NOVA IMS Information Management School

Text Mining

Word Representations and RNNs

Academic Year 2024-2025

Bruno Jardim bjardim@novaims.unl.pt
Rita Oliveira roliveira@novaims.unl.pt

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Lecture Plan

1. Word Representations
2. Word2Vec
3. Sequential Input
4. Recurrent Neural Networks

NOVA IMS Information Management School

1. Word Representations

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Recall Problems with BoW

1. Curse of dimensionality

2. No semantic relationships
   (Word order is discarded)

3. All words have the same importance

Recall Problems with BoW

1. Curse of dimensionality

2. No semantic relationships
   (Word order is discarded)

3. All words have the same importance

Recall Problems with BoW

1. Curse of dimensionality

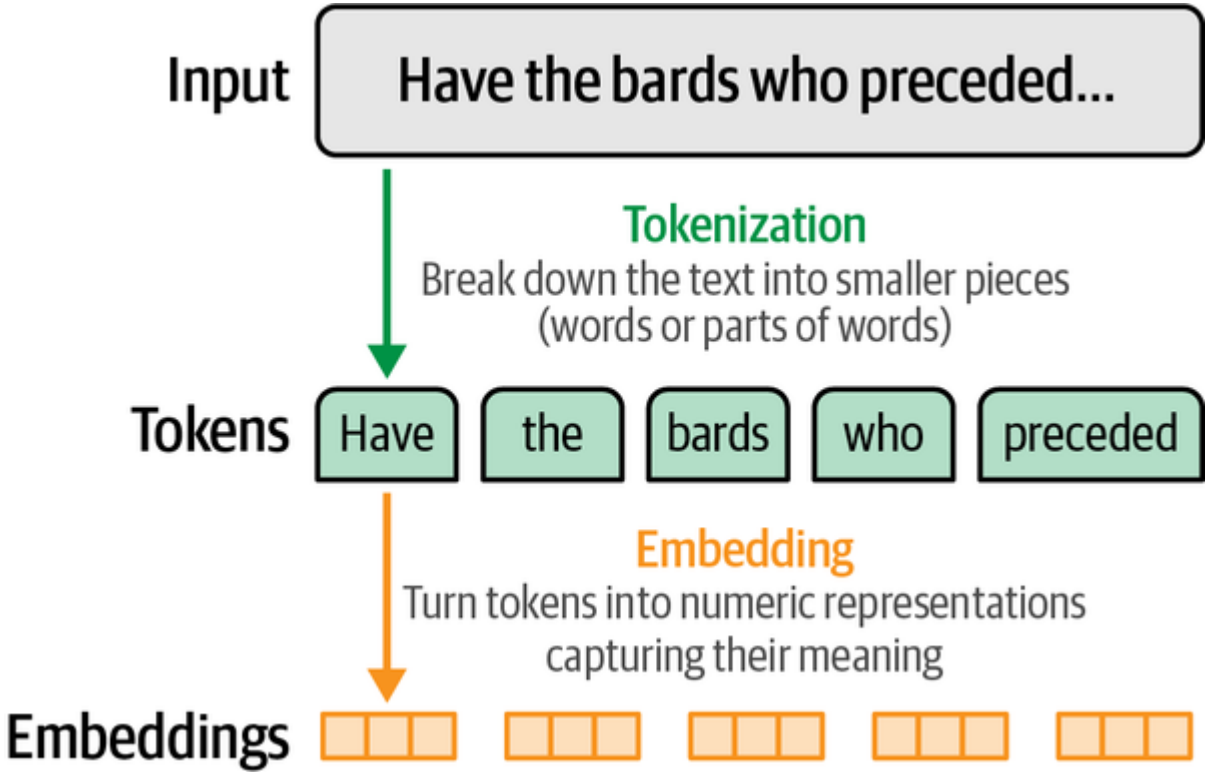
2. No semantic relationships
   (Word order is discarded)

3. All words have the same importance

But how to solve it?

Word Representations

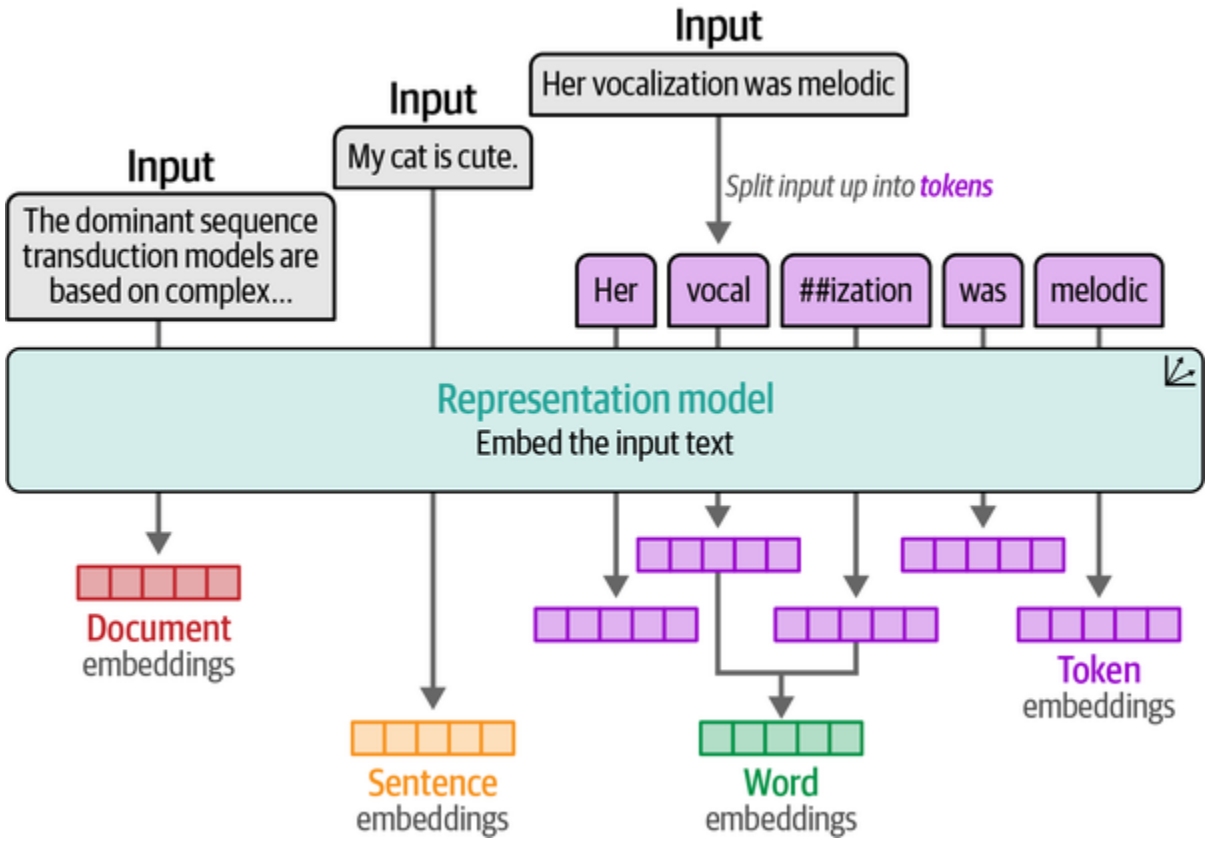
Use word/token representations instead of document ones!



Alammar, J., & Grootendorst, M. (2024). Hands-On large language models: Language Understanding and Generation. Sebastopol : O'Reilly Media.

Word Representations

Use word/token representations instead of document ones!



Alammar, J., & Grootendorst, M. (2024). Hands-On large language models: Language Understanding and Generation. Sebastopol : O'Reilly Media.

Word Representations

Word representations capture semantic relationships

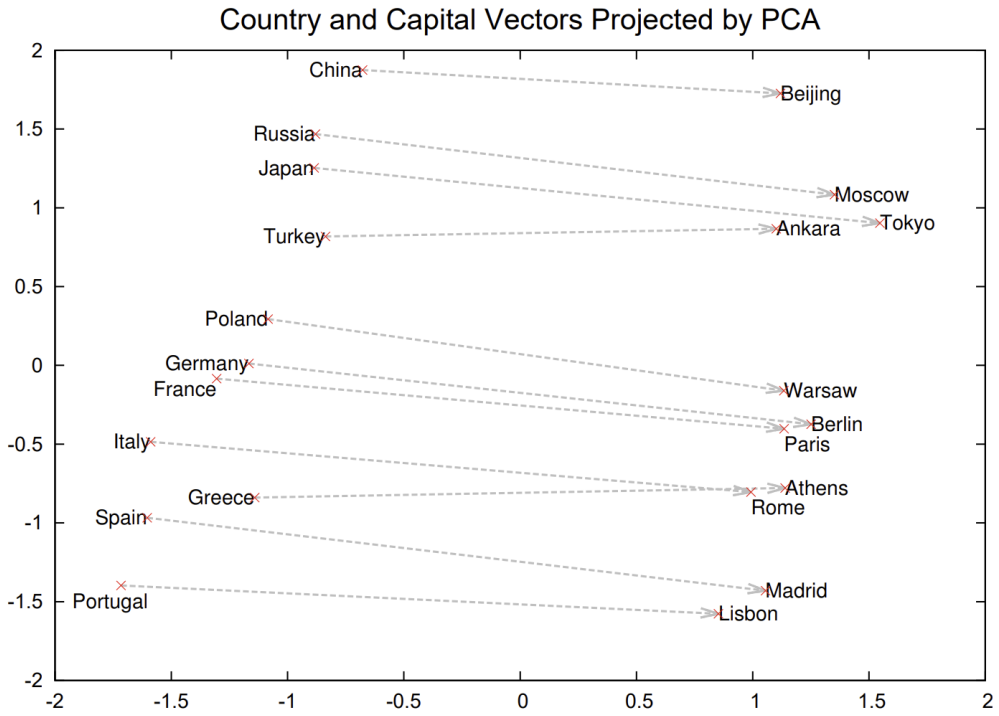
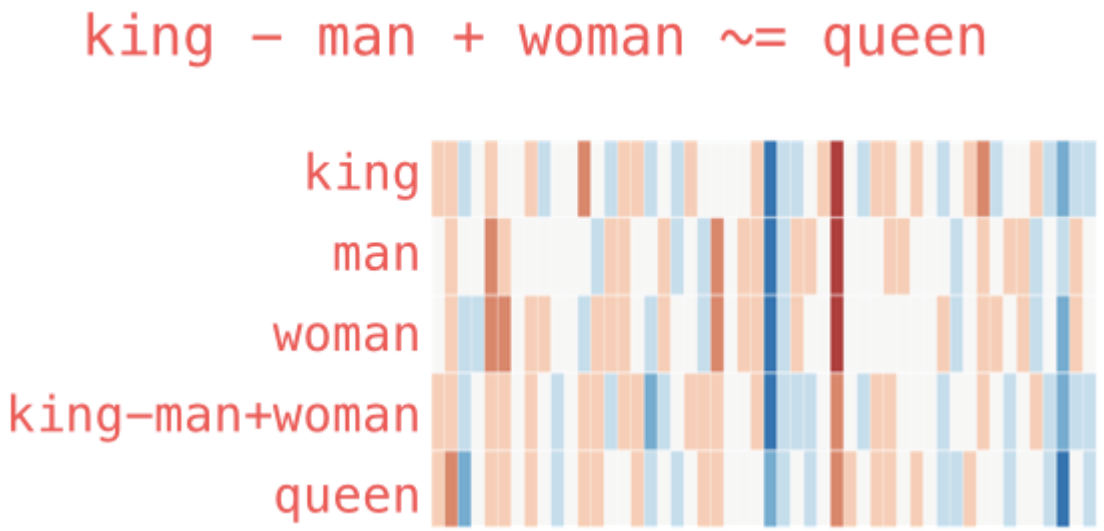


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Word Representations

Word representations capture semantic relationships

The resulting vector from "king-man+woman" almost exactly equals "queen".



