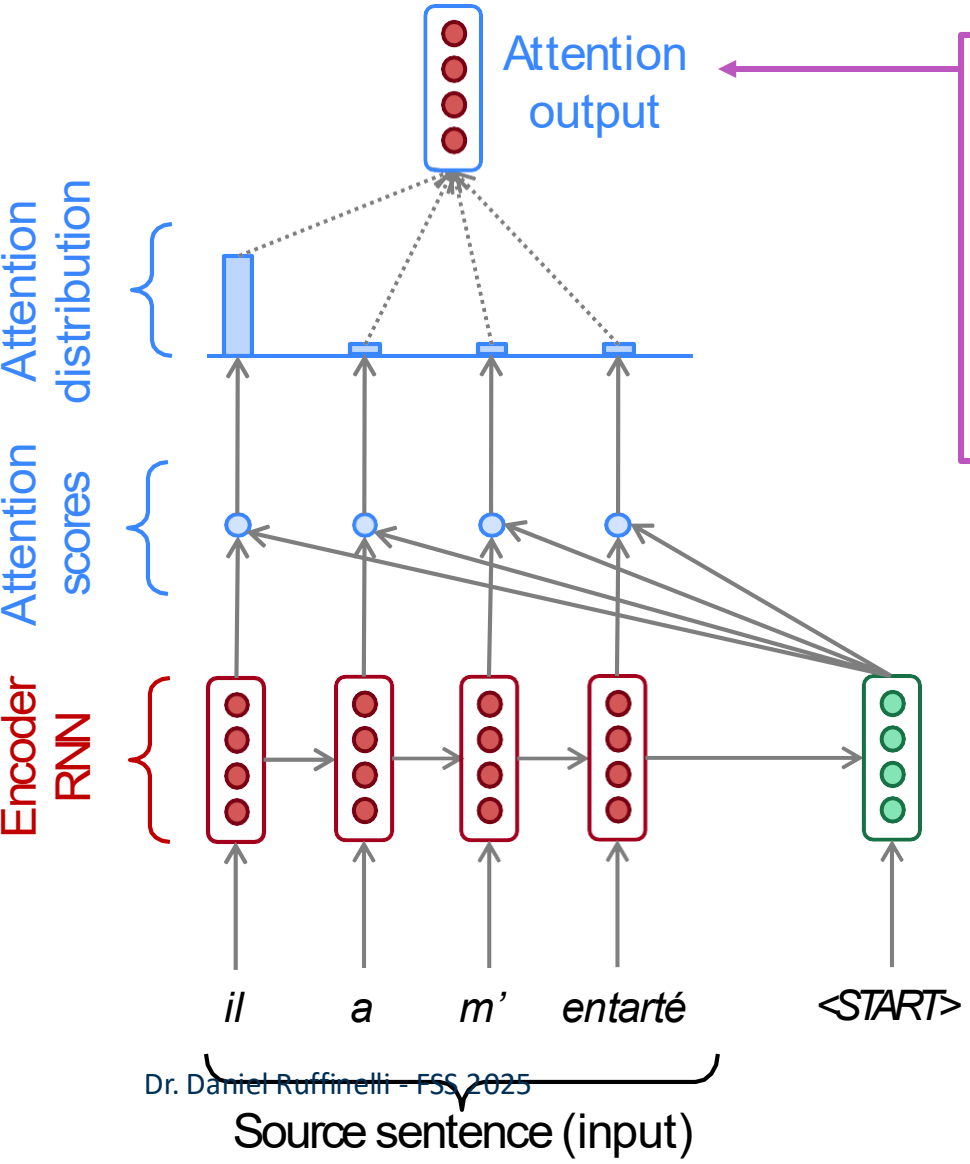


Encoder-decoder with Attention

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics



Encoder-decoder with Attention

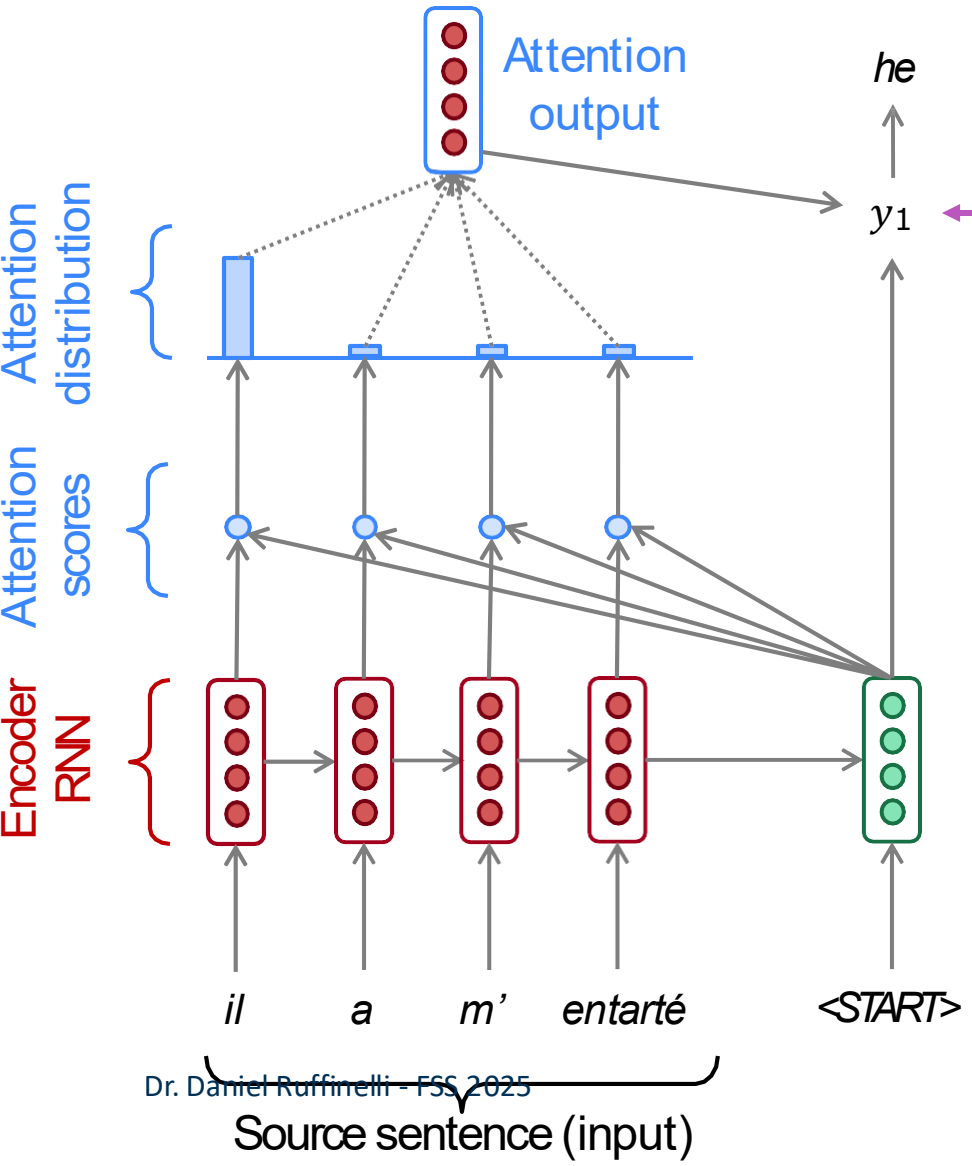


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

Encoder-decoder with Attention

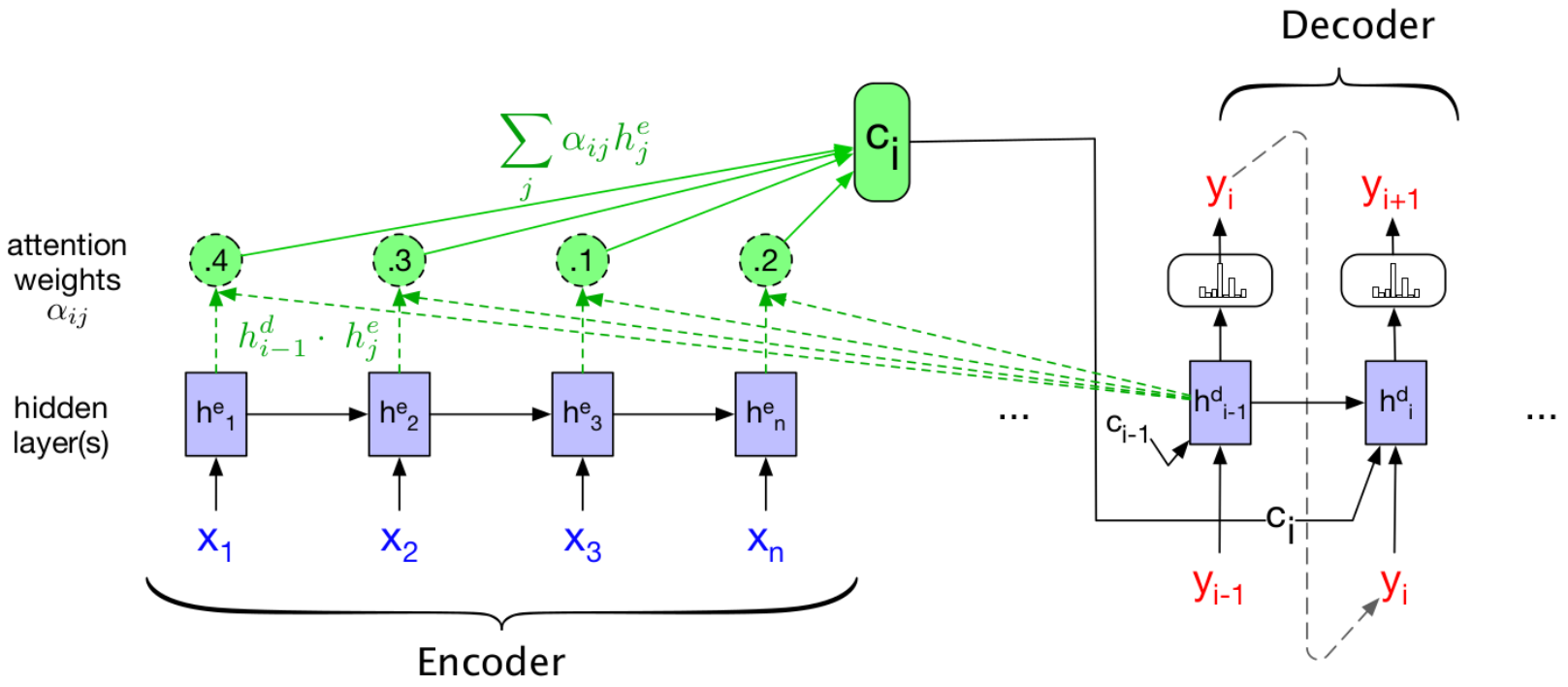


Concatenate attention output with decoder hidden state, then use to compute y_1 as before

Decoder RNN

RNN Encoder-Decoder with Attention

- In a single image:
  - Note that c_i is time dependent
  - That is, decoder can attend to different parts of input to generate different parts of the output text



Summary: Encoder-Decoders

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- **Seq2seq:** tasks where number of output units is variable
  - E.g. translation, summarization, etc.
- Components
  - **Encoder:** encodes input into multiple vectors
  - **Context vector:** encodes input into single vector
  - **Decoder:** produces output states used to generate output sequence
- Limitation
  - Context vector is fixed
  - Limited capacity to encode arbitrarily long sequences
- **Attention mechanism**
  - Designed to bypass need to encode entire input sequence
  - Context vector at each time step based on linear combination of encoder hidden states
  - Fundamental component of transformer architecture (more in next lecture)

References

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Speech and Language Processing, Jurafsky et al., 2024
  - Chapter 9
- Natural Language Processing: A Machine Learning Perspective, Zhang et al, 2018
  - Chapter 14
- Natural Language Processing, Eisenstein, 2018
  - Chapter 6, Section 3
- Some images taken from Stanford's NLP course

  https://web.stanford.edu/class/cs224n/slides/cs224n-2024-lecture05-rnnlm.pdf
- This slide set is based on a previous version of this lecture created by Prof. Dr. Simone Ponzetto