Word Representations

Some ways to generate word vectors:

- Term-Term matrix
- Point-wise Mutual Information (PMI)
- Skip-gram (word2vec)

Word Representations

Some ways to generate word vectors:

- **Term-term matrix** is of dimensionality |V|×|V| and each cell records the number of times the row word (target) and the column word (context) co-occur in the training corpus

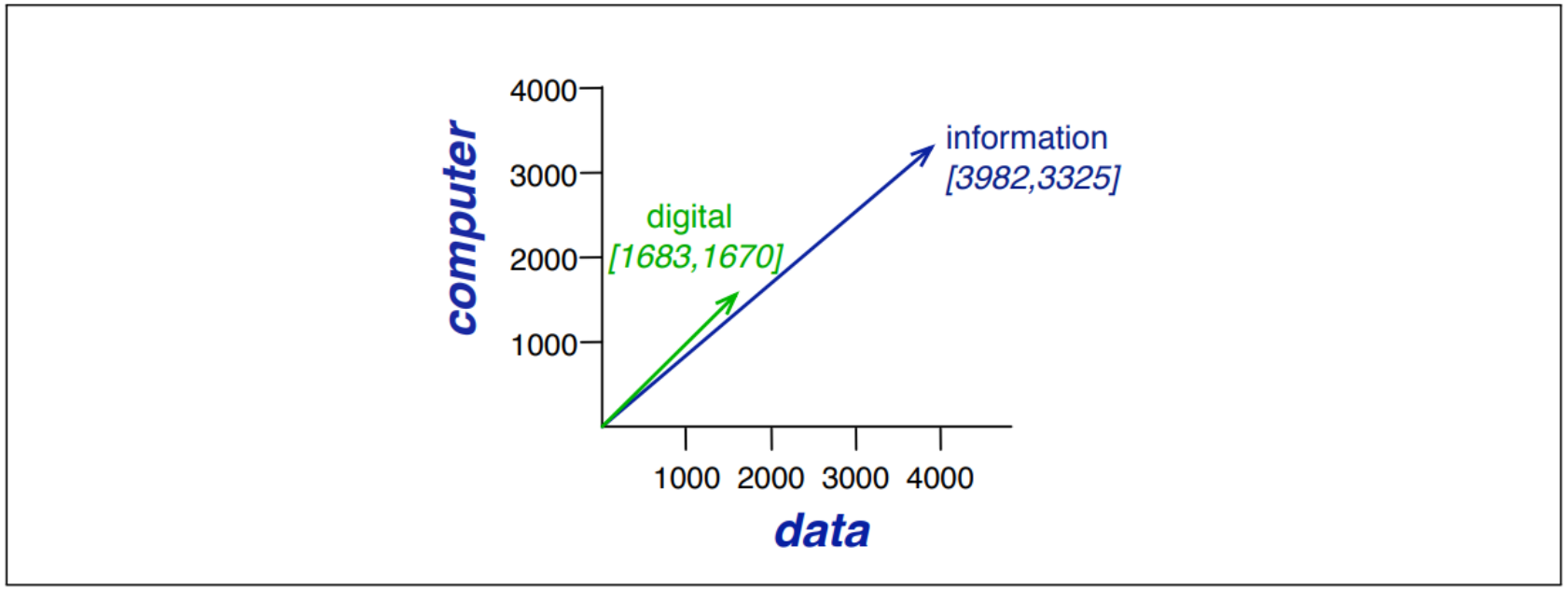
is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Word Representations

Some ways to generate word vectors:

- A spatial visualization of word vectors for digital and information, showing just two of the dimensions, corresponding to the words data and computer.



Word Representations

Some ways to generate word vectors:

- **Point-wise Mutual Information (PMI)** is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

- The pointwise mutual information between a target word w and a context word c is then defined as:

$$\mathrm{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- It is common to use Positive PMI (called PPMI) which replaces all negative PMI values with zero:

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI)

- word = **information,** context = **data**:

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI) between **information** and **data**:

1- Calculate co-occurrence matrix

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI) between **information and data**:

2- Calculate word probability

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI) between **information and data**:

2- Calculate context probability

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

$$P(\text{c=data}) = \frac{5673}{11716} = .4842$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI):

3- Calculate co-occurrence probability

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$P(\text{w=information, c=data}) = \frac{3982}{11716} = .3399$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI):

4- Create Probabilities Matrix

	p(w,context)					p(w)
	computer	**data**	**result**	**pie**	**sugar**	**p(w)**
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$P(\text{w=information, c=data}) = \frac{3982}{11716} = .3399$$

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

$$P(\text{c=data}) = \frac{5673}{11716} = .4842$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI):

5- Calculate PPMI:

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

$$\text{PPMI}(\text{information,data}) = \log_2(.3399/(.6575 * .4842)) = .0944$$

Word Representations

Point-wise Mutual Information (PMI)

Some ways to generate word vectors:

- Calculating Positive Point-wise Mutual Information (PPMI):

5- Calculate PPMI:

$$\text{PPMI}(\text{information},\text{data}) = \log_2(.3399/(.6575 * .4842)) = .0944$$

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

NOVA IMS Information Management School

2. Word2Vec

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

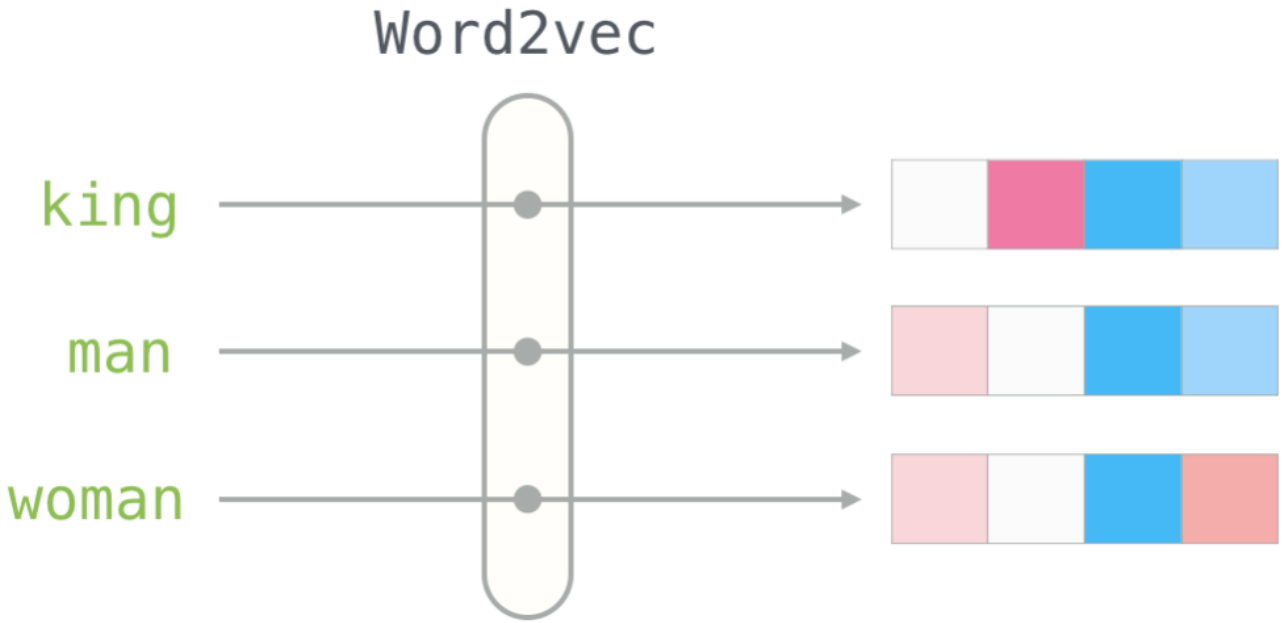
Problems with BoW (cont.)

1. Curse of dimensionality
2. No semantic rela
   (**Word orde** ed)
3. All words have the same importance

Can all be solved through **Word Embeddings**!

Skip-gram

Skip-gram (also known as **word2vec**) is a method to estimate **short dense word vectors**.



Skip-gram

- Skip-gram (also known as **word2vec**) is a method to estimate **short dense word vectors**.
- We **train a classifier** (like simple neural network with a single hidden layer) to perform a binary prediction task: “Is word w likely to show up near c?”
- The intuition is that **similar words tend to appear together**.
- We’re not actually going to use that classifier for the task we trained it on.
- The goal is just to **learn the weights of the hidden layer** – these weights are the word vectors that we’re trying to learn.

Skip-gram

1. Define word and context:

Given a corpus, select all the possible context windows (for example 2 words) of all words:

"(…) Money is a terrible master but an excellent servant (…)"

Skip-gram

1. Define word and context:

Given a corpus, select all the possible context windows (for example 2 words) of all words:

Money is a terrible **master** but an excellent servant

Money is a terrible master **but** an excellent servant

Money is a terrible master but **an** excellent servant

Money is a terrible master but an **excellent** servant

etc.

Word	Context	Target
master	terrible	1
master	but	1
but	master	1
but	an	1
an	but	1
an	excellent	1
excellent	an	1
excellent	servent	1

Etc…

Skip-gram

2. Add negative samples

Add samples of words that are not context in the corpus. After training, the model should be able to predict 0's and 1's.

Word	Context	Target
master	terrible	1
master	**ring**	**0**
master	but	1
but	master	1
but	**late**	**0**
but	an	1
an	**crop**	**0**
an	but	1

Etc…

Skip-gram

3. Define task

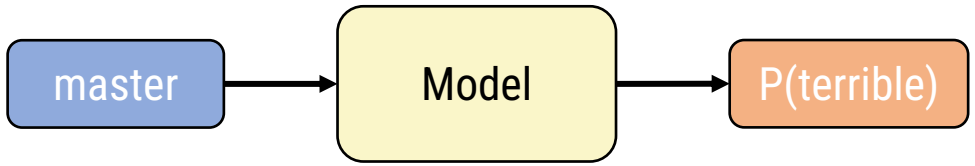
Now the model needs to solve the following task: given a word w and a context word c, predict if this combination of w and c occurred in the corpus (i.e. if target is 1or 0):

Word	Context	Target
master	terrible	?
master	ring	?
master	but	?
but	master	?
but	late	?
but	an	?
an	crop	?
an	but	?

Skip-gram

3. Define task

Now the model needs to solve the following task: give a word w and a context word c, predict if this combination of w and c occurred in the corpus:

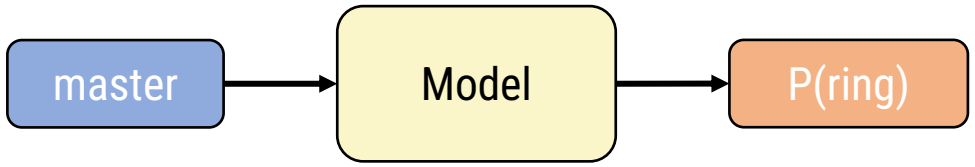


Word	Context	Target
master	terrible	1
master	ring	?
master	but	?
but	master	?
but	late	?
but	an	?
an	crop	?
an	but	?

Skip-gram

3. Define task

Now the model needs to solve the following task: give a word w and a context word c, predict if this combination of w and c occurred in the corpus:



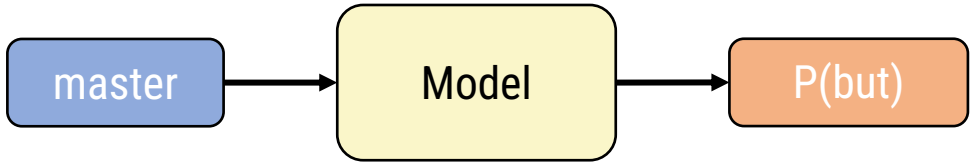
Word	Context	Target
master	terrible	1
master	**ring**	**0**
master	but	?
but	master	?
but	late	?
but	an	?
an	crop	?
an	but	?

Skip-gram

3. Define task

Now the model needs to solve the following task: give a word w and a context word c, predict if this combination of w and c occurred in the corpus:

Word	Context	Target
master	terrible	1
master	ring	0
master	**but**	**1**
but	master	?
but	late	?
but	an	?
an	crop	?
an	but	?



Etc…

Skip-gram

3. Define task

Once the model is trained, its weights are used to represent the words

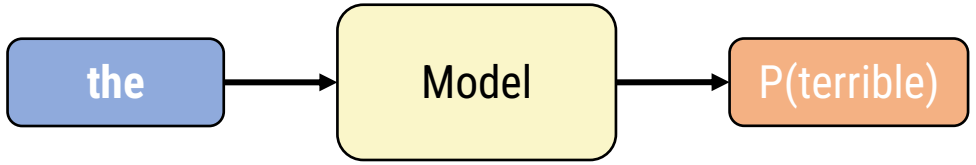
Weight	Value
w1	0.234
w2	1.298
w3	2.762
w4	0.567
w5	1.222
w6	0.334
(...)	

Skip-gram

4. Train the model

Example training input **P(the|terrible)**

Define input:



Input size
1 x v, where v is the size of the vocabulary

(…)	the	master	money	(…)
(…)	0	0	0	(…)
(…)	1	0	0	(…)
(…)	0	1	0	(…)
(…)	0	0	1	(…)
(…)	0	0	0	(…)
(…)	0	0	0	(…)
(…)	(…)	(…)	(…)	(…)

Size of vocabulary

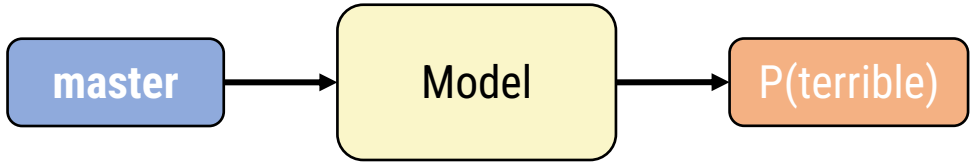
Input vector of “the” (with size of vocabulary)
[(…), 0, 1, 0, 0, 0, (…)]

Skip-gram

4. Train the model

Example training input **P(master|terrible)**

Define input:



Input size
1 x v, where v is the size of the vocabulary

(…)	the	master	money	(…)
(…)	0	0	0	(…)
(…)	1	0	0	(…)
(…)	0	1	0	(…)
(…)	0	0	1	(…)
(…)	0	0	0	(…)
(…)	0	0	0	(…)
(…)	(…)	(…)	(…)	(…)

Size of vocabulary

Input vector of “master” (with size of vocabulary)
[(…), 0, 1, 0, 0, 0, (…)]

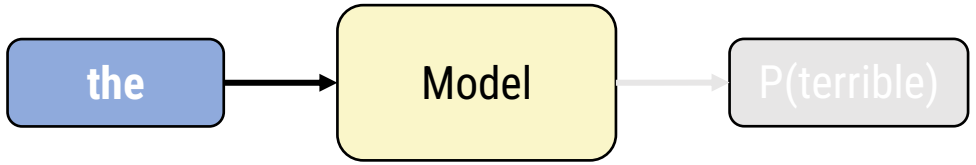
Skip-gram

4. Train the model

Define the size of the weight matrix (W), this will be our embedding size, i.e. size of word vector retrieved)

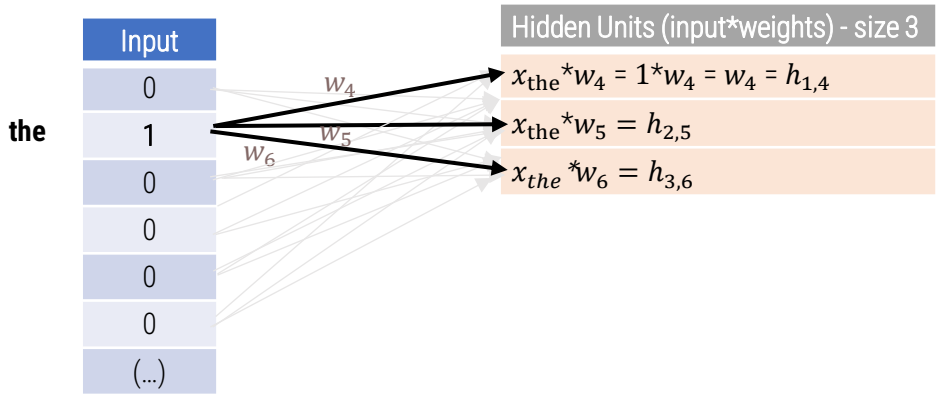
In the example below **W is size 3**

(weights can be initialized randomly)



Input
1 x v

Weights Matrix shape
v x n, where **n** is the size of the embeddings (here it is 3)



Hidden vector of size 3 (embedding size)

$[h_{1,4}, h_{2,5}, h_{3,6}]$

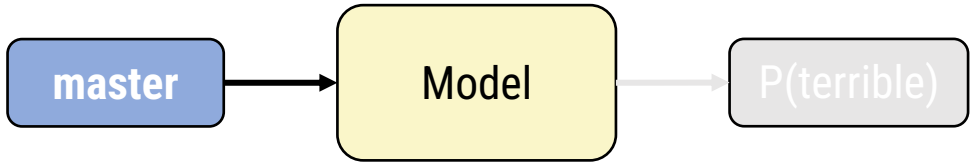
Skip-gram

4. Train the model

Define the size of the weight matrix (W), this will be our embedding size, i.e. size of word vector retrieved)

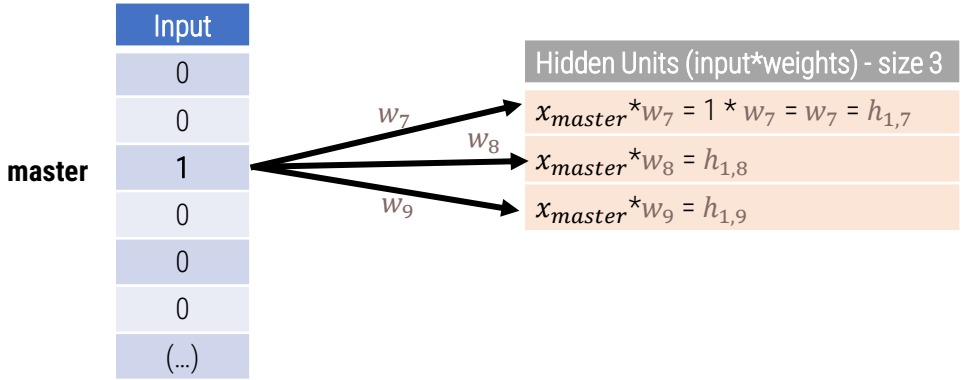
In the example below **W is size 3**

(weights can be initialized randomly)



Input
1 x v

Weights Matrix shape
v x n, where **n** is the size of the embeddings (here it is 3)



Hidden vector of size 3 (embedding size)

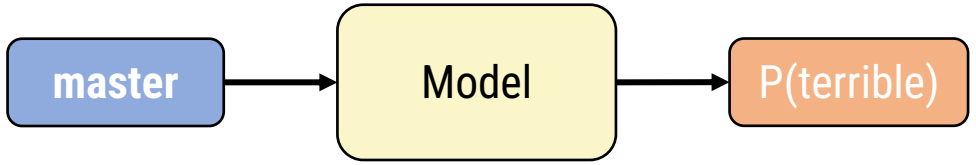
$[h_{1,7}, h_{1,8}, h_{1,9}]$

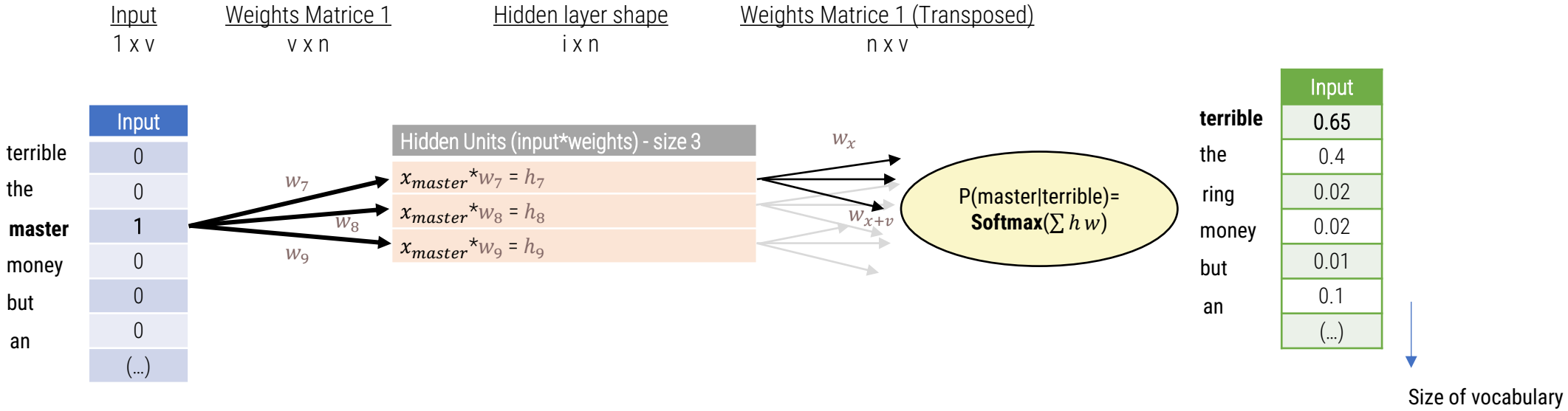
Skip-gram

4. Train the model

Multiple by a second matrix of weights, sum the results and transform it into probabilities for each possible word.

Output will be a value between 0 and 1.



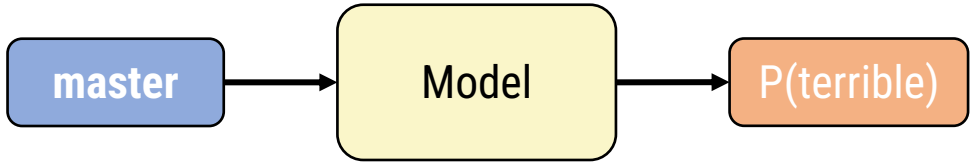


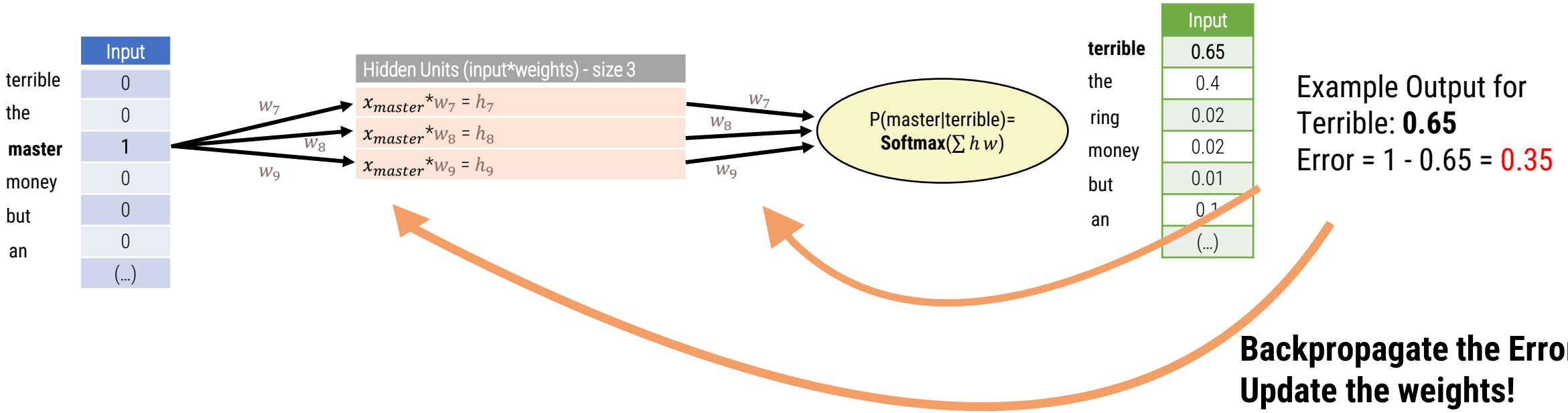
Skip-gram

4. Train the model

Multiple by a second matrix of weights, sum the results and transform it into probabilities for each possible word.

Output will be a value between 0 and 1.



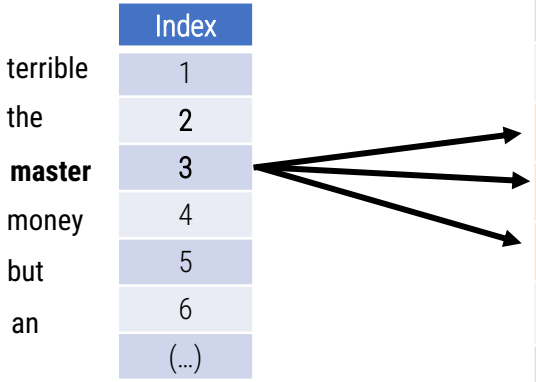


Skip-gram

4. Retrieve Embeddings

After training the model, the embeddings are stored in the weight matrix W=[w_1, w_2, w_3, (…), w_4]

In the example below, **master** can be represented as the embedding **[0.445, 0.897, 0.996]**



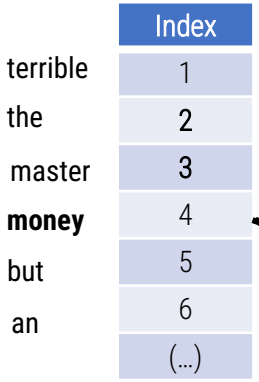
	Index
terrible	1
the	2
master	3
money	4
but	5
an	6
	(…)

Embeddings	Example Values
(…)	(…)
$w_{3\ \text{the}}$	0.454
$w_{4\ \text{the}}$	0.233
$w_{5\ the}$	0.345
$\mathbf{w_{7\ master}}$	0.445
$\mathbf{w_{8\ master}}$	0.897
$\mathbf{w_{9\ master}}$	0.966
$w_{10\ \text{money}}$	0.327
$w_{11\ \text{money}}$	0.988
$w_{12\ money}$	0.276
(…)	

Skip-gram

4. Retrieve Embeddings

- After training the model, the embeddings are stored in the weight matrix W=[w_1, w_2, w_3, (…), w_4]
- In the example below, **money** can be represented as the embedding **[0.327, 0.988, 0.276]**

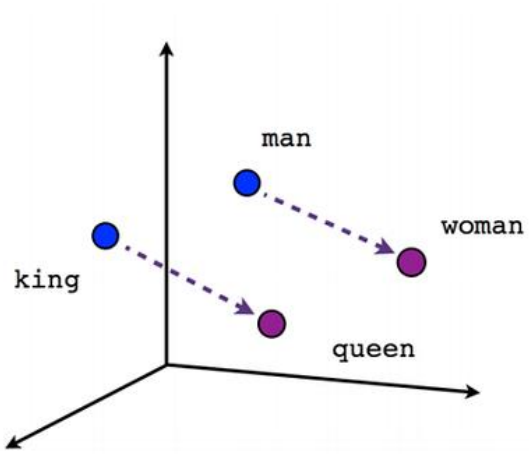


Embeddings	Example Values
(…)	(…)
w_{3} the	0.454
w_{4} the	0.233
w_{5} *the*	0.345
w_{7} master	0.445
w_{8} master	0.897
w_{9} master	0.966
w_{10} money	0.327
w_{11} money	0.988
w_{12} money	0.276
(…)	

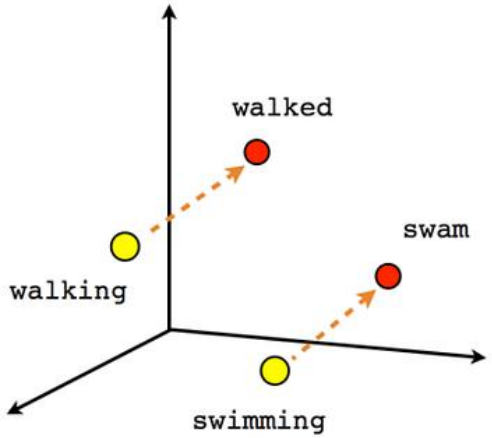
Word Embeddings

Word Embeddings Properties

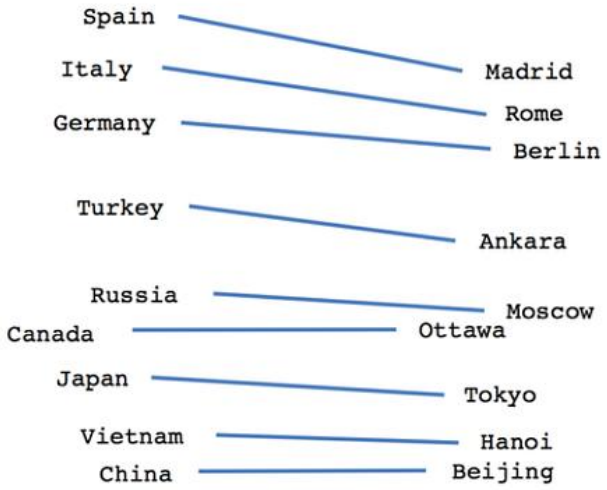
- After trained, Word Embeddings can extract **semantic relationships** between words



Male-Female



Verb tense



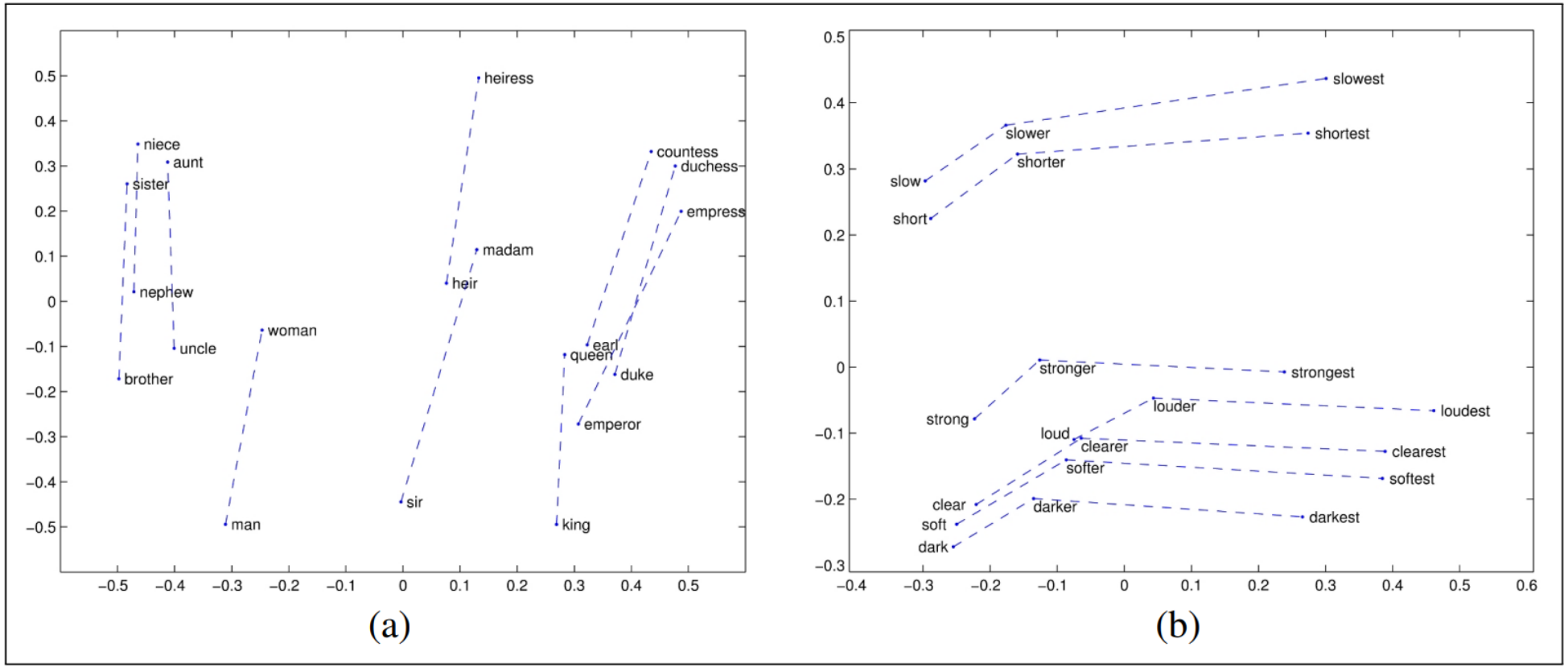
Country-Capital

https://medium.com/data-science/creating-word-embeddings-coding-the-word2vec-algorithm-in-python-using-deep-learning-b337d0ba17a8

Word Embeddings

Word Embeddings Properties

- After trained, Word Embeddings can extract **semantic relationships** between words

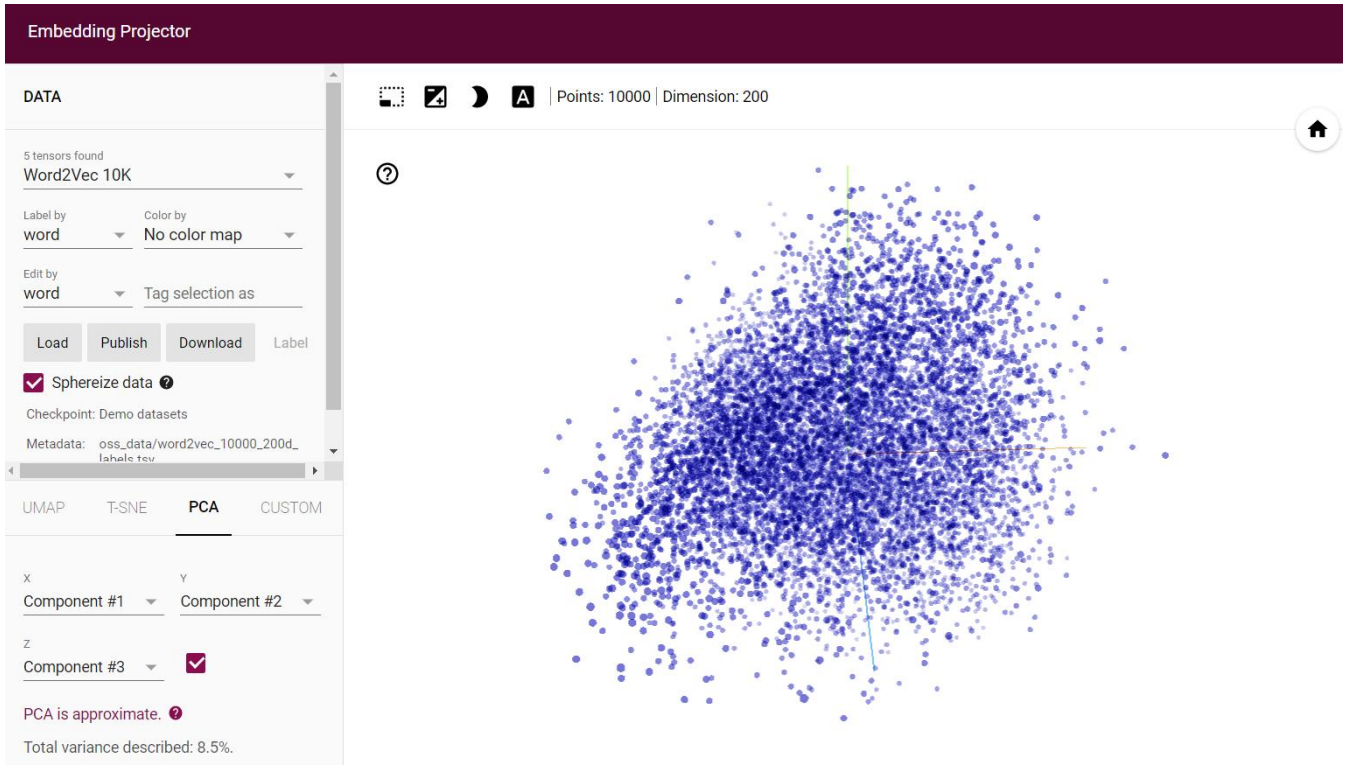


Word Embeddings

Word Embeddings Properties

- Word Embeddings can extract **semantic relationships** between words

https://projector.tensorflow.org/



Word Embeddings

Read the following for a deeper understanding of how word2vec and word embeddings work:

- Web format: https://jalammar.github.io/illustrated-word2vec/
- Video format: https://www.youtube.com/watch?v=ISPId9Lhc1g

NOVA IMS Information Management School

3. Sequential Input

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Word Embeddings

Word embeddings are word vector representations, making each sentence/document a sequence of vectors:

Example: “No pain no gain”

Word Embeddings

Word embeddings are word vector representations, making each sentence/document a sequence of vectors:

Example: “No pain no gain”

Word	Example Embedding
no	[0.765, 0.523, 0.895]
pain	[0.145, 0.534, 0.556]
gain	[0.034, 0.235, 0.987]

Word Embeddings

Word embeddings are word vector representations, making each sentence/document a sequence of vectors:

Example: “**No** pain **no** gain”

Word	Example Embedding
no	[0.765, 0.523, 0.895]
pain	[0.145, 0.534, 0.556]
gain	[0.034, 0.235, 0.987]

Sentence vector (considering embeddings of size 3):

[[0.765, 0.523, 0.895], [0.145, 0.534, 0.556], [0.765, 0.523, 0.895], [0.034, 0.235, 0.987]]

Word Embeddings

Word embeddings are word vector representations, making each sentence/document a sequence of vectors:

Example: “No **pain** no gain”

Word	Example Embedding
no	[0.765, 0.523, 0.895]
pain	[0.145, 0.534, 0.556]
gain	[0.034, 0.235, 0.987]

Sentence vector (considering embeddings of size 3):

[[0.765, 0.523, 0.895], [0.145, 0.534, 0.556], [0.765, 0.523, 0.895], [0.034, 0.235, 0.987]]

Word Embeddings

Word embeddings are word vector representations, making each sentence/document a sequence of vectors:

Example: “No pain no **gain**”

Word	Example Embedding
no	[0.765, 0.523, 0.895]
pain	[0.145, 0.534, 0.556]
gain	[0.034, 0.235, 0.987]

Sentence vector (considering embeddings of size 3):

[[0.765, 0.523, 0.895], [0.145, 0.534, 0.556], [0.765, 0.523, 0.895], [0.034, 0.235, 0.987]]

Word Embeddings

Recall how to input **BoW** vectors to model:

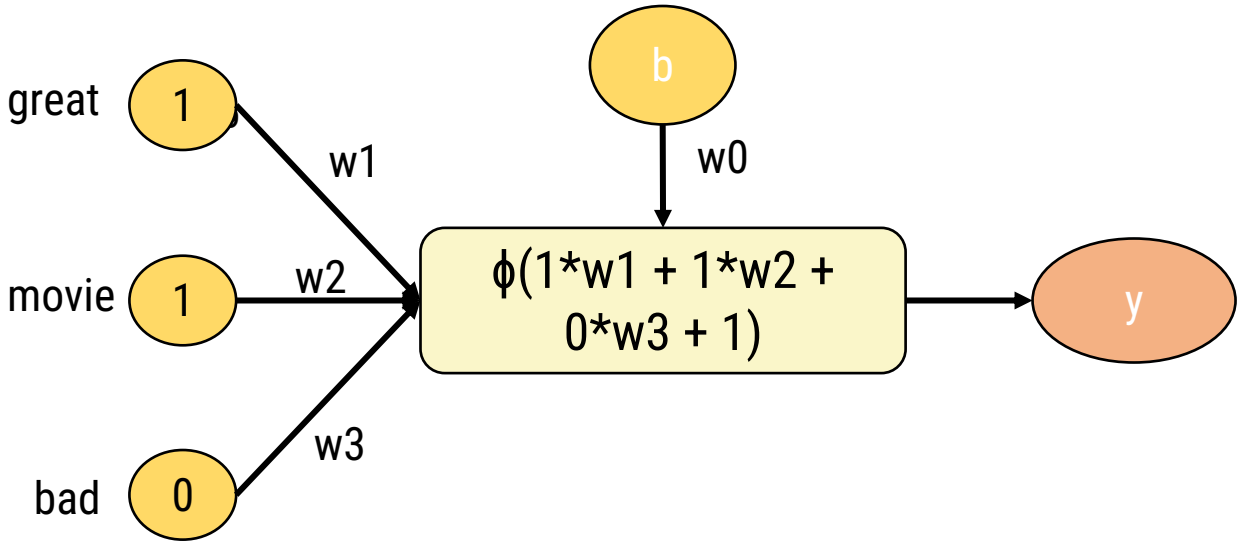
Review 1 – “Great movie”
Review 2 – “Bad movie”

	great	movie	bad
R1	1	1	0
R2	0	1	1

Task:

Predict the sentiment of movie reviews using a neural network.

Review 1:



Word Embeddings

What if we consider word embeddings:

Review 1 – “Great movie”
Review 2 – “Bad movie”

	great	movie	bad
R1	[0.034, 0.235, 0.987]	[0.765, 0.523, 0.895]	-
R2	-	[0.765, 0.523, 0.895]	[0.145, 0.534, 0.556]

Note: We assume embeddings size 3 for simplicity

Task:

Predict the sentiment of movie reviews using a neural network.

Word Embeddings

What if we consider word embeddings:

Review 1 – “Great movie”
Review 2 – “Bad movie”

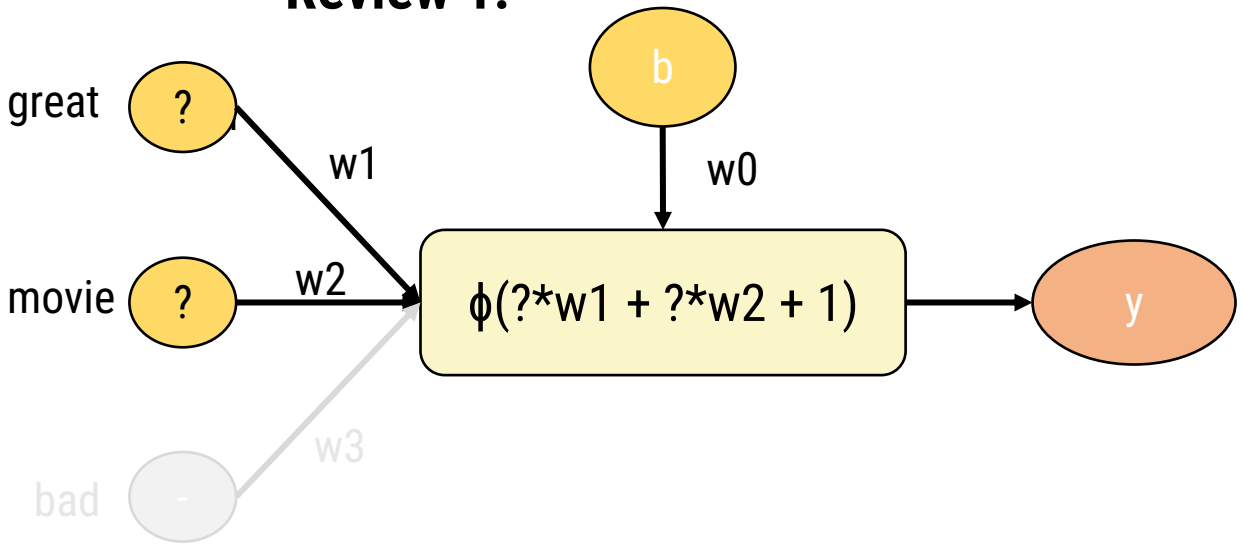
	great	movie	bad
R1	[0.034, 0.235, 0.987]	[0.765, 0.523, 0.895]	-
R2	-	[0.765, 0.523, 0.895]	[0.145, 0.534, 0.556]

Note: We assume embeddings size 3 for simplicity

Task:

Predict the sentiment of movie reviews using a neural network.

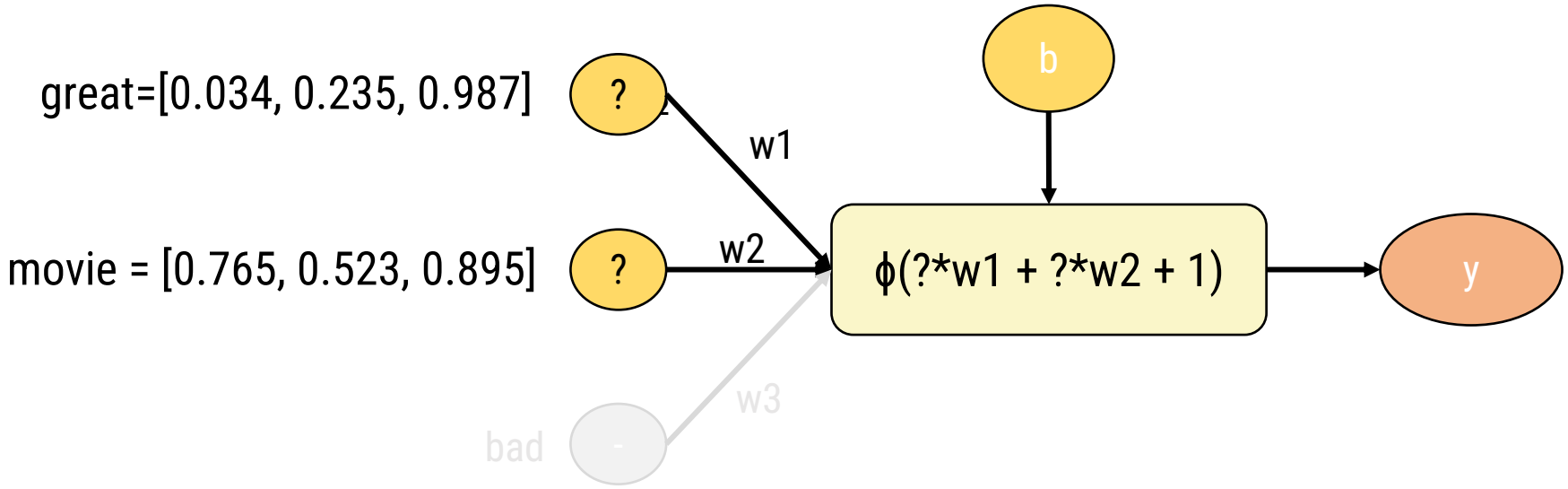
Review 1:



Word Embeddings

How does a sentence enter the network if it's a set of vectors and not a single vector?

Review 1 – "Great movie"



Word Embeddings

One way could be to **average the vectors** of the sentence:

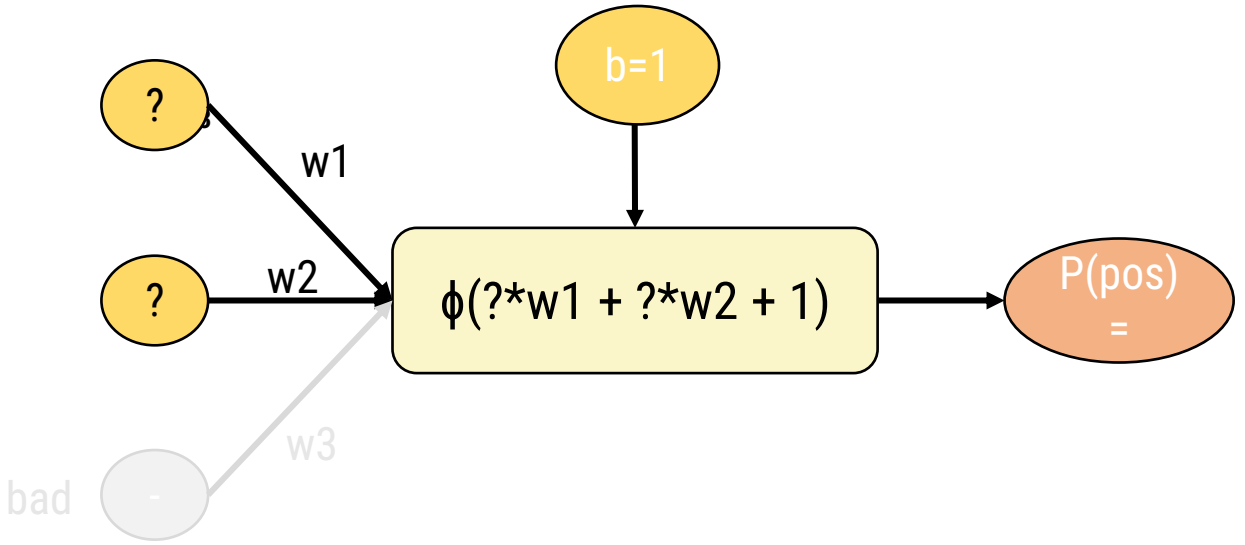
Review 1 – “Great movie”

(great+movie)/2 =

= ([0.034, 0.235, 0.987]+ [0.765, 0.523, 0.895])/2=

=([0.799, 0.758, 1.882])/2=

=[0.399, 0.379, 0.941]

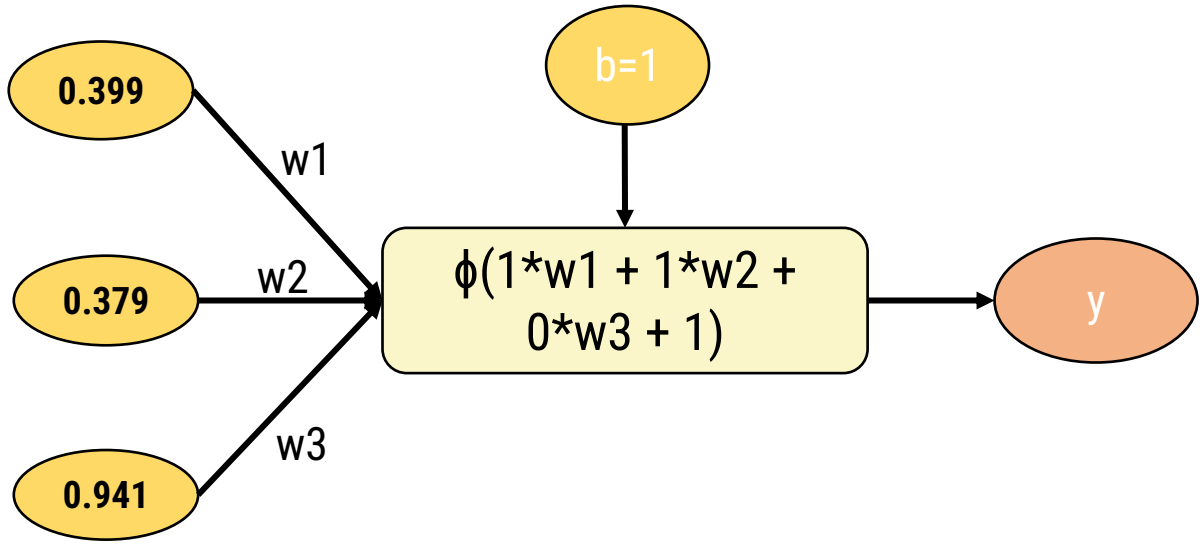


Word Embeddings

Now we have again an input able to enter the model:

Review 1 – “Great movie”

R1	0.399	0.379	0.941



Word Embeddings

BoW vs Word Embeddings

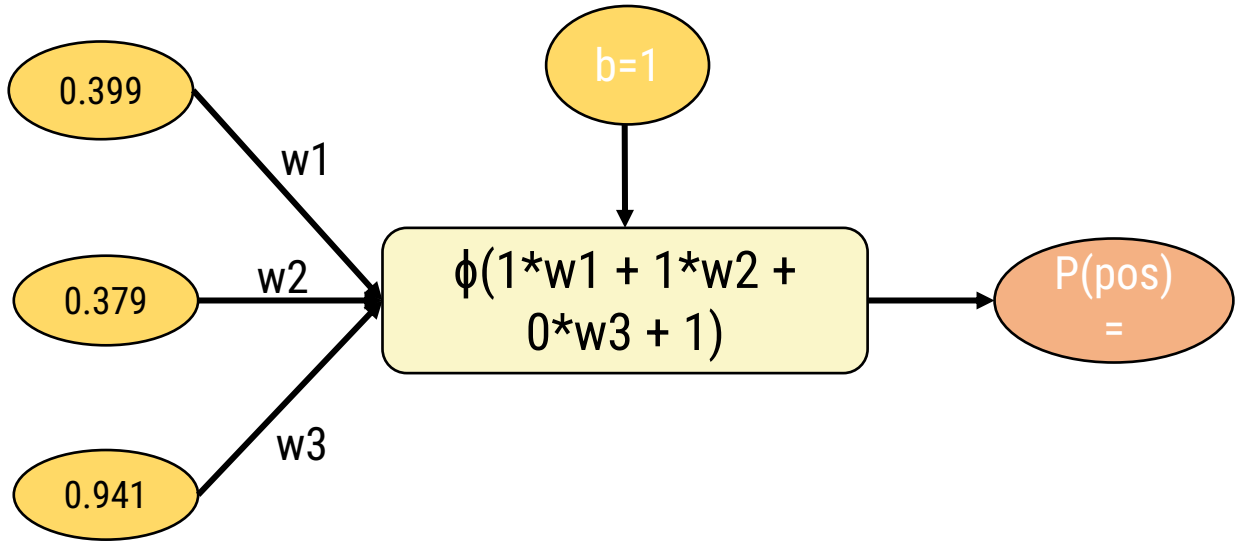
Review 1 – “Great movie”

BoW

	great	movie	bad	…	Word n
R1	1	1	0	…	0

Word Embeddings:

R1	0.399	0.379	0.941



Word Embeddings

BoW vs Word Embeddings

Review 1 – “Great movie”

BoW (with n=size of vocabulary)

	great	movie	bad	…	word n
R1	1	1	0	0	0

Word Embeddings (with n=3):

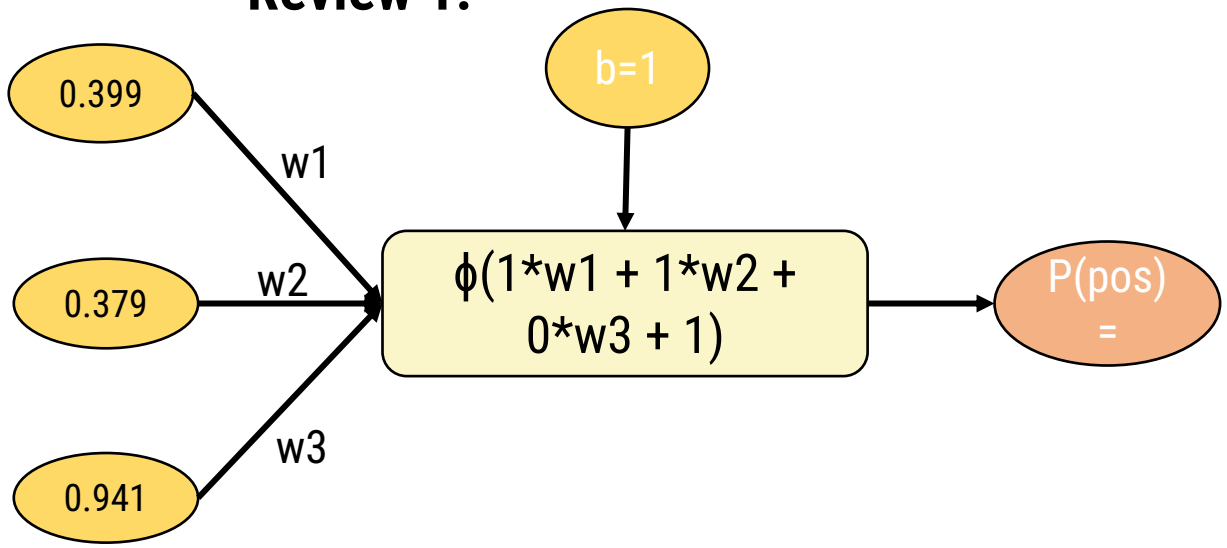
R1	0.399	0.379	0.941

Sparse vs Dense vectors!

Task:

Predict the sentiment of movie reviews using a neural network.

Review 1:



Word Embeddings

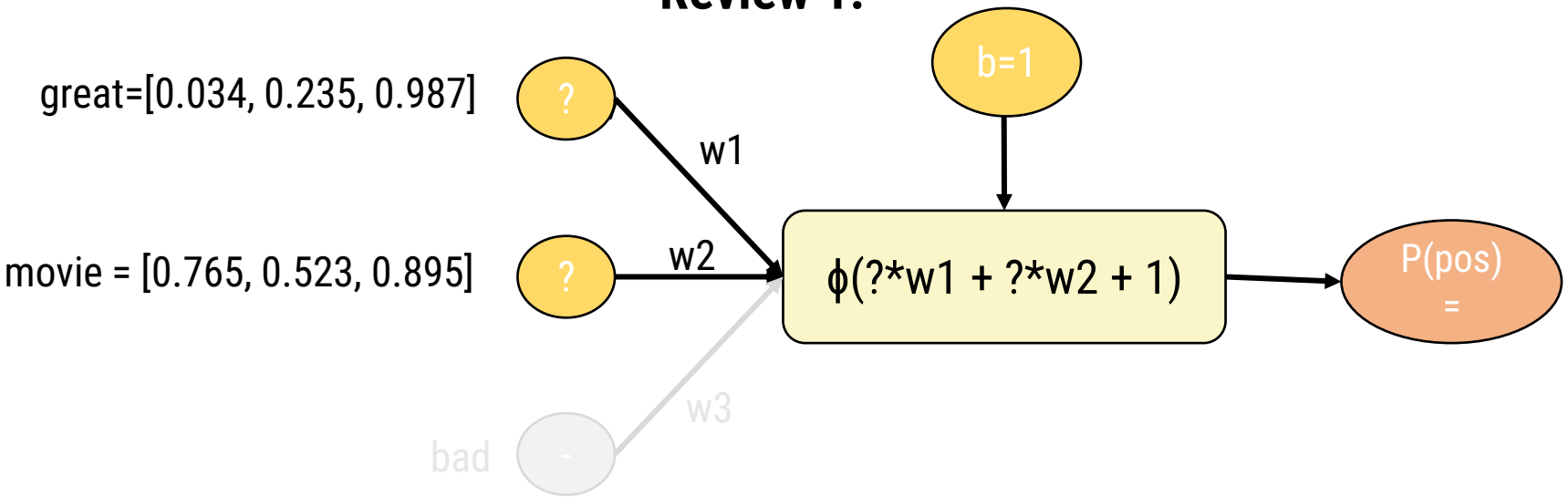
What if we do not want to average/sum/concatenate the vectors?

Review 1 – “Great movie”
Review 2 – “Bad movie”

Task:

Predict the sentiment of movie reviews using a neural network.

Review 1:

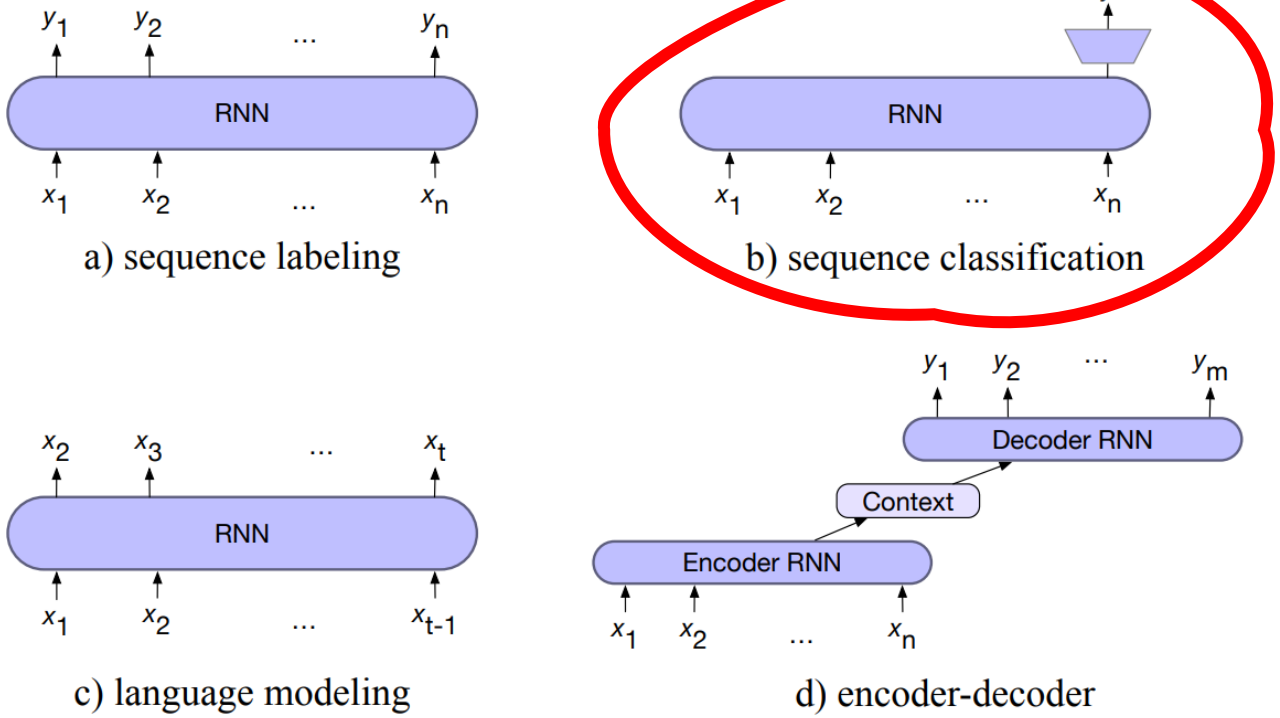


Word Embeddings

To process sequential input, we can use some well know sequential models, like **Recurrent Neural Networks (RNN)** and **Long Short Term Memory Neural Networks (LSTM)**

Common Tasks

- Next word prediction
- Sequence Labeling
- **Sequence Classification**
- Machine Translation



NOVA IMS Information Management School

4. Recurrent Neural Networks

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA IMS Information Management School

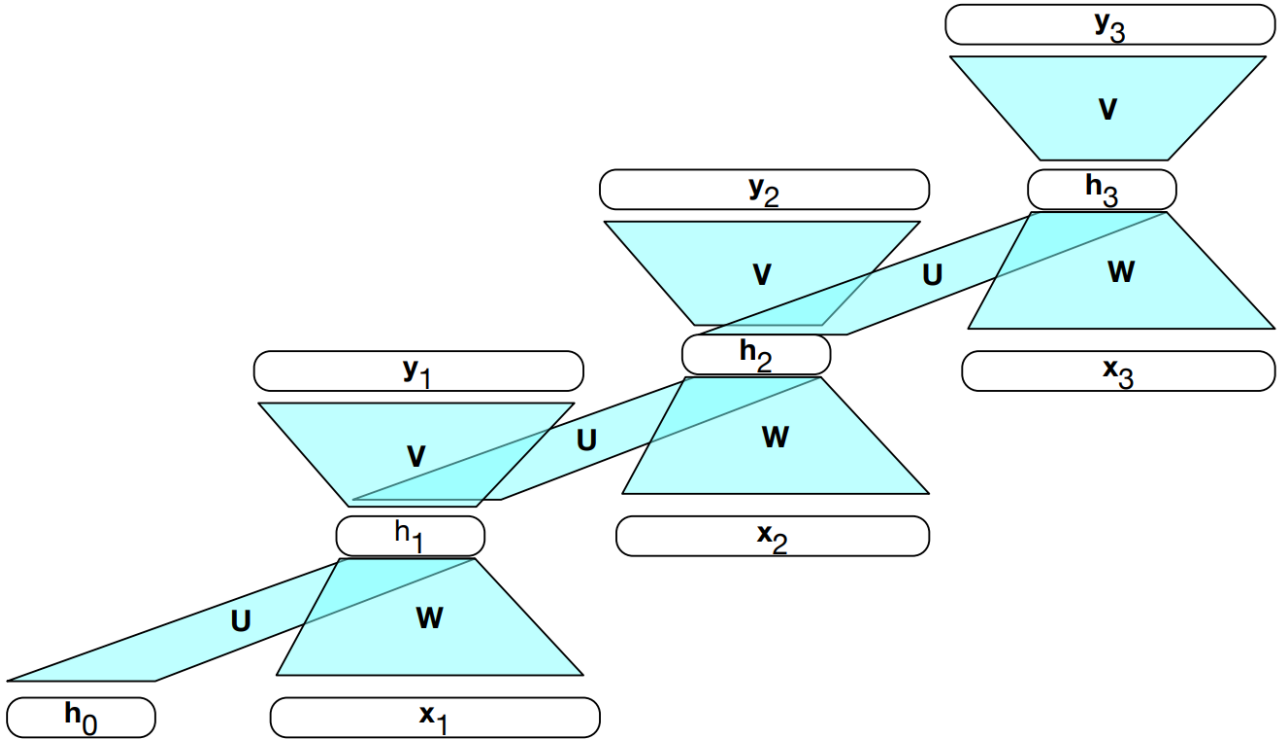
Recurrent Neural Networks

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

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

Recurrent Neural Networks

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



Recurrent Neural Networks

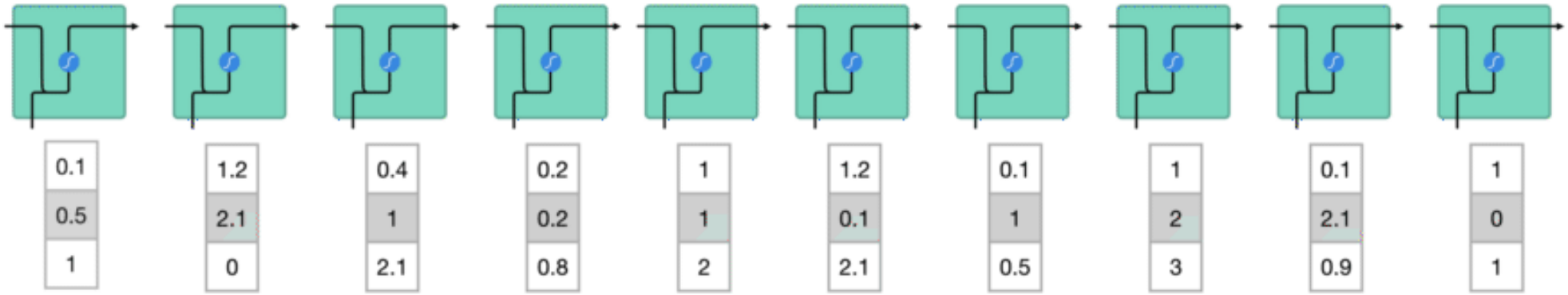
The matrices U, V and W are shared across time, while new values for h and y are calculated with each time step

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Recurrent Neural Networks

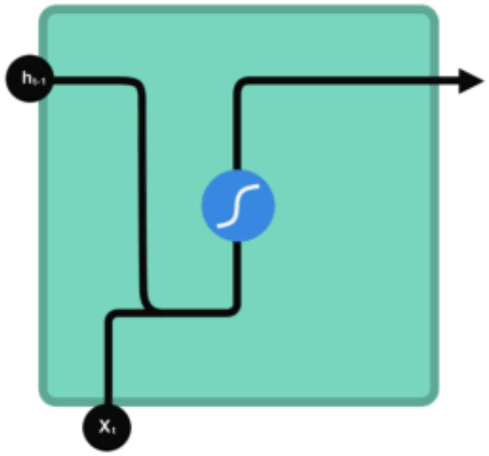
For each sentence:

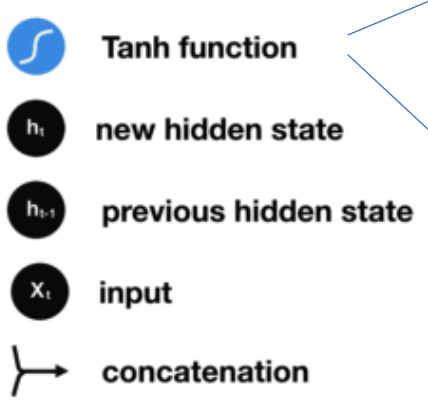


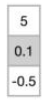
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Recurrent Neural Networks

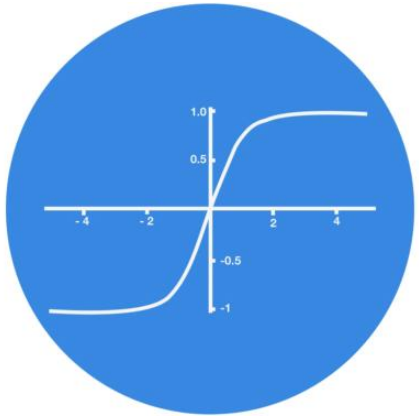
For each word:







Tanh



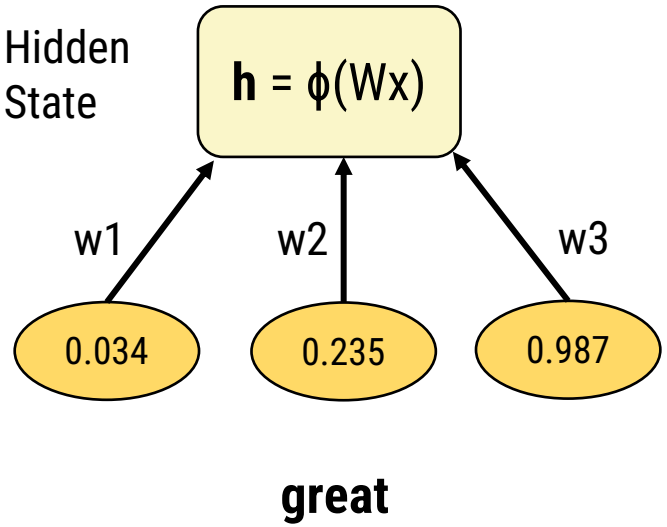
A tanh function ensures that the values stay between -1 and 1, thus regulating the output of the neural network, i.e., allowing the values to stay between the boundaries -1 and 1. Otherwise, some values can explode and become astronomical, causing other values to seem insignificant.

Recurrent Neural Networks

Example:

Review 1 – “**Great** movie”

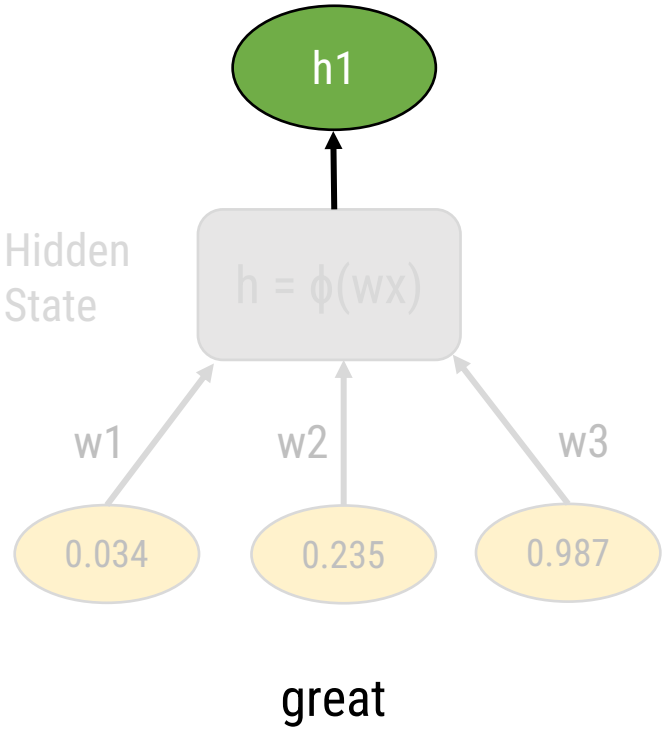
	Word Embeddings (size 3)	
	great	movie
R1	[0.034, 0.235, 0.987]	[0.765, 0.523, 0.895]



Recurrent Neural Networks

Example:

Review 1 – “**Great** movie”



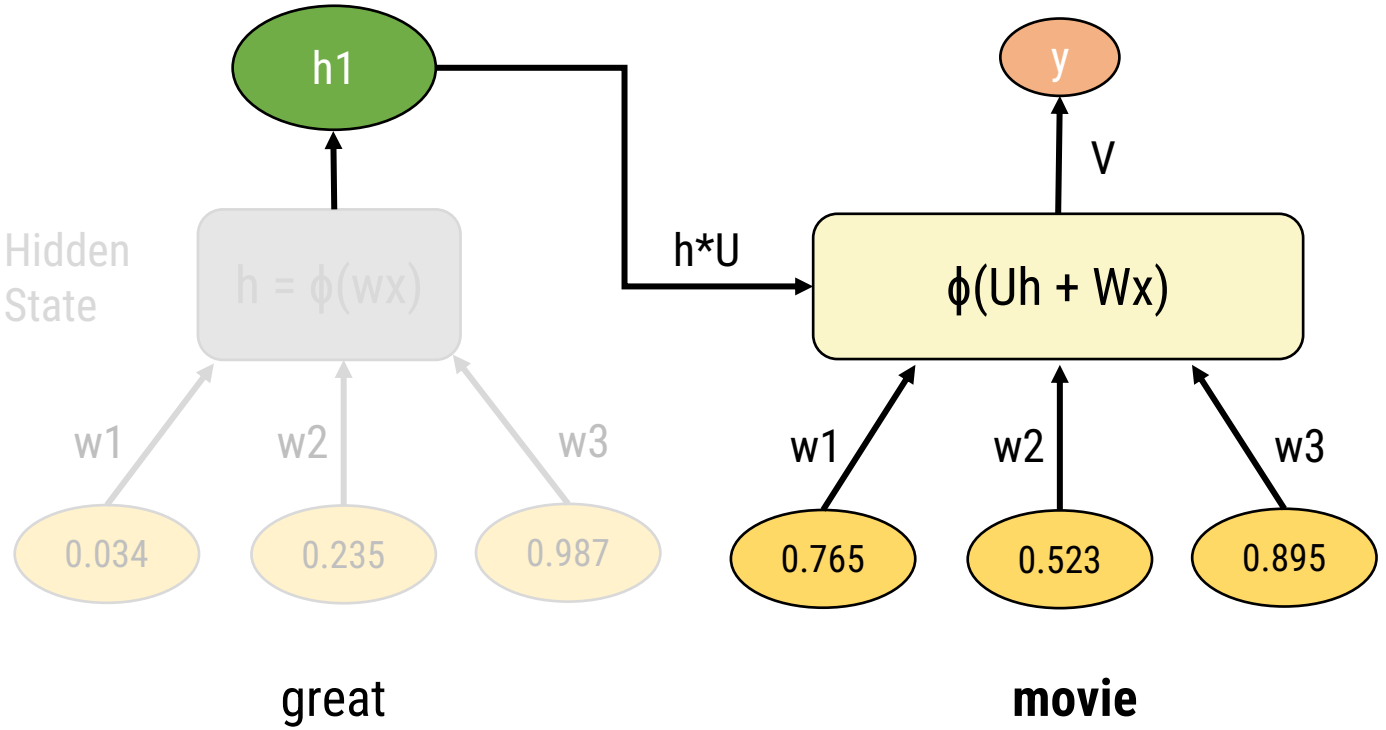
	Word Embeddings (size 3)	
	great	movie
R1	[0.034, 0.235, 0.987]	[0.765, 0.523, 0.895]

Recurrent Neural Networks

Example:

Review 1 – “Great **movie**”

	Word Embeddings (size 3)	
	great	**movie**
R1	[0.034, 0.235, 0.987]	[0.765, 0.523, 0.895]

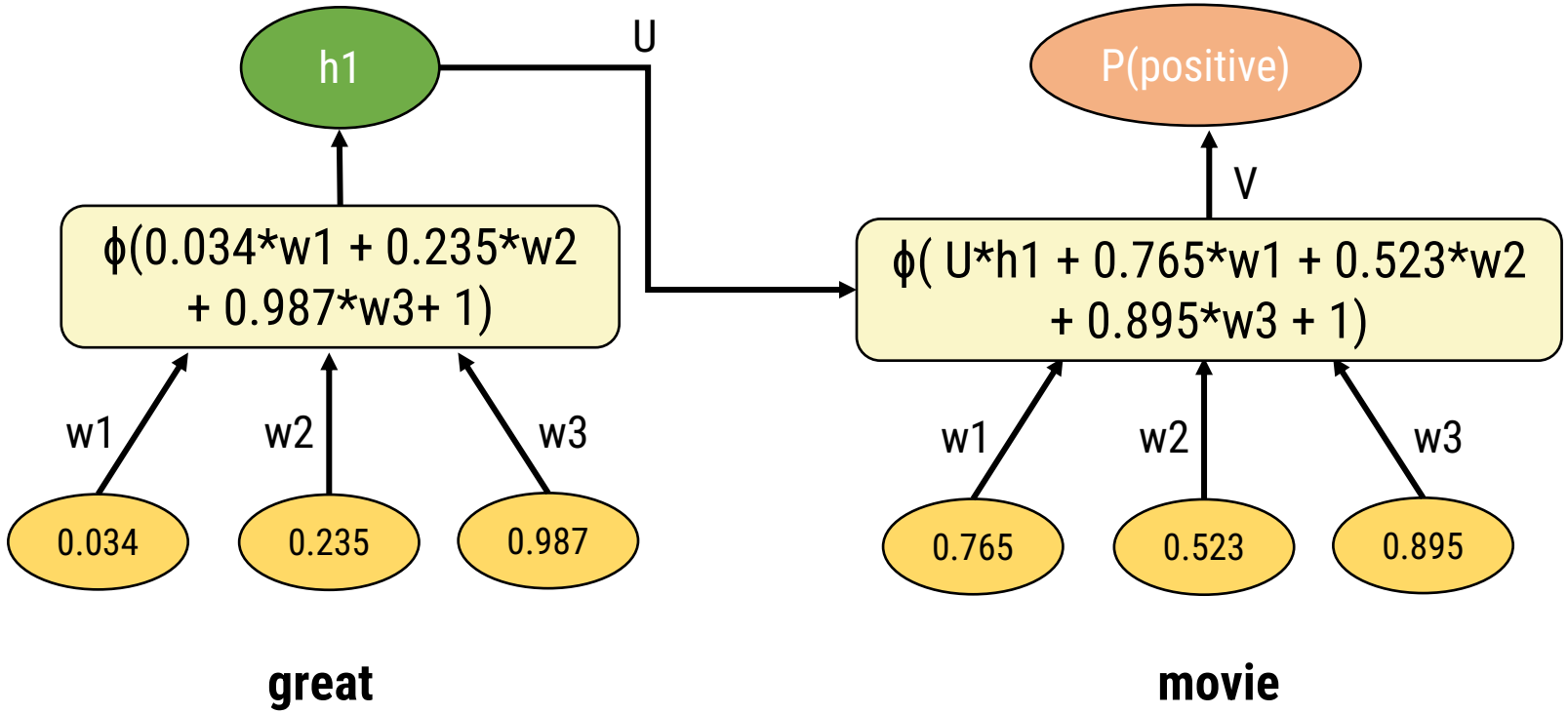


Recurrent Neural Networks

Example:

Review 1 – “Great movie”

Task:
Predict the sentiment of review using a RNN:



Recurrent Neural Networks

Problems with RNNs

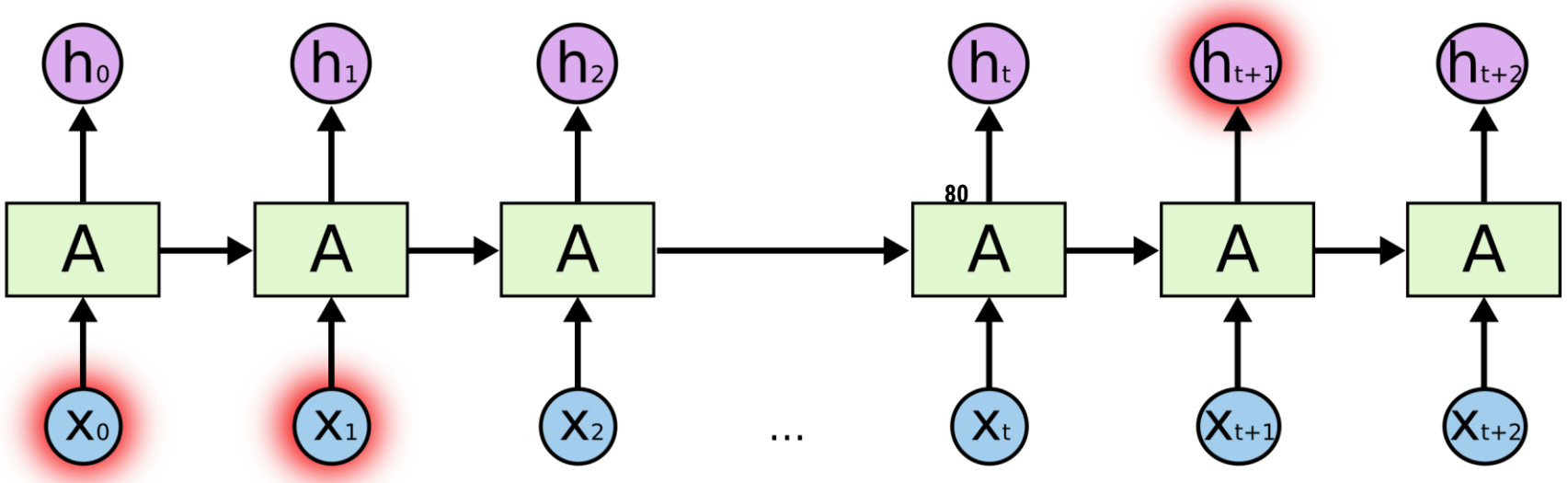
- **Long Dependencies:**

The final hidden state reflects more information about the end of the sentence than its beginning.

Recurrent Neural Networks

Problems with RNNs

- **Long Dependencies:**



Recurrent Neural Networks

Problems with RNNs

- **Long Dependencies:**

Consider trying to predict the last word in the text

"I grew up in France (…) I speak fluent **French**."

Recurrent Neural Networks

Problems with RNNs

- **Long Dependencies:**

Consider trying to predict the last word in the text

"I grew up in France (…) I speak fluent **French**."

Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

Problems with RNNs

- **Vanishing Gradients**

During the backward pass of training, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence.

A frequent result of this process is that the gradients are eventually driven to zero, which means that the **initial weights cannot be updated properly!**

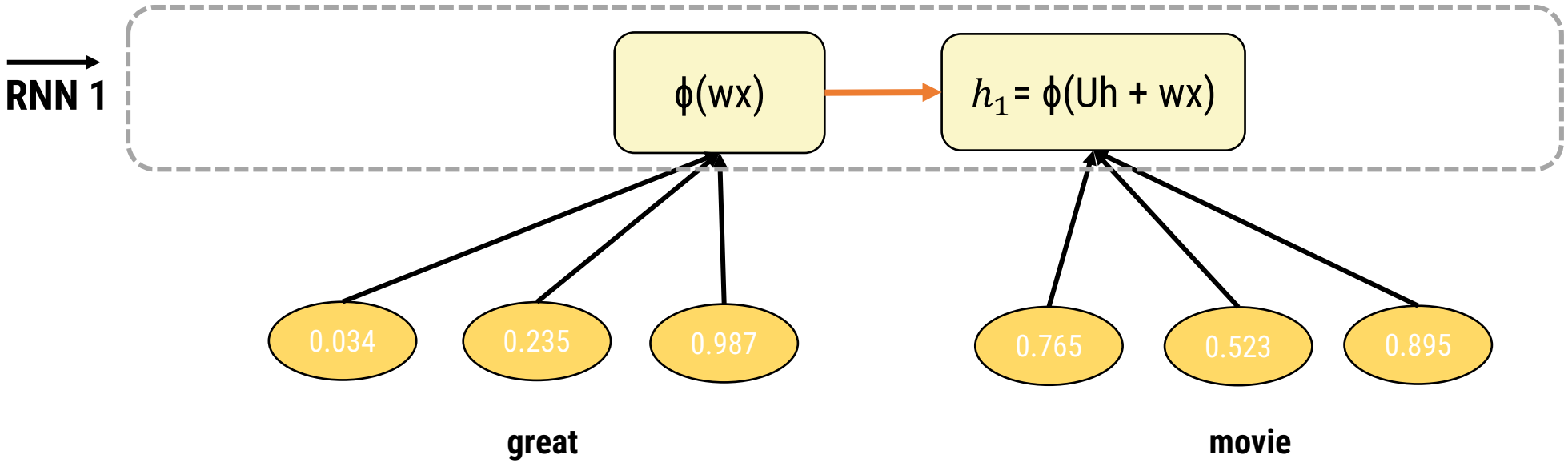
Recurrent Neural Networks

Problems with RNNs

Solution 1: Make the model Bidirectional!

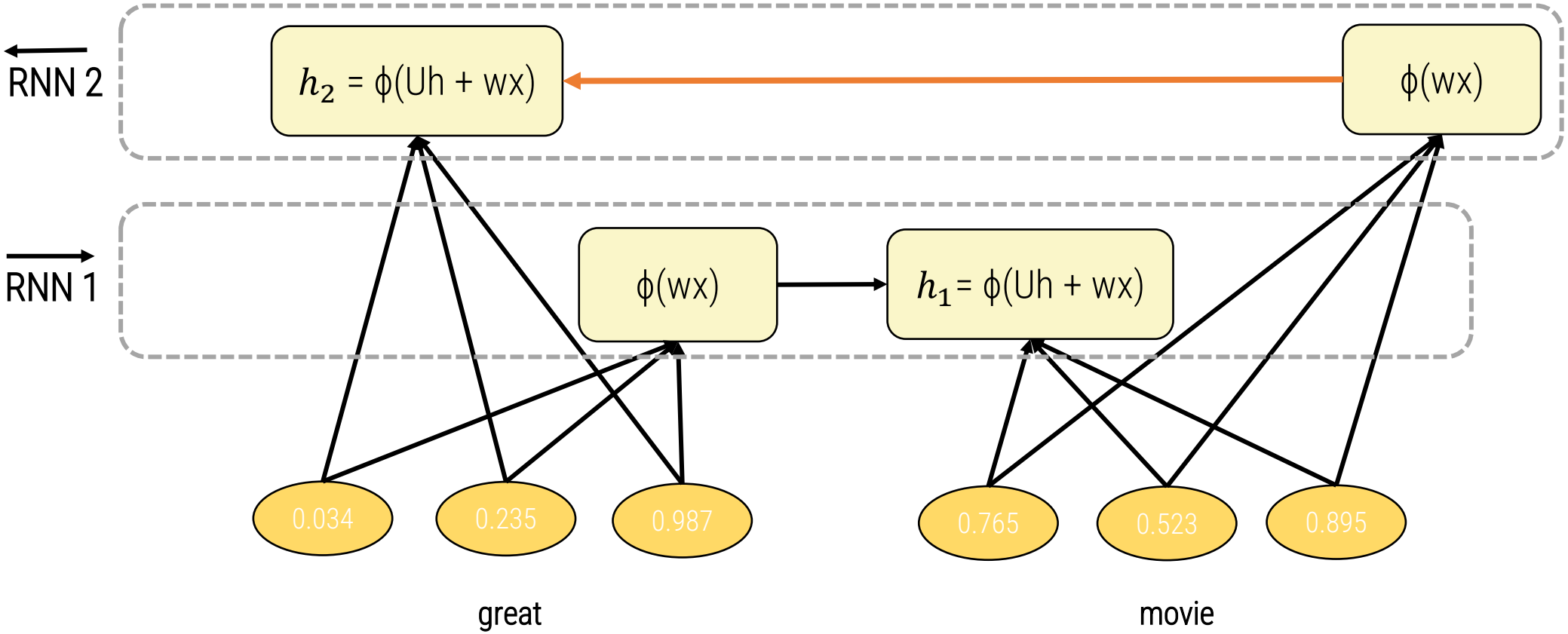
Bidirectional RNNs

Left to Right:

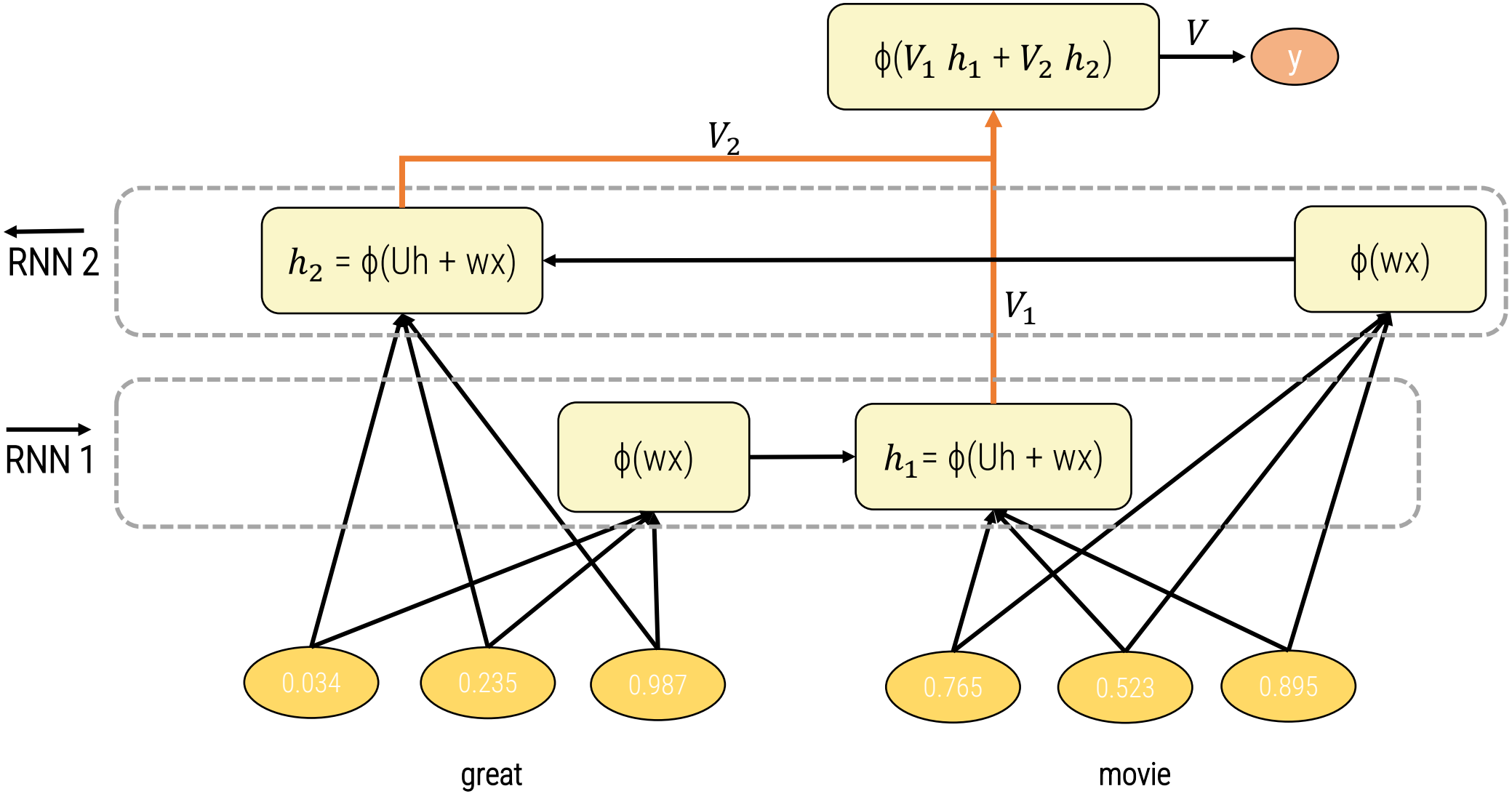


Bidirectional RNNs

Right to Left:



Bidirectional RNNs



NOVA IMS Information Management School

Recurrent Neural Networks

Problems with RNNs

Solution 1: Make the model Bidirectional.

Solution 2: **Long Short Term Memory (LSTM) Neural Networks**

Bibliography

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). **Distributed representations of words and phrases and their compositionality**. arXiv. https://doi.org/10.48550/arXiv.1310.4546
- Dan Jurafsky and James H. Martin. **Speech and Language Processing** 3rd ed. https://web.stanford.edu/~jurafsky/slp3/
- Alammar, J., & Grootendorst, M. (2024). **Hands-On large language models: Language Understanding and Generation**. Sebastopol : O'Reilly Media.
  NOVA IMS Access: https://search.library.novaims.unl.pt/cgi-bin/koha/opac-detail.pl?biblionumber=97234

NOVA IMS Information Management School

Thank you!

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa