MDI 341 - Machine Learning avancé

Machine Learning for Natural Language Processing : Sequence modeling

Matthieu Labeau

matthieu.labeau@telecom-paris.fr

Outline

- Introduction: Language modeling
- N-gram Neural Language Model
- Reccurent Language Model and extensions
- Convolutional architectures in NLP
- NMT and Sequence-to-Sequence models
- References

A central task: Language Modeling

What is language modeling ?

$\rightarrow$ A Language Model (**LM**) can:

- Compute the probability of a sentence:

$$P(\text{What is language modeling ?}) = ?$$

- Compute the probability of a word, given previous ones (**context**):

$$P(\text{modeling}|\text{What is language}) = ?$$

A central task: Language Modeling

What is language modeling ?

$\rightarrow$ A Language Model (**LM**) can:

- Compute the probability of a sentence:

$$P(\text{What is language modeling ?}) = ?$$

- Compute the probability of a word, given previous ones (**context**):

$$P(\text{modeling}|\text{What is language}) = ?$$

$\rightarrow$ Language models are mainly used to rank word and sentences probabilities for various tasks:

$$P(\text{What is language modeling ?}) > P(\text{What language modeling is ?})$$

$$P(\text{modeling}|\text{What is language}) \; > P(\text{learning}|\text{What is language})$$

A central task: Language Modeling

Applications ?

A central task: Language Modeling

Applications ?

$$P(\text{'recognize speech'}) > P(\text{'wreck a nice beach'})$$

- Speech Recognition

A central task: Language Modeling

Applications ?

- Speech Recognition
- Machine Translation

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction
- Optical Character Recognition

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction
- Optical Character Recognition
- Information Retrieval

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction
- Optical Character Recognition
- Information Retrieval
- Summarization

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction
- Optical Character Recognition
- Information Retrieval
- Summarization
- Question Answering

A central task: Language Modeling

Applications ?

- Speech Recognition

- Machine Translation
- Grammar/Spelling Correction
- Optical Character Recognition
- Information Retrieval
- Summarization
- Question Answering
- ... and many other tasks (including anything that necessitate **Text Generation**)

Counting words

How to compute the probability of a word w given a context of previous words h ?

Counting words

How to compute the probability of a word w given a context of previous words h ?

$\rightarrow$ Directly from **word counts**:

$$P(\text{modeling}|\text{What is language}) = \frac{C(\text{What is language modeling})}{C(\text{What is language})}$$

Counting words

How to compute the probability of a word w given a context of previous words h ?

$\rightarrow$ Directly from **word counts**:

$$P(\text{modeling}|\text{What is language}) = \frac{C(\text{What is language modeling})}{C(\text{What is language})}$$

But: There is too many possible sentences and not enough data !

$\rightarrow$ We use the simplifying **Markov assumption**:

$$(\text{Order 1}) \quad P(\text{modeling}|\text{What is language}) \approx \; P(\text{modeling}|\text{language})$$

$$\downarrow$$

$$(\text{Order m}) \quad P(w_i|w_1, ..., w_{i-1}) \approx \; P(w_i|w_{i-m}, ..., w_{i-1})$$

N-gram models

With the **Markov assumption** and the **chain rule**, we obtain simple language models:

$$\text{(Order 1: Unigram)} \qquad P(w_1, ..., w_n) \approx \prod_{i=1}^{n} P(w_i)$$

$$\text{(Order 2 : Bigram)} \quad P(w_1, ..., w_n) \approx P(w_1) \prod_{i=2}^{n} P(w_i | w_{i-1})$$

N-gram models

With the **Markov assumption** and the **chain rule**, we obtain simple language models:

$$(\text{Order 1: Unigram}) \qquad P(w_1, ..., w_n) \approx \prod_{i=1}^{n} P(w_i)$$

$$(\text{Order 2 : Bigram}) \quad P(w_1, ..., w_n) \approx P(w_1) \prod_{i=2}^{n} P(w_i | w_{i-1})$$

$\rightarrow$ They can be extended to trigram, 4-gram, 5-gram LMs

- Because of long-term dependancies, this is in general insufficient to model language !
- However, n-gram models are still used in numerous applications

N-gram models: Estimating probabilities

$$P(\text{What is language modeling}) \approx P(\text{What}) \times P(\text{is} \mid \text{What})$$
$$\times P(\text{language} \mid \text{is}) \times P(\text{modeling} \mid \text{language})$$

Some practical advice:

N-gram models: Estimating probabilities

$$P(\text{What is language modeling}) \approx P(\text{What}) \times P(\text{is} \mid \text{What})$$
$$\times P(\text{language} \mid \text{is}) \times P(\text{modeling} \mid \text{language})$$

Some practical advice:

- Quantities are small and summing is easier than multiplying: we do everything in **logarithms** !

N-gram models: Estimating probabilities

$$P(\text{What is language modeling}) \approx P(\text{What}) \times P(\text{is} \mid \text{What})$$
$$\times P(\text{language} \mid \text{is}) \times P(\text{modeling} \mid \text{language})$$

Some practical advice:

- Quantities are small and summing is easier than multiplying: we do everything in **logarithms** !
- Be careful to **tokenization**:

$$\ldots \text{ language modeling } ? \neq \ldots \text{ language modeling?}$$

N-gram models: Estimating probabilities

$$P(\text{What is language modeling}) \approx P(\text{What}) \times P(\text{is} \mid \text{What})$$
$$\times P(\text{language} \mid \text{is}) \times P(\text{modeling} \mid \text{language})$$

Some practical advice:

- Quantities are small and summing is easier than multiplying: we do everything in **logarithms** !
- Be careful to **tokenization**:

$$\ldots \text{ language modeling } ? \neq \ldots \text{ language modeling?}$$

- Represent the beginning and end of sentences with special tokens:

$$<\text{s}> \text{ What is language modeling ? } </\text{s}>$$

N-gram models: Estimating probabilities

$$P(\text{What is language modeling}) \approx P(\text{What}) \times P(\text{is} \mid \text{What})$$
$$\times P(\text{language} \mid \text{is}) \times P(\text{modeling} \mid \text{language})$$

Some practical advice:

- Quantities are small and summing is easier than multiplying: we do everything in **logarithms** !
- Be careful to **tokenization**:

$$\ldots\ \text{language modeling}\ ? \neq \ldots\ \text{language modeling?}$$

- Represent the beginning and end of sentences with special tokens:

$$<\text{s}>\ \text{What is language modeling ?}\ </\text{s}>$$

- Represent rare words with another special token:

$$\text{What is language}\ <\text{UNK}>\ ?$$

N-gram models: Evaluation

- **Extrinsic** evaluation:

  Apply the models that we want to compare on a a task, and use the related metric !

  $\rightarrow$ It's time consuming, and can vary a lot depending on tasks !

N-gram models: Evaluation

- **Extrinsic** evaluation:

  Apply the models that we want to compare on a a task, and use the related metric !

  $\rightarrow$ It's time consuming, and can vary a lot depending on tasks !

- **Intrinsic** evaluation - **Perplexity**:

  A measure of *uncertainty*: how surprised is the model by the data ?

  $\rightarrow$ On test data, lower is better !

$$Perplexity(w_1, ..., w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_{i-m}, ..., w_{i-1})}}$$

  - Can be interpreted as a *branching factor*
  - But is **tied to a particular vocabulary** $\mathcal{V}$
  - Is not great at tracking progress and measuring performance

N-gram models: Generalization

- **Zero probability n-grams**:
  In a corpus, occuring n-grams represent a very small part of the possible n-grams.
  $\rightarrow$ In Shakespeare's work (Example from Dan Jurafsky):
  - 884,647 tokens and a vocabulary of 29,066 words
  - 300K out of the 844M bigrams appear (0,04%) !

$\rightarrow$ Zero counts imply zero probabilities and no perplexity !

N-gram models: Generalization

- **Zero probability n-grams**:

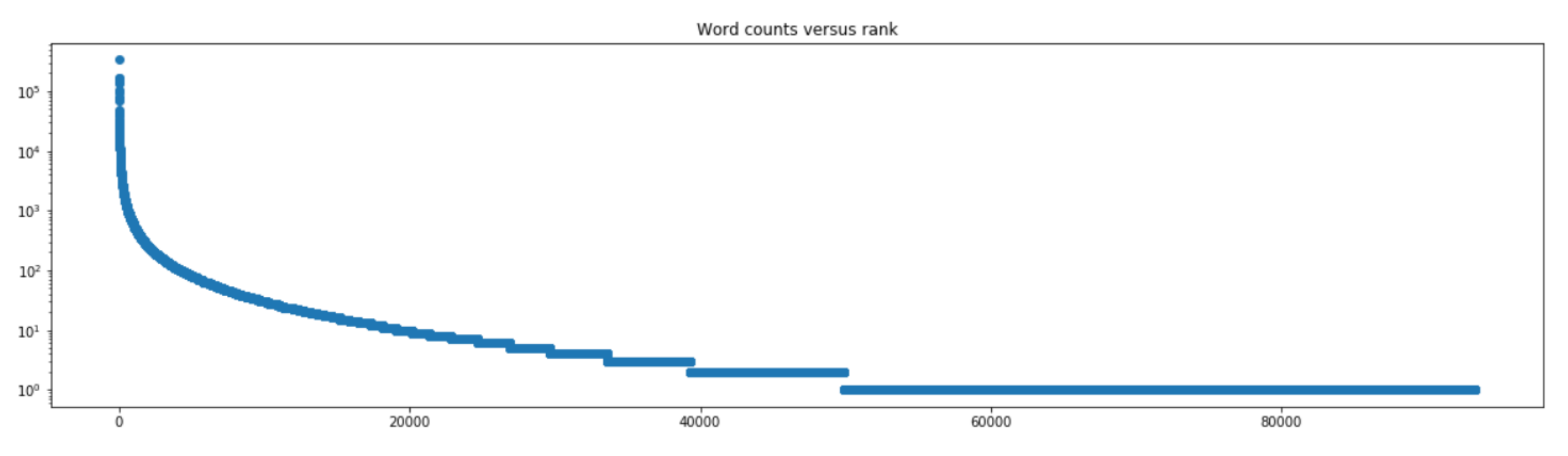
  In a corpus, occuring n-grams represent a very small part of the possible n-grams.

  $\rightarrow$ In Shakespeare's work (Example from Dan Jurafsky):

  - 884,647 tokens and a vocabulary of 29,066 words
  - 300K out of the 844M bigrams appear (0,04%) !

$\rightarrow$ Zero counts imply zero probabilities and no perplexity !

- Tied to **Zipf's Law**: $frequency \propto 1/rank$



N-gram models: Smoothing

Intuition: 'Steal' probability from data seen during training, to make counts non-sparse !

N-gram models: Smoothing

Intuition: 'Steal' probability from data seen during training, to make counts non-sparse !

- **Laplace Smoothing:**

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |\mathcal{V}|}$$

N-gram models: Smoothing

Intuition: 'Steal' probability from data seen during training, to make counts non-sparse !

- **Laplace Smoothing:**

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |\mathcal{V}|}$$

- **Backoff** and **Interpolation**:
  When you don't have enough information at order n, back-off to, or interpolate with, smaller orders.

N-gram models: Smoothing

Intuition: 'Steal' probability from data seen during training, to make counts non-sparse !

- **Laplace Smoothing:**

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |\mathcal{V}|}$$

- **Backoff** and **Interpolation**:
  When you don't have enough information at order n, back-off to, or interpolate with, smaller orders.
- **Kneyser-Ney smoothing**:
  Considers probabilities in relation to higher orders: we need to look at $P(\text{San Francisco})$ to correctly use $P(\text{francisco})$
  $\rightarrow$ Still widely used as fast and easy to train baseline !

N-gram *neural* models

A Neural Probabilistic Language Model (Bengio et al, 2003)

Applied to speech recognition *(Schwenk and Gauvain, 2002)* and machine translation.

Main ideas:

N-gram *neural* models

A Neural Probabilistic Language Model (Bengio et al, 2003)

Applied to speech recognition *(Schwenk and Gauvain, 2002)* and machine translation.

Main ideas:

- **Continuous feature vectors:** Each word is represented by a continuous feature vector of dimension $d \ll |\mathcal{V}|$

N-gram *neural* models

A Neural Probabilistic Language Model (Bengio et al, 2003)

Applied to speech recognition *(Schwenk and Gauvain, 2002)* and machine translation.

Main ideas:

- **Continuous feature vectors:** Each word is represented by a continuous feature vector of dimension $d \ll |\mathcal{V}|$
- **Continuous probability function**: The probability of the next word is expressed as a continuous function of the features of the word in the current context - using a feedforward neural network

N-gram *neural* models

A Neural Probabilistic Language Model (Bengio et al, 2003)

Applied to speech recognition *(Schwenk and Gauvain, 2002)* and machine translation.

Main ideas:

- **Continuous feature vectors:** Each word is represented by a continuous feature vector of dimension $d \ll |\mathcal{V}|$
- **Continuous probability function**: The probability of the next word is expressed as a continuous function of the features of the word in the current context - using a feedforward neural network
- **Joint learning**: Both the parameters of the word representation and of the probability function are learnt jointly.

NPLM: Joint learning

Why should it work ?

$\rightarrow$ Continuity !

The probability function is smooth, implying that a small change in the feature vector will induce a small change in the probability.

Hence, the updates caused by having the following sentence:

$$\text{A dog was running in a room}$$

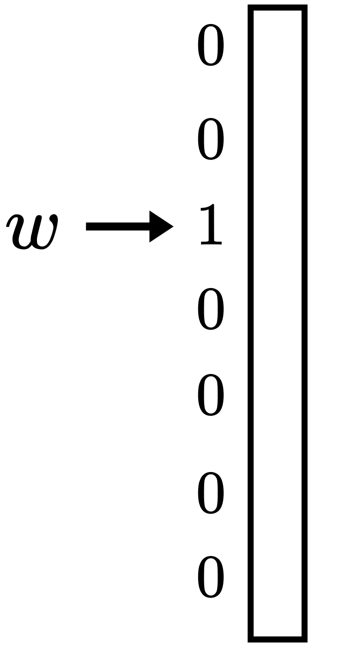
in the training data will increase the probability of all 'neighbor' sentences. Having also the following sentence:

$$\text{The cat was running in a room}$$

will make the features of the words (dog, cat) get close to each other.

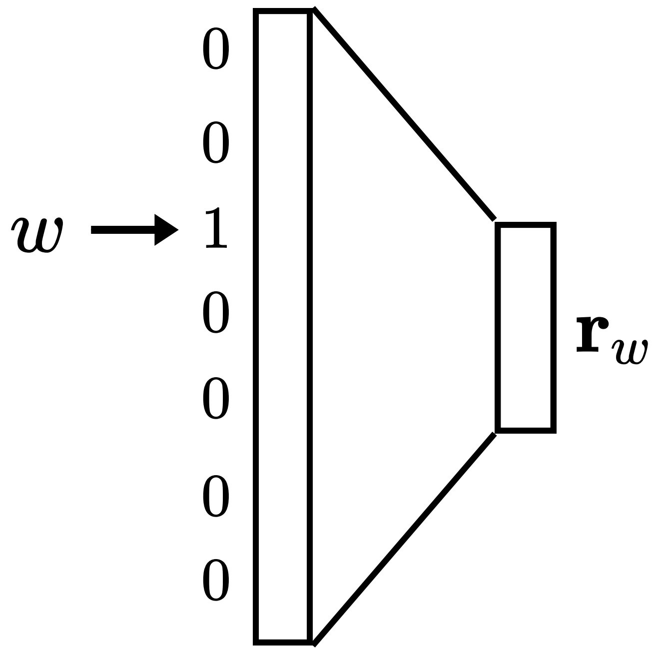
NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.



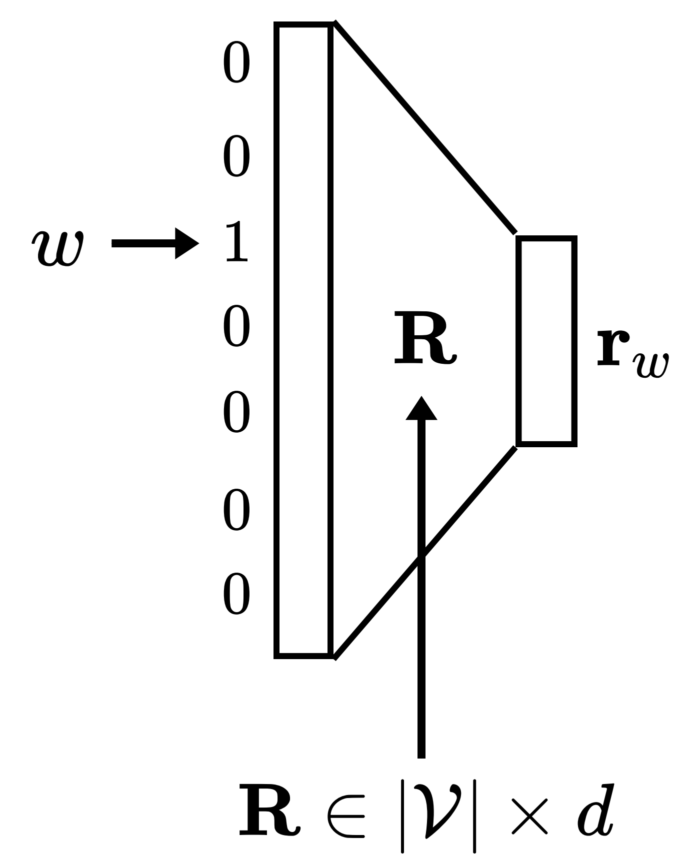
NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.



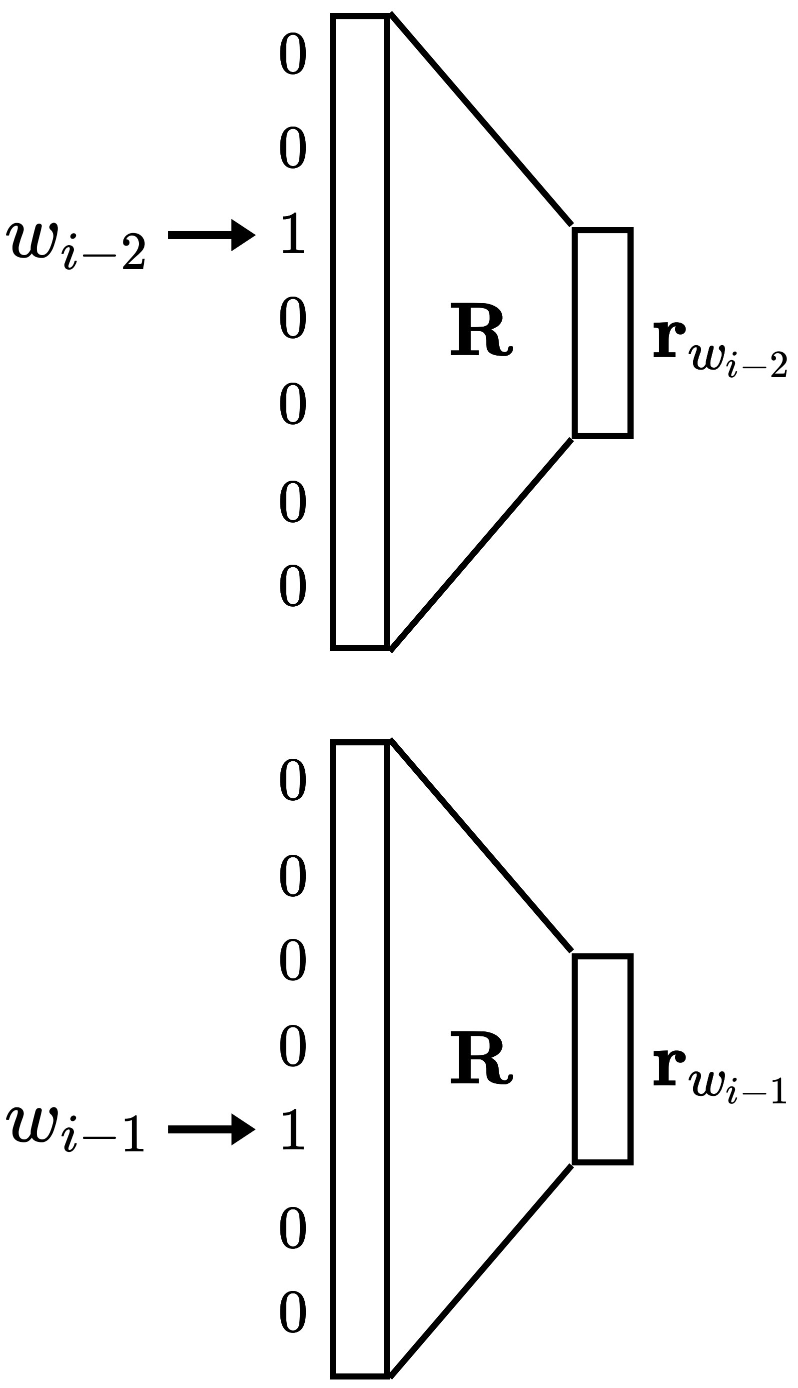
NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.
- The parameters of the weight matrix $\mathbf{R}$ are what we call the **word embeddings**.



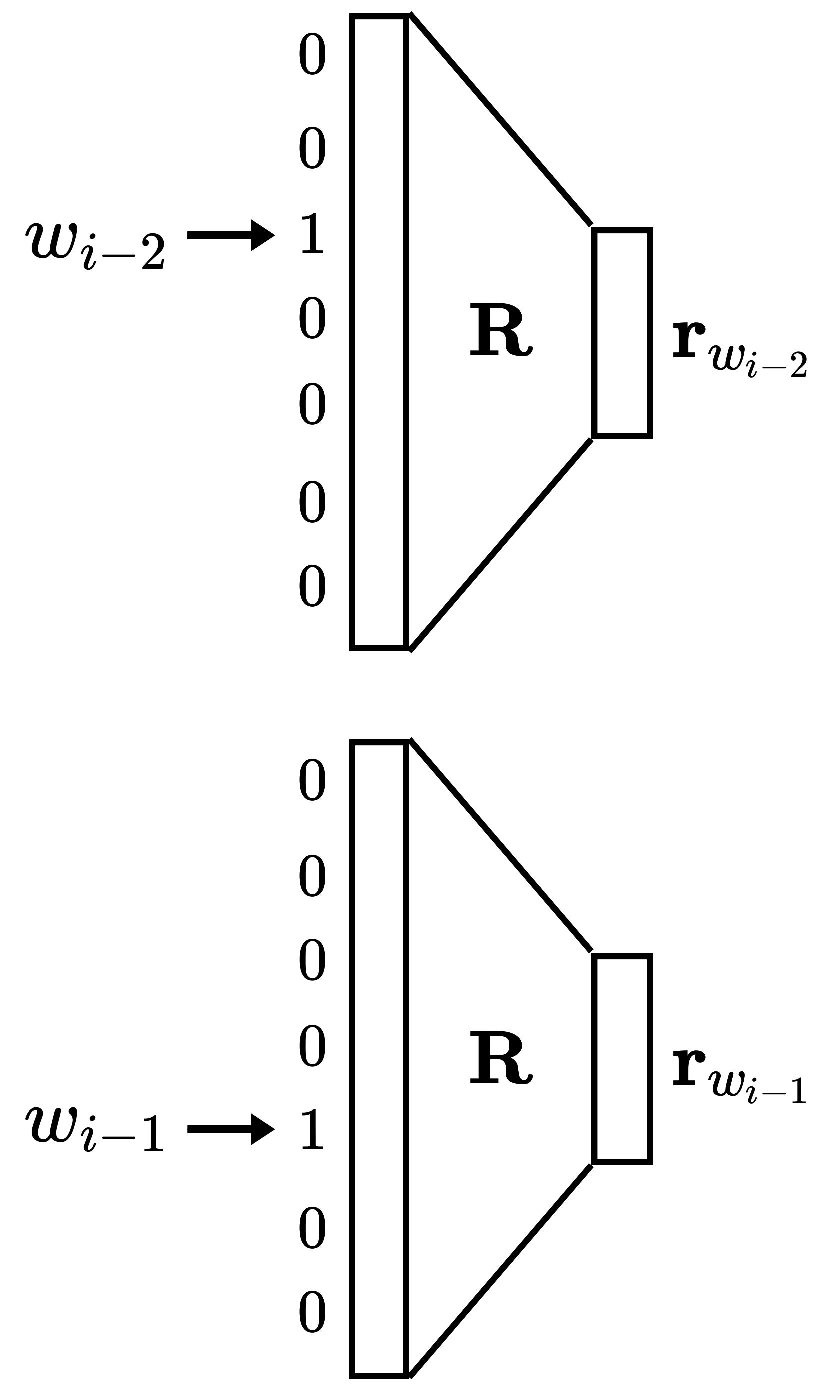
NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.
- The parameters of the weight matrix $\mathbf{R}$ are what we call the **word embeddings**.



NPLM: Projecting words

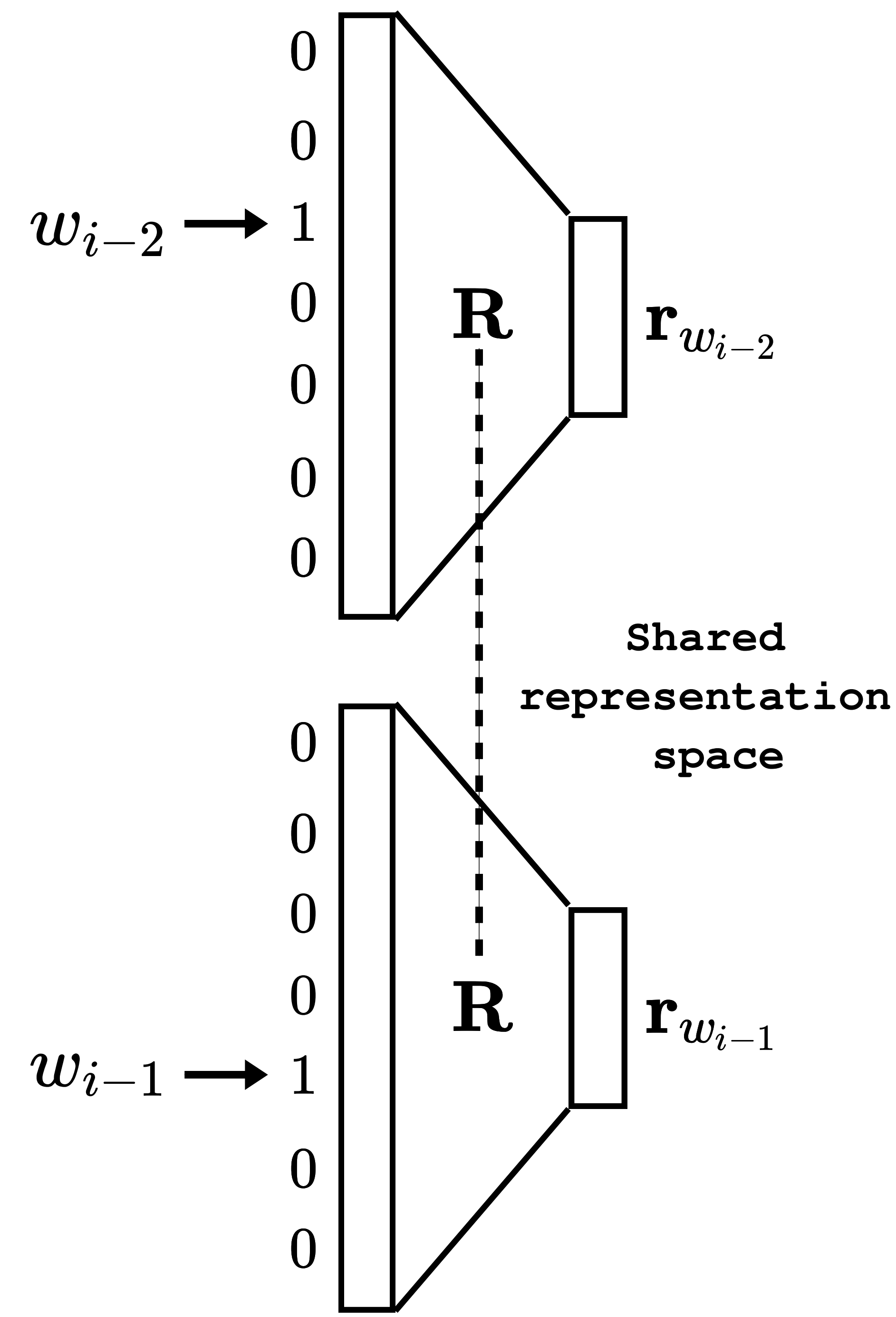
- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.
- The parameters of the weight matrix $\mathbf{R}$ are what we call the **word embeddings**.



- For a trigram model, we project two words

NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.
- The parameters of the weight matrix $\mathbf{R}$ are what we call the **word embeddings**.

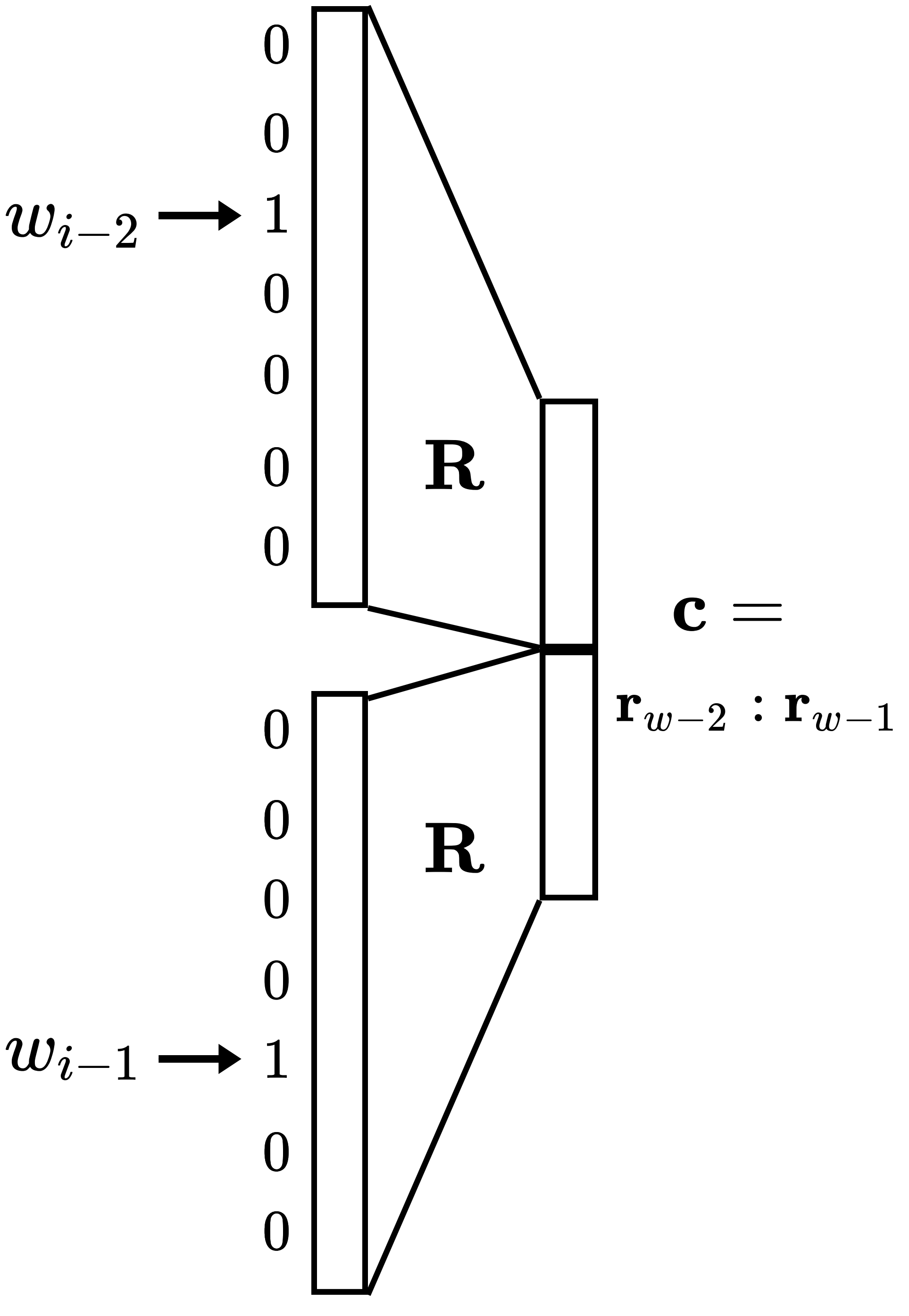


- For a trigram model, we project two words

NPLM: Projecting words

- The first layer is the **vocabulary** $\mathcal{V}$: the input is a one-hot vector.
- This layer is densely connected to a smaller continuous layer, of dimension d.
- The parameters of the weight matrix $\mathbf{R}$ are what we call the **word embeddings**.

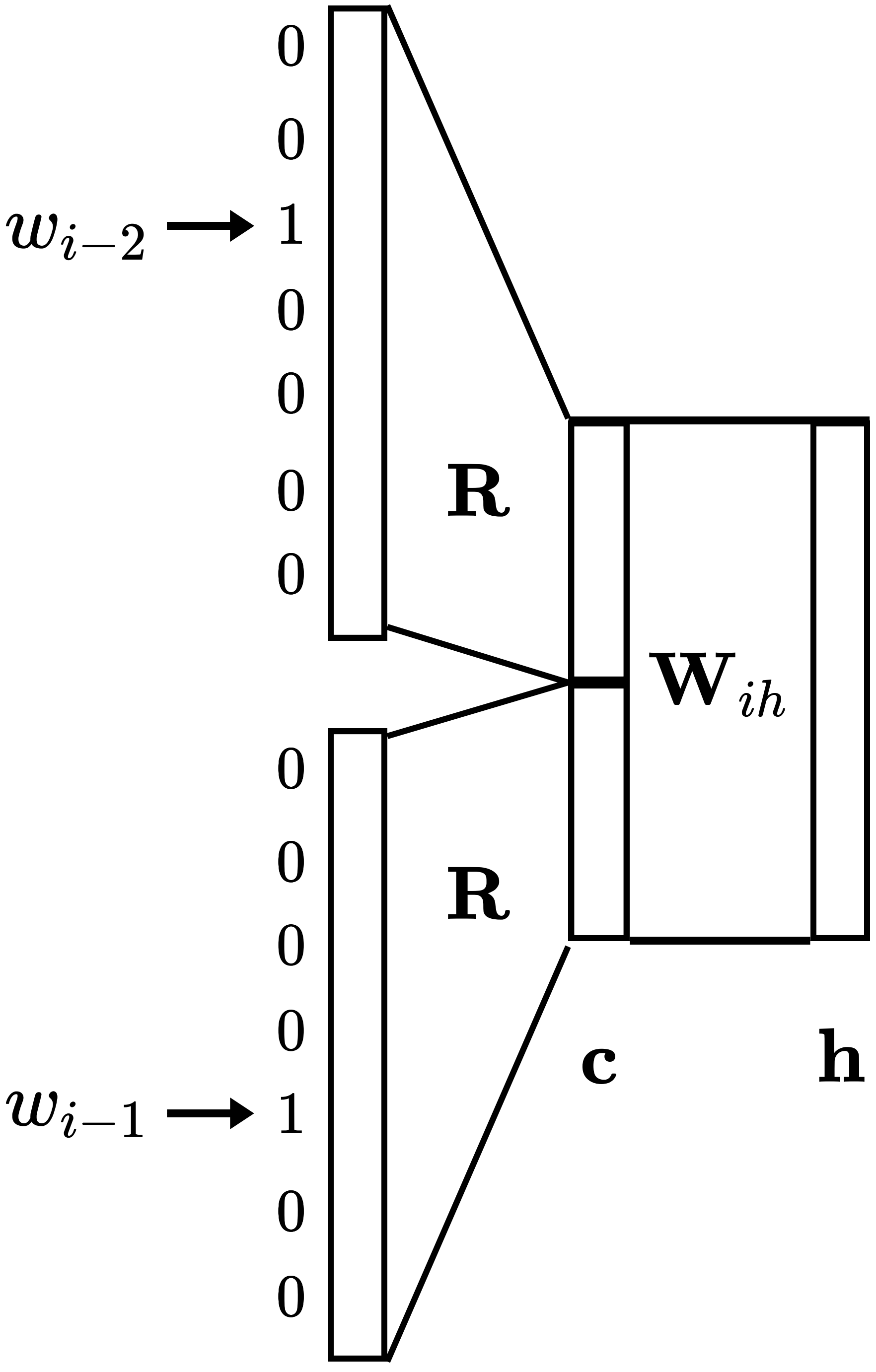
- For a trigram model, we project two words
- These are merged into a single vector $\mathbf{c}$ representing the context



NPLM: Estimating probabilities

- Given the context representation $\mathbf{c}$, create a hidden representation:

$$\mathbf{h} = \phi(\mathbf{W}_{ih}\mathbf{c})$$



NPLM: Estimating probabilities

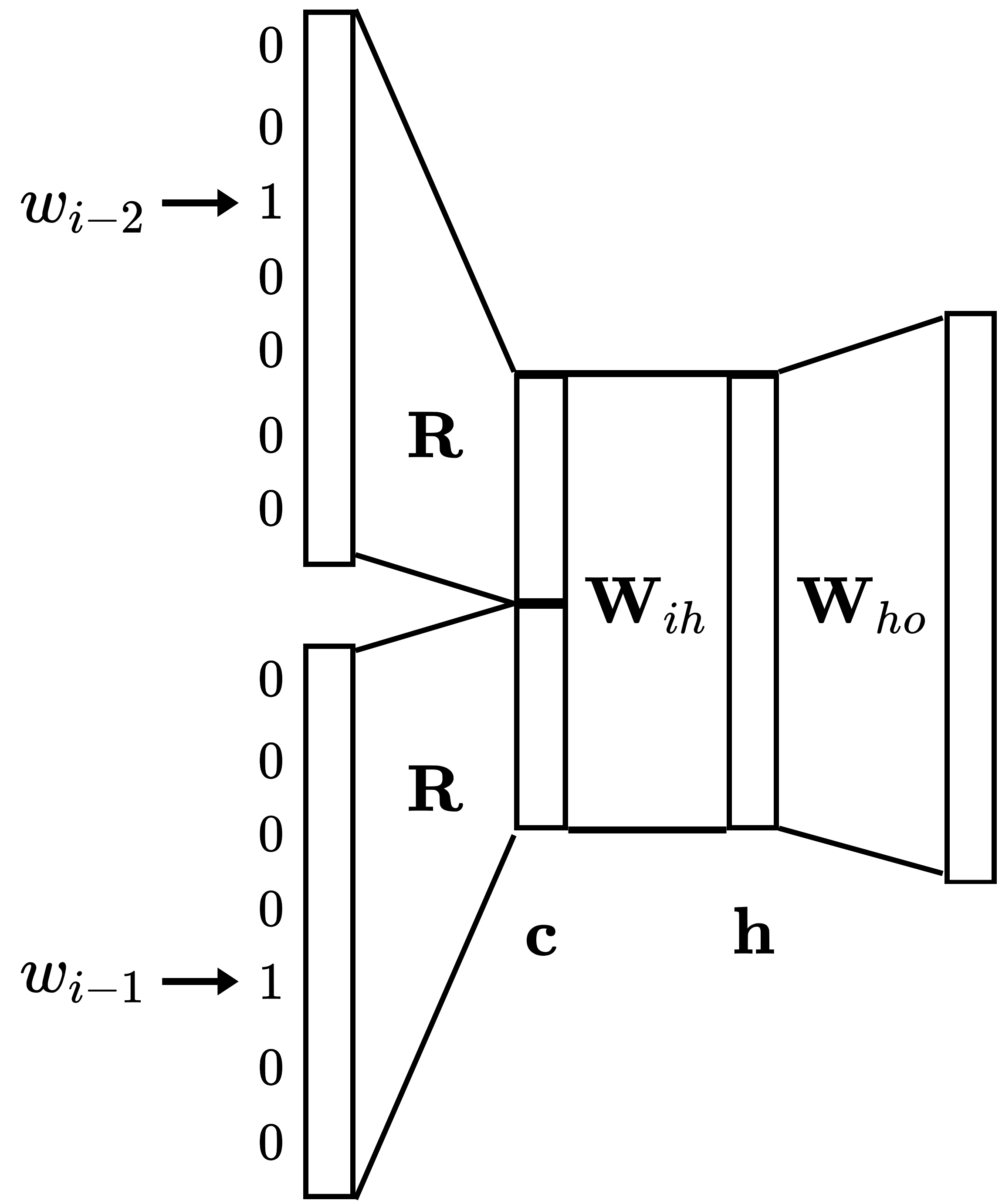
- Given the context representation $\mathbf{c}$, create a hidden representation:

$$\mathbf{h} = \phi(\mathbf{W}_{ih}\mathbf{c})$$

- Then, estimate probabilities for all words in $\mathcal{V}$ given $\mathbf{h}$, with the softmax function:

$$\mathbf{o} = \mathbf{W}_{ho}\mathbf{h}$$

$$P(w|w_{i-1}, w_{i-2}) = \frac{\exp(\mathbf{o}_w)}{\sum_{w' \in \mathcal{V}} \exp(\mathbf{o}_{w'})}$$



NPLM: Estimating probabilities

- Given the context representation $\mathbf{c}$, create a hidden representation:

$$\mathbf{h} = \phi(\mathbf{W}_{ih}\mathbf{c})$$

- Then, estimate probabilities for all words in $\mathcal{V}$ given $\mathbf{h}$, with the softmax function:

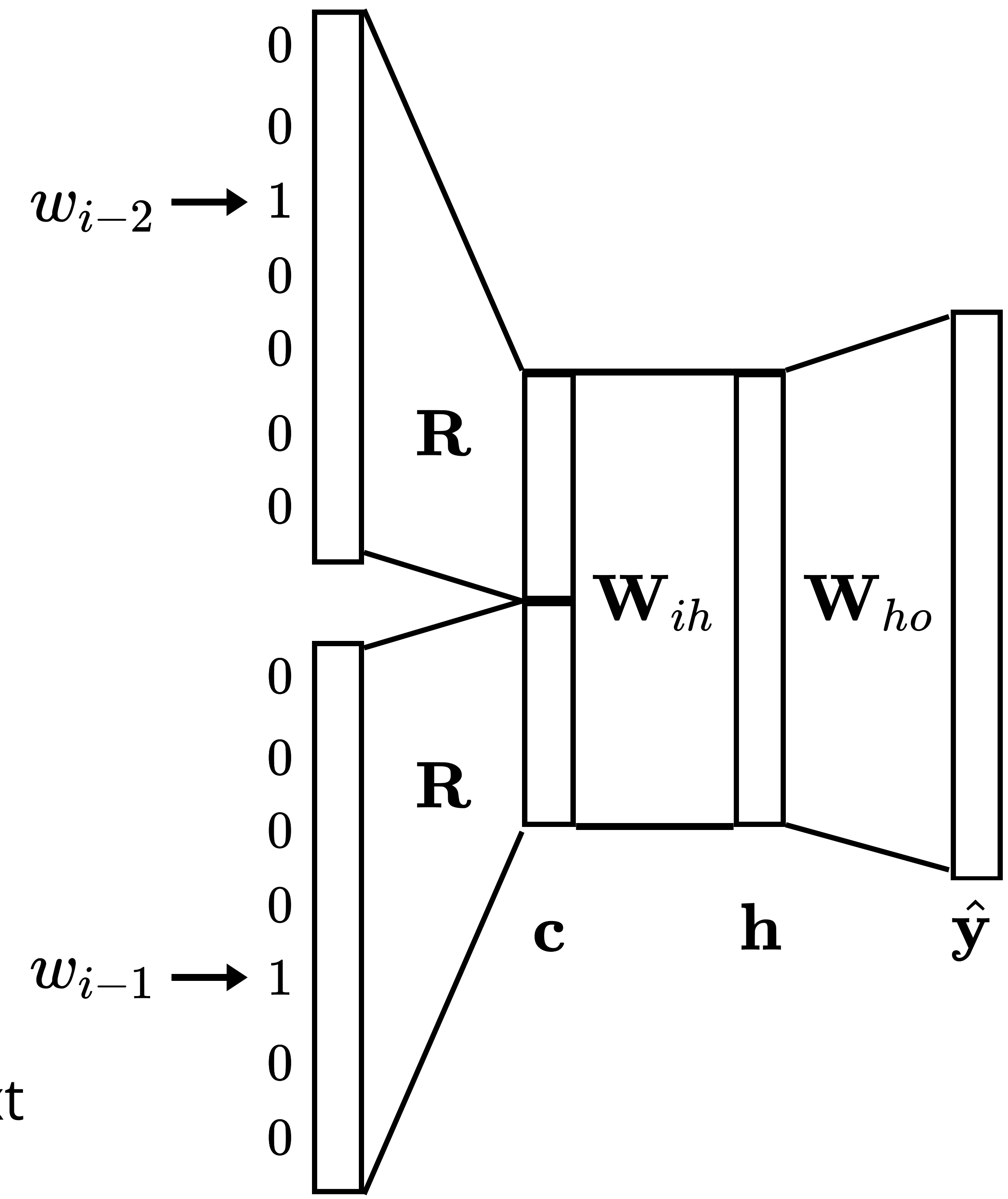
$$\mathbf{o} = \mathbf{W}_{ho}\mathbf{h}$$

$$P(w|w_{i-1}, w_{i-2}) = \frac{\exp(\mathbf{o}_w)}{\sum_{w' \in \mathcal{V}} \exp(\mathbf{o}_{w'})}$$

- We note the predicted vector as $\hat{\mathbf{y}}$:

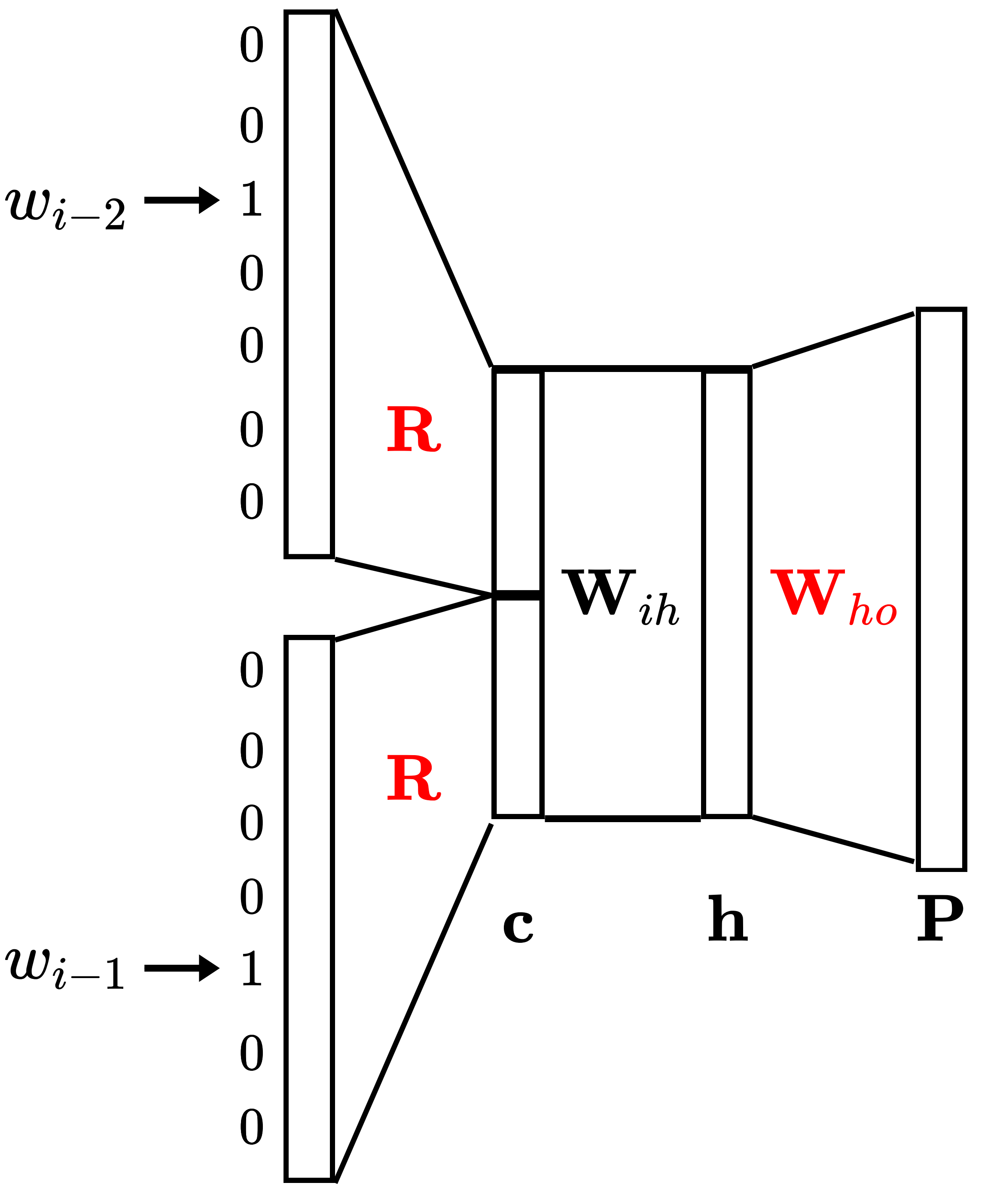
$$\hat{\mathbf{y}}_{i-1} = (P(w|w_{i-2}, w_{i-1}))_{w \in \mathcal{V}}$$

and use the *argmax* to choose the next word.



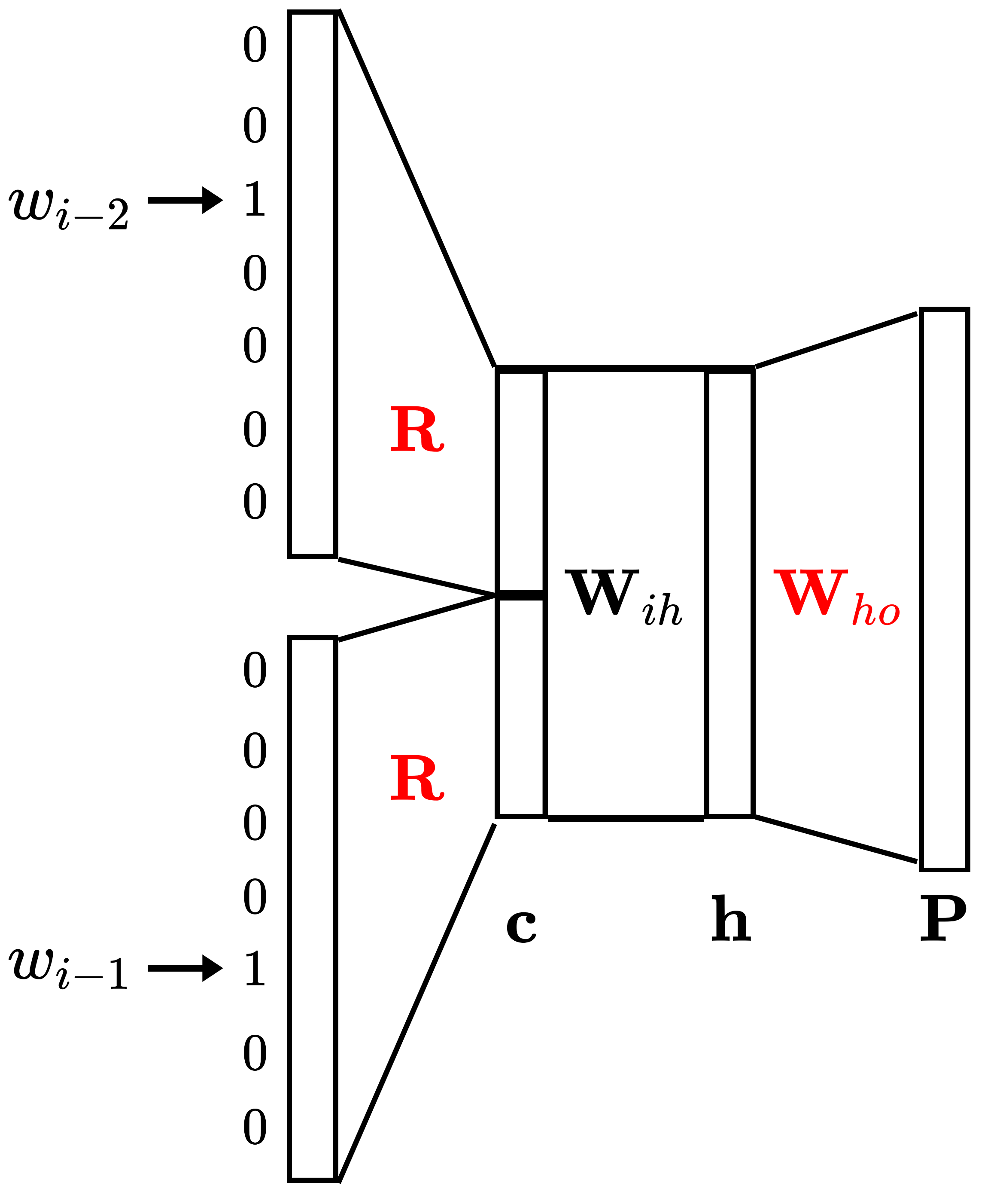
NPLM: Assessment

- Probability estimation is based on the similarity among the feature vectors



NPLM: Assessment

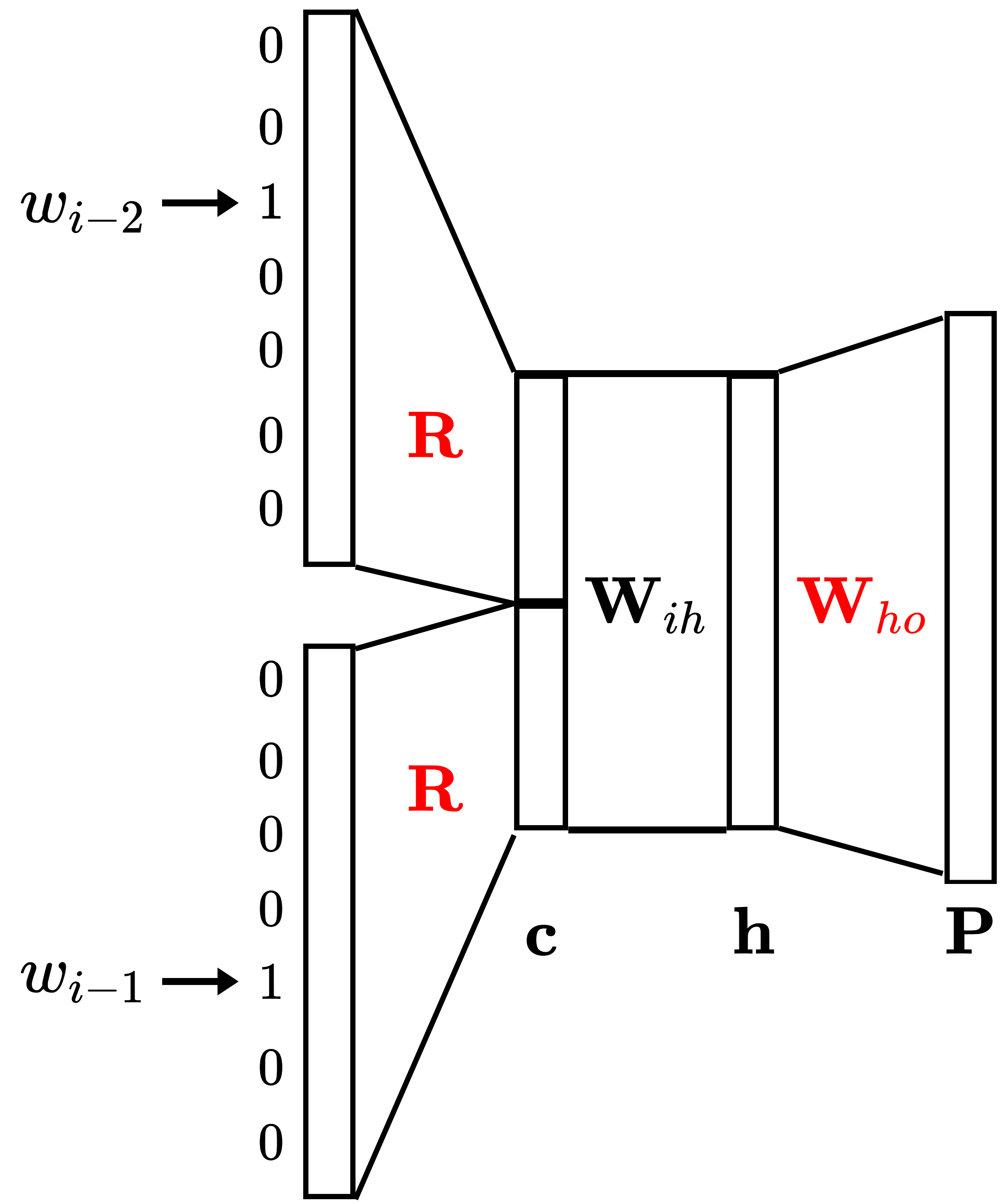
- Probability estimation is based on the similarity among the feature vectors
- Projecting in continuous spaces reduces the **sparsity issue.**



NPLM: Assessment

- Probability estimation is based on the similarity among the feature vectors
- Projecting in continuous spaces reduces the **sparsity issue.**

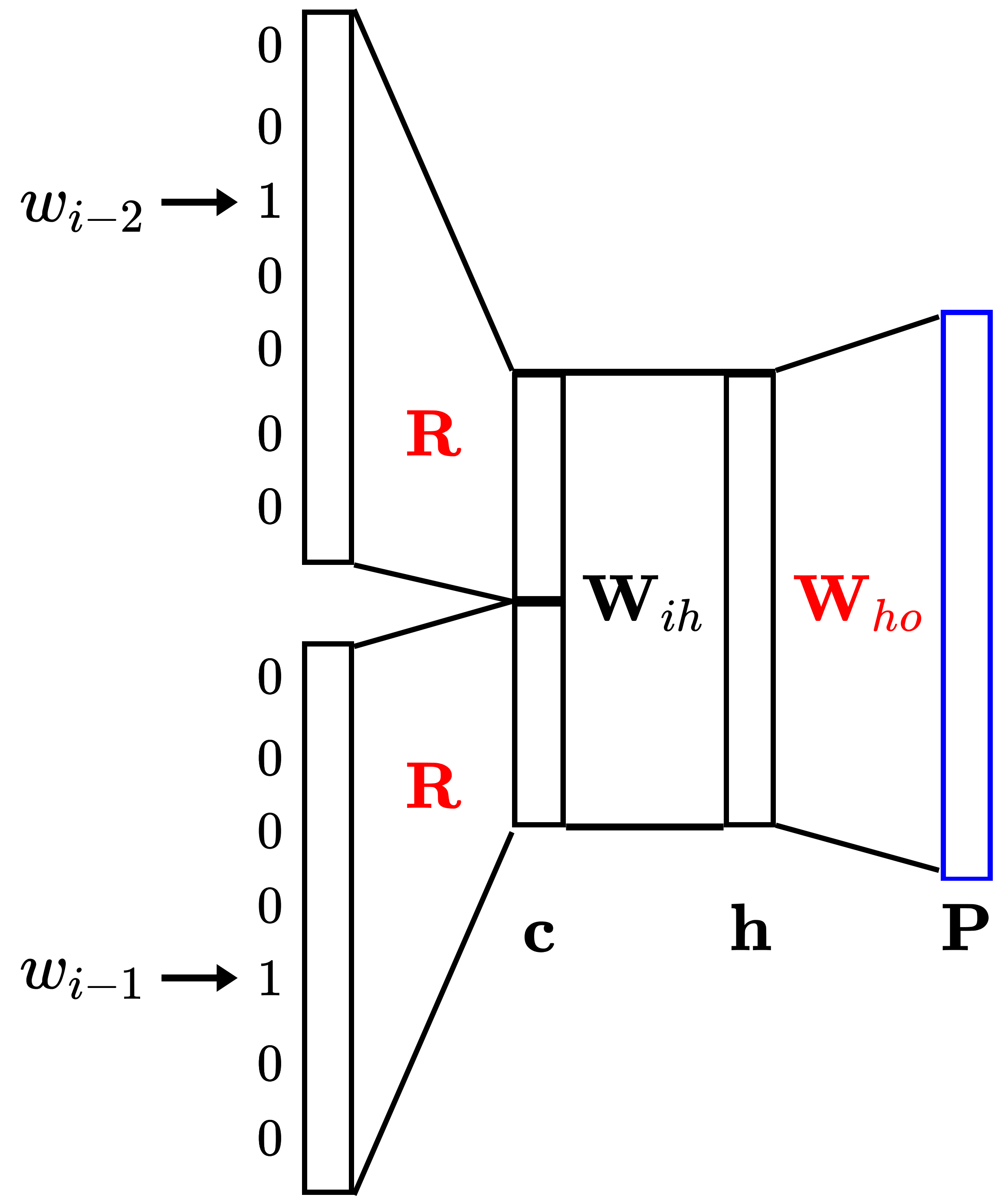
- Increasing the number of input words does not affect the complexity of the model



NPLM: Assessment

- Probability estimation is based on the similarity among the feature vectors
- Projecting in continuous spaces reduces the **sparsity issue.**

- Increasing the number of input words does not affect the complexity of the model
- The bottleneck is the **output vocabulary size** **!**



NPLM: Learning bottleneck

- The *projection* and *prediction* are learned jointly: $\theta = (\mathbf{R}, \mathbf{W}_{ih}, \mathbf{W}_{ho})$, the parameters of the model, are trained through the **MLE** objective, which equates minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$:

$$NLL(\theta) = -\sum_{i=1}^{N} l(\hat{\mathbf{y}}_{i-1}|\theta, w_i) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-2}, w_{i-1})$$

NPLM: Learning bottleneck

- The *projection* and *prediction* are learned jointly: $\theta = (\mathbf{R}, \mathbf{W}_{ih}, \mathbf{W}_{ho})$, the parameters of the model, are trained through the **MLE** objective, which equates minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$:

$$NLL(\theta) = -\sum_{i=1}^{N} l(\hat{\mathbf{y}}_{i-1}|\theta, w_i) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-2}, w_{i-1})$$

- The updates are computed using:

$$\frac{\delta}{\delta\theta} \log P_\theta(w_i|w_{i-2}, w_{i-1}) \ = \frac{\delta}{\delta\theta}\mathbf{o}_{w_i} - \sum_{w\in\mathcal{V}} P_\theta(w|w_{i-2}, w_{i-1})\frac{\delta}{\delta\theta}\mathbf{o}_w$$

NPLM: Learning bottleneck

- The *projection* and *prediction* are learned jointly: $\theta = (\mathbf{R}, \mathbf{W}_{ih}, \mathbf{W}_{ho})$, the parameters of the model, are trained through the **MLE** objective, which equates minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$:

$$NLL(\theta) = -\sum_{i=1}^{N} l(\hat{\mathbf{y}}_{i-1}|\theta, w_i) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-2}, w_{i-1})$$

- The updates are computed using:

$$\frac{\delta}{\delta\theta} \log P_\theta(w_i|w_{i-2}, w_{i-1}) \; = \frac{\delta}{\delta\theta}\mathbf{o}_{w_i} - \sum_{w \in \mathcal{V}} P_\theta(w|w_{i-2}, w_{i-1}) \frac{\delta}{\delta\theta}\mathbf{o}_w$$

- The first term increases the conditional log-likelihood of w_i given w_{i-2}, w_{i-1}

NPLM: Learning bottleneck

- The *projection* and *prediction* are learned jointly: $\theta = (\mathbf{R}, \mathbf{W}_{ih}, \mathbf{W}_{ho})$, the parameters of the model, are trained through the **MLE** objective, which equates minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^N$:

$$NLL(\theta) = -\sum_{i=1}^{N} l(\hat{\mathbf{y}}_{i-1}|\theta, w_i) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-2}, w_{i-1})$$

- The updates are computed using:

$$\frac{\delta}{\delta\theta} \log P_\theta(w_i|w_{i-2}, w_{i-1}) = \frac{\delta}{\delta\theta}\mathbf{o}_{w_i} - \sum_{w\in\mathcal{V}} P_\theta(w|w_{i-2}, w_{i-1})\frac{\delta}{\delta\theta}\mathbf{o}_w$$

- The first term increases the conditional log-likelihood of w_i given w_{i-2}, w_{i-1}
- The second decreases the conditional log-likelihood of all the other words $w \in \mathcal{V}$ - and implies a double summation on $\mathcal{V}$!

NPLM: Learning bottleneck

- The *projection* and *prediction* are learned jointly: $\theta = (\mathbf{R}, \mathbf{W}_{ih}, \mathbf{W}_{ho})$, the parameters of the model, are trained through the **MLE** objective, which equates minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^N$:

$$NLL(\theta) = -\sum_{i=1}^{N} l(\hat{\mathbf{y}}_{i-1}|\theta, w_i) = -\sum_{i=1}^{N} \log P_\theta(w_i|w_{i-2}, w_{i-1})$$

- The updates are computed using:

$$\frac{\delta}{\delta\theta} \log P_\theta(w_i|w_{i-2}, w_{i-1}) = \frac{\delta}{\delta\theta}\mathbf{o}_{w_i} - \sum_{w\in\mathcal{V}} P_\theta(w|w_{i-2}, w_{i-1})\frac{\delta}{\delta\theta}\mathbf{o}_w$$

- The first term increases the conditional log-likelihood of w_i given w_{i-2}, w_{i-1}
- The second decreases the conditional log-likelihood of all the other words $w \in \mathcal{V}$ - and implies a double summation on $\mathcal{V}$!

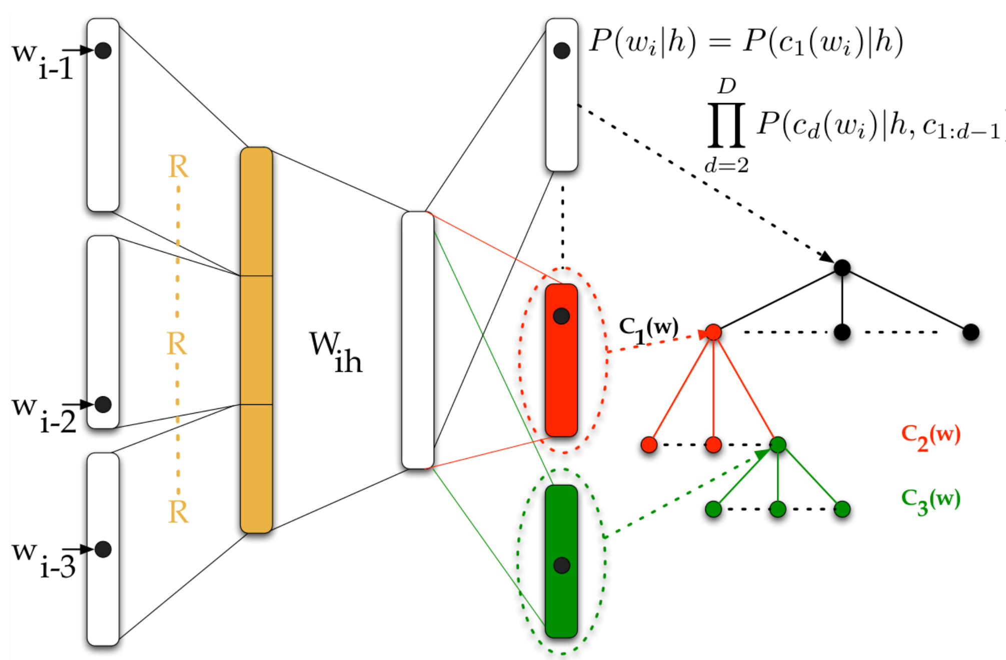
$\rightarrow$ The **softmax** causes this computational **bottleneck** !

NPLM: Dealing with that bottleneck ?

- Use a class-based language model, making **hierarchical** predictions:
  $\rightarrow$ Replace complexity in $O(|\mathcal{V}|)$ by $O(\log |\mathcal{V}|)$

NPLM: Dealing with that bottleneck ?

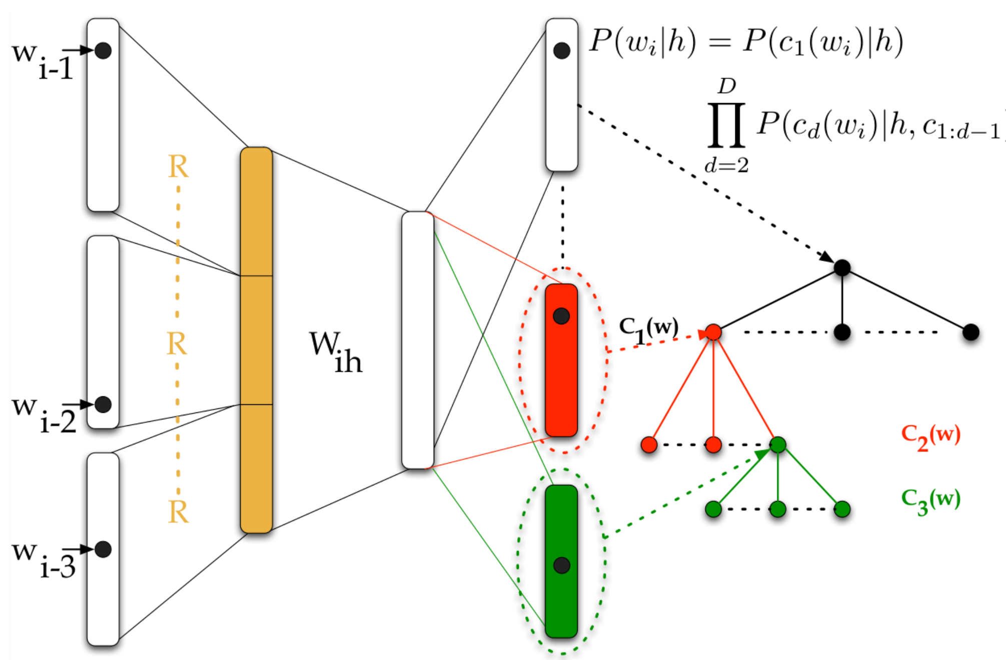
- Use a class-based language model, making **hierarchical** predictions:
  $\rightarrow$ Replace complexity in $O(|\mathcal{V}|)$ by $O(\log |\mathcal{V}|)$



From *Structured Output Layer neural network Language Model* (Hai-Son Le, Alexandre Allauzen, François Yvon, 2012)

NPLM: Dealing with that bottleneck ?

- Use a class-based language model, making **hierarchical** predictions:
  $\rightarrow$ Replace complexity in $O(|\mathcal{V}|)$ by $O(\log |\mathcal{V}|)$

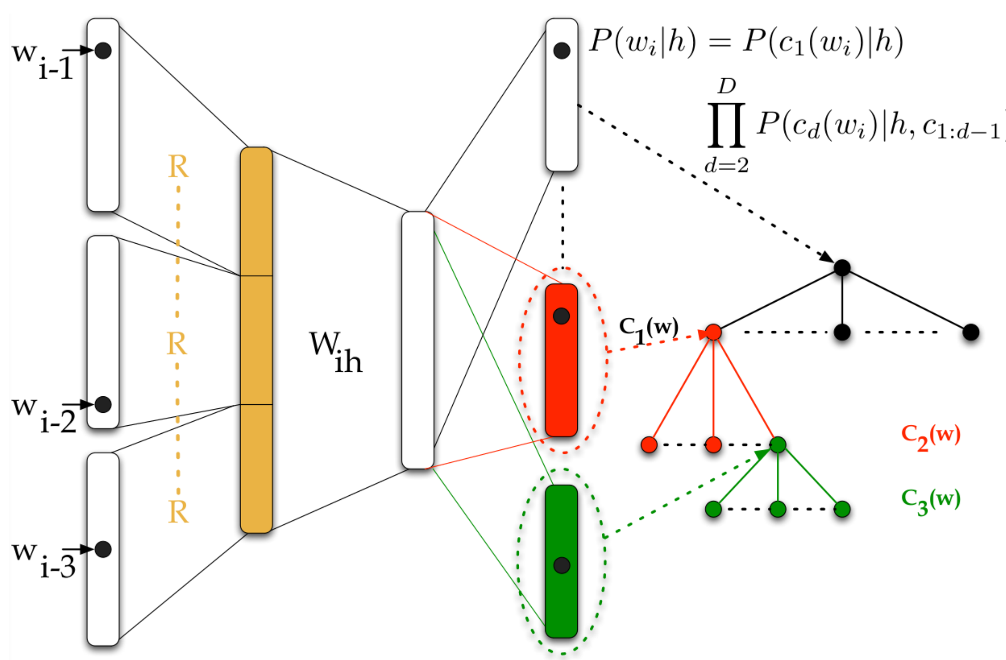


From *Structured Output Layer neural network Language Model* (Hai-Son Le, Alexandre Allauzen, François Yvon, 2012)

- Use **sampling-based** methods, aiming at replacing the sum over the vocabulary by k samples when computing the gradient ($k \ll |\mathcal{V}|$):
  $\rightarrow$ **Importance Sampling**, **Noise-Contrastive Estimation**, ...

NPLM: Dealing with that bottleneck ?

- Use a class-based language model, making **hierarchical** predictions:
  $\rightarrow$ Replace complexity in $O(|\mathcal{V}|)$ by $O(\log|\mathcal{V}|)$



From *Structured Output Layer neural network Language Model* (Hai-Son Le, Alexandre Allauzen, François Yvon, 2012)

- Use **sampling-based** methods, aiming at replacing the sum over the vocabulary by k samples when computing the gradient ($k \ll |\mathcal{V}|$):
  $\rightarrow$ **Importance Sampling**, **Noise-Contrastive Estimation**, ...
- More recently: implement a class-based softmax, based on word frequencies, and attribute more parameters to frequent words
  $\rightarrow$ **Adaptive softmax**

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)

I	ate	the	spaghetti	with	meatballs	.
Pro	V	Det	N	Prep	N	PUN

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)
  - **Chunking**

[NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs]

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)
  - **Chunking**
  - **Named-entity Recognition** (NER)

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)
  - **Chunking**
  - **Named-entity Recognition** (NER)
  - **Semantic-role labeling** (SRL)

(Agent Patient Source Destination Instrument)

John	drove	Mary	from	Austin	to	Dallas	in	his	DS Citroën
A		P		S		D			I

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)
  - **Chunking**
  - **Named-entity Recognition** (NER)
  - **Semantic-role labeling** (SRL)
- Build a model that excels on multiple benchmarks, without needing task-specific representations or engineering ?

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

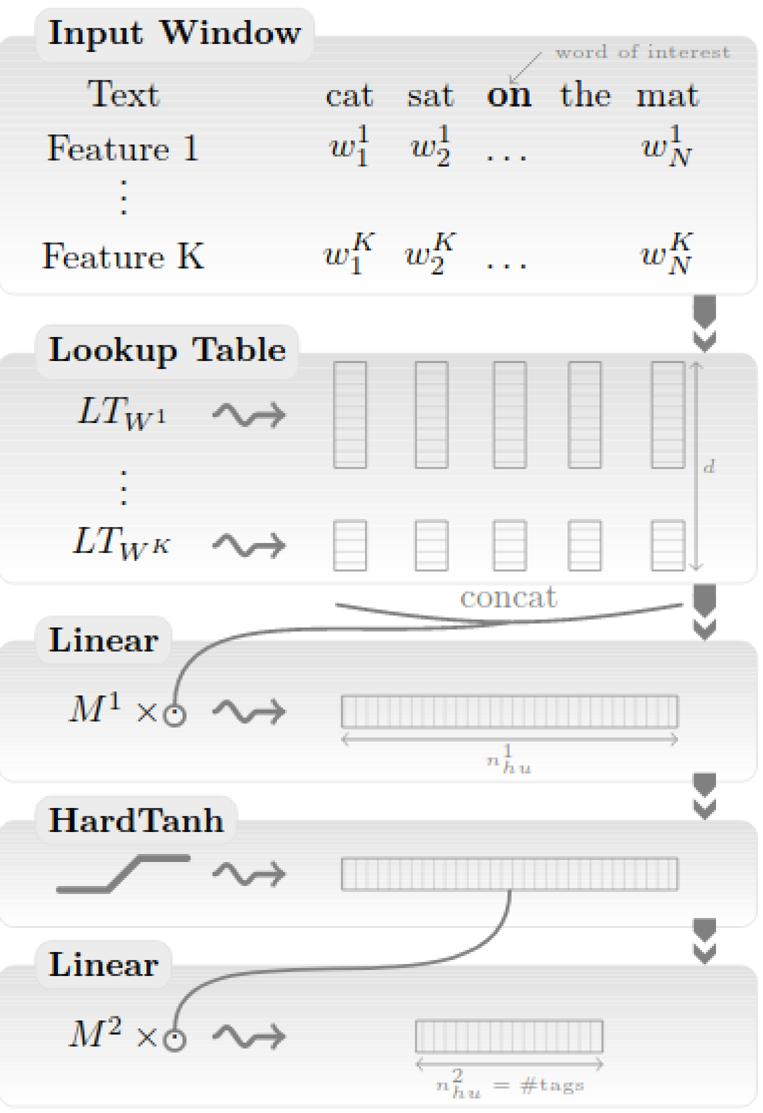
- Four standard NLP tasks, for which the state-of-the-art depend on task-specific features:
  - **Part-of-speech** tagging (POS)
  - **Chunking**
  - **Named-entity Recognition** (NER)
  - **Semantic-role labeling** (SRL)
- Build a model that excels on multiple benchmarks, without needing task-specific representations or engineering ?

$\rightarrow$ A **task-independent** approach is better because no single task can provide a complete representation of a text

Applying window-based Neural models to sequential tasks

Natural language processing (almost) from scratch (Collobert et al, 2011)

- First approach: **window-based**
- Language-modeling like training, **to rank unlabelled sentences**, brings results above the state-of-the-art for the three first tasks
- However, for SRL, the correct tag for a word may easily depend on a word outside of the current window



Towards RNN Language Models

With n-gram neural language model, *sparsity* is not an issue anymore, but we still can not deal with **long distance dependancies** between words:

- While we can increase the size of the input window, it is not practical (and actually stops improving results quite fast)
- Input words are processed differently depending on their position

$\rightarrow$ We need an architecture that can process **inputs of any lengths**, using the **same weights** for every input word.

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

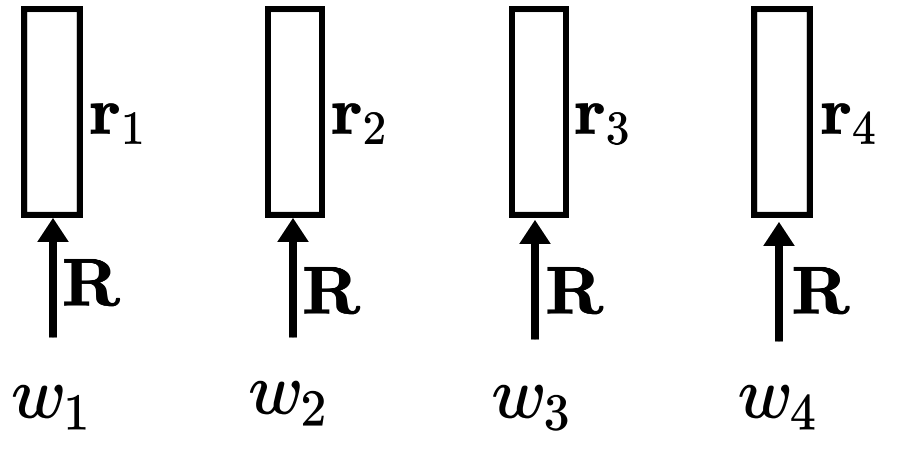
RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

w_1 w_2 w_3 w_4

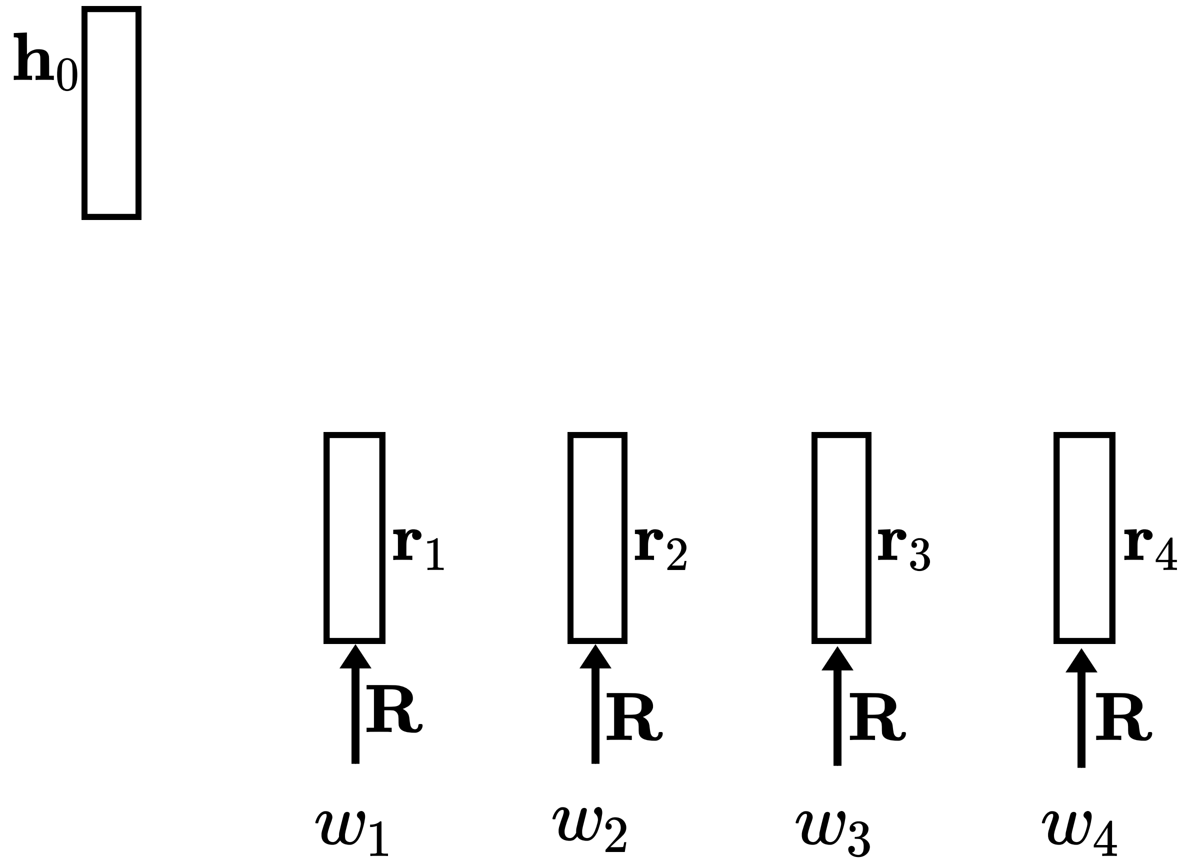
RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)



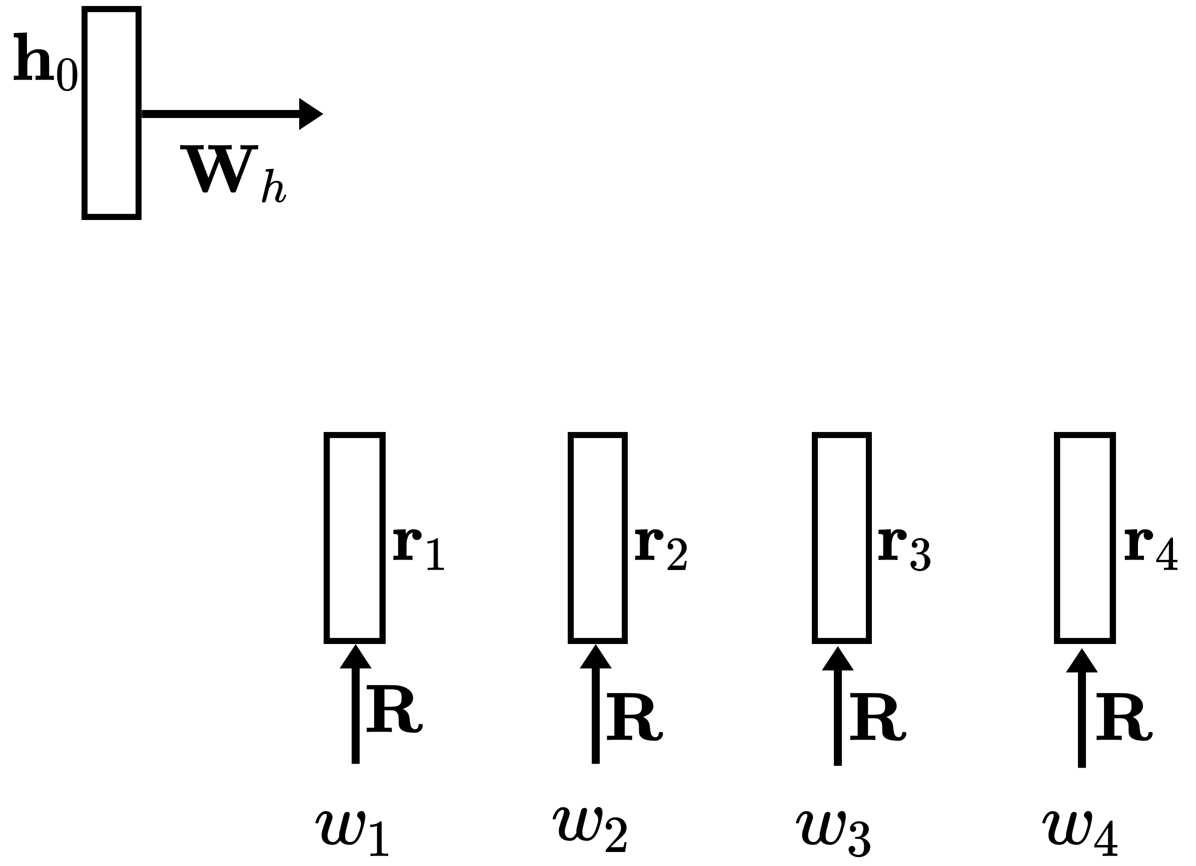
RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)



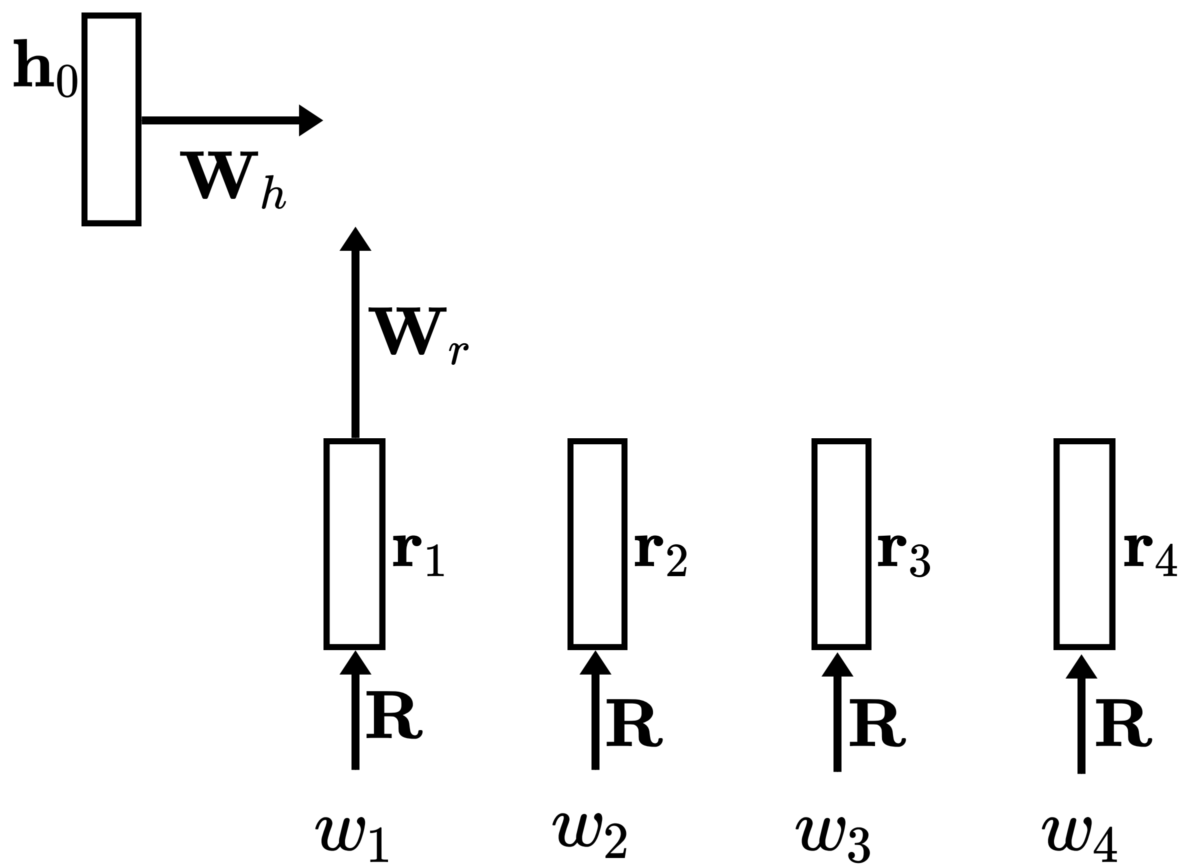
RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)



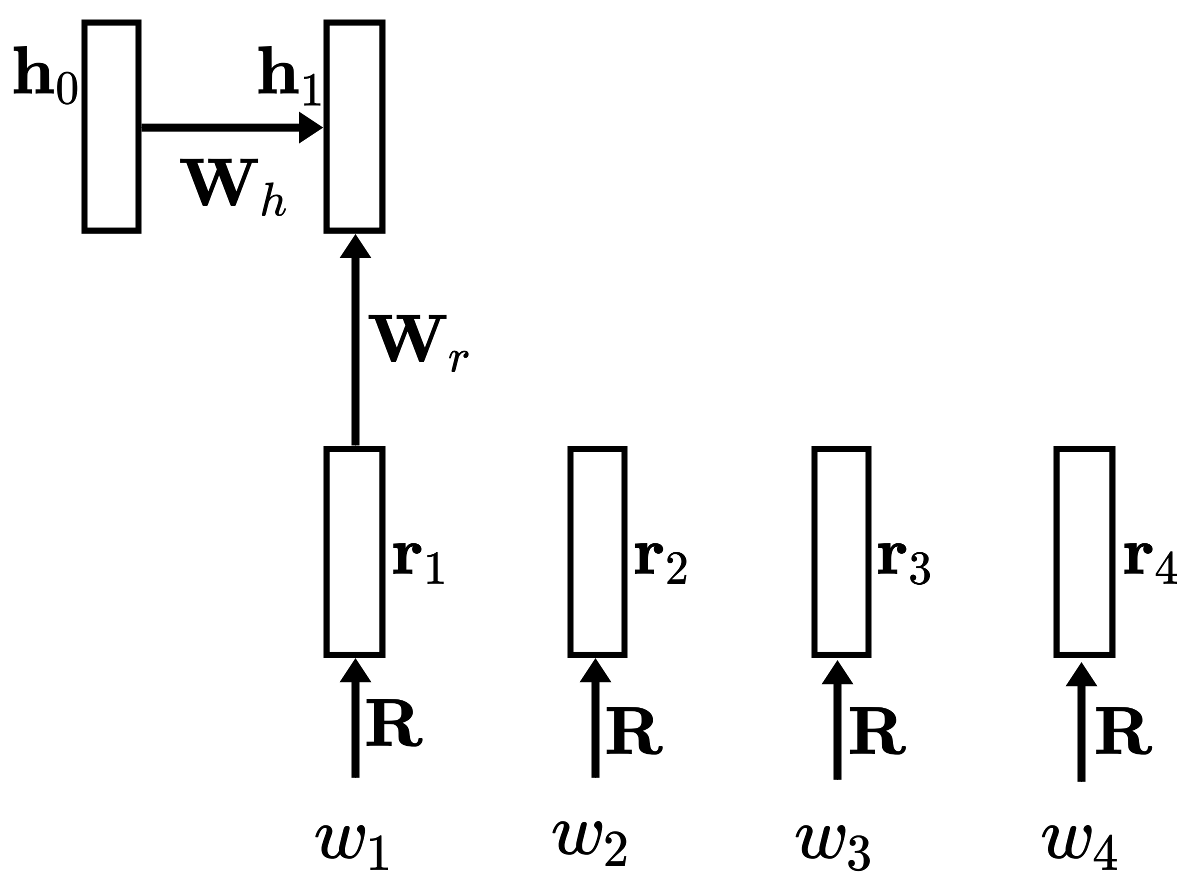
RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)



RNN Language Models

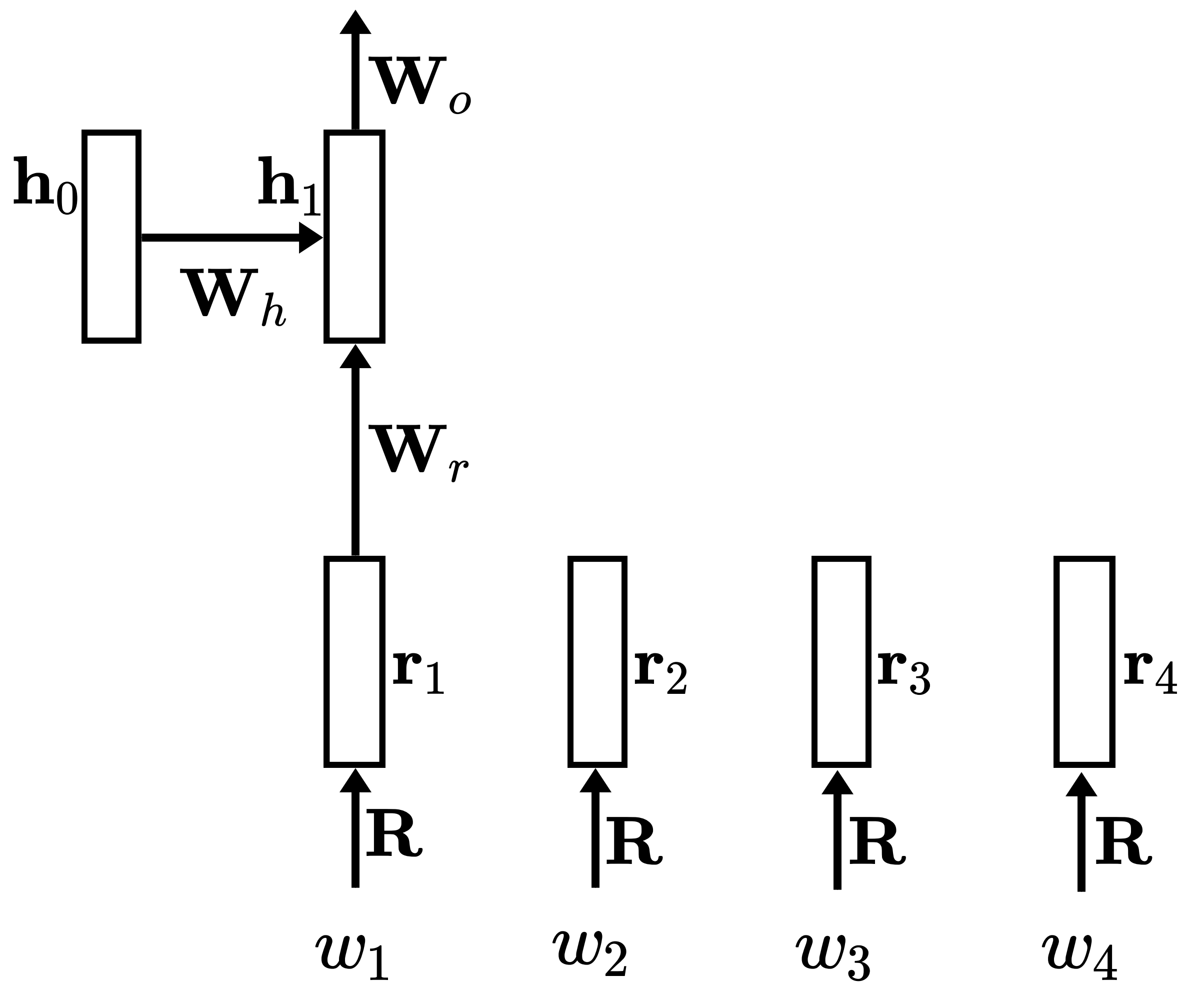
Extensions of recurrent neural network language model, (Mikolov et al, 2011)



$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h}_{\mathbf{i-1}})$$

RNN Language Models

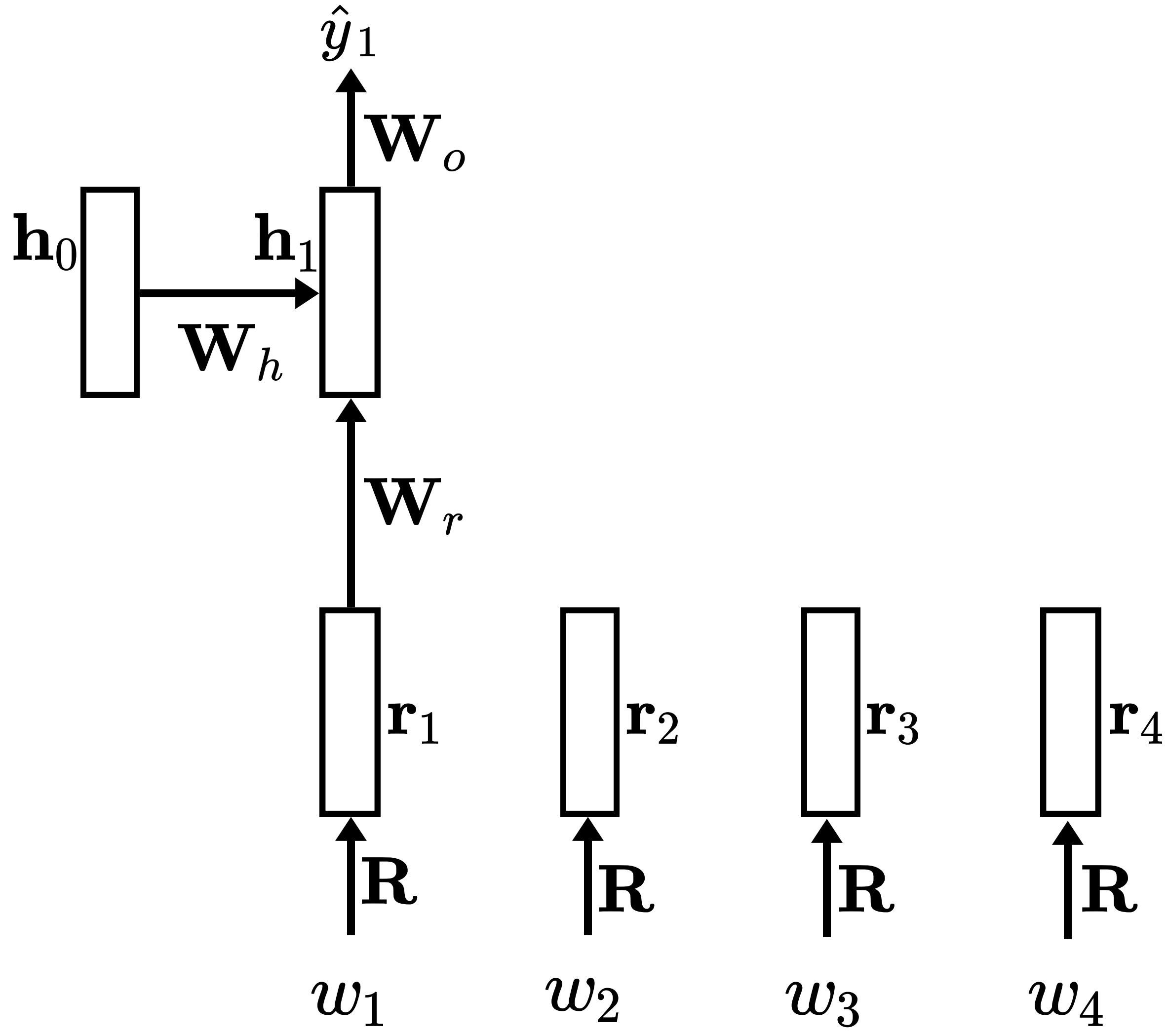
Extensions of recurrent neural network language model, (Mikolov et al, 2011)



$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h}_{\mathbf{i-1}})$$

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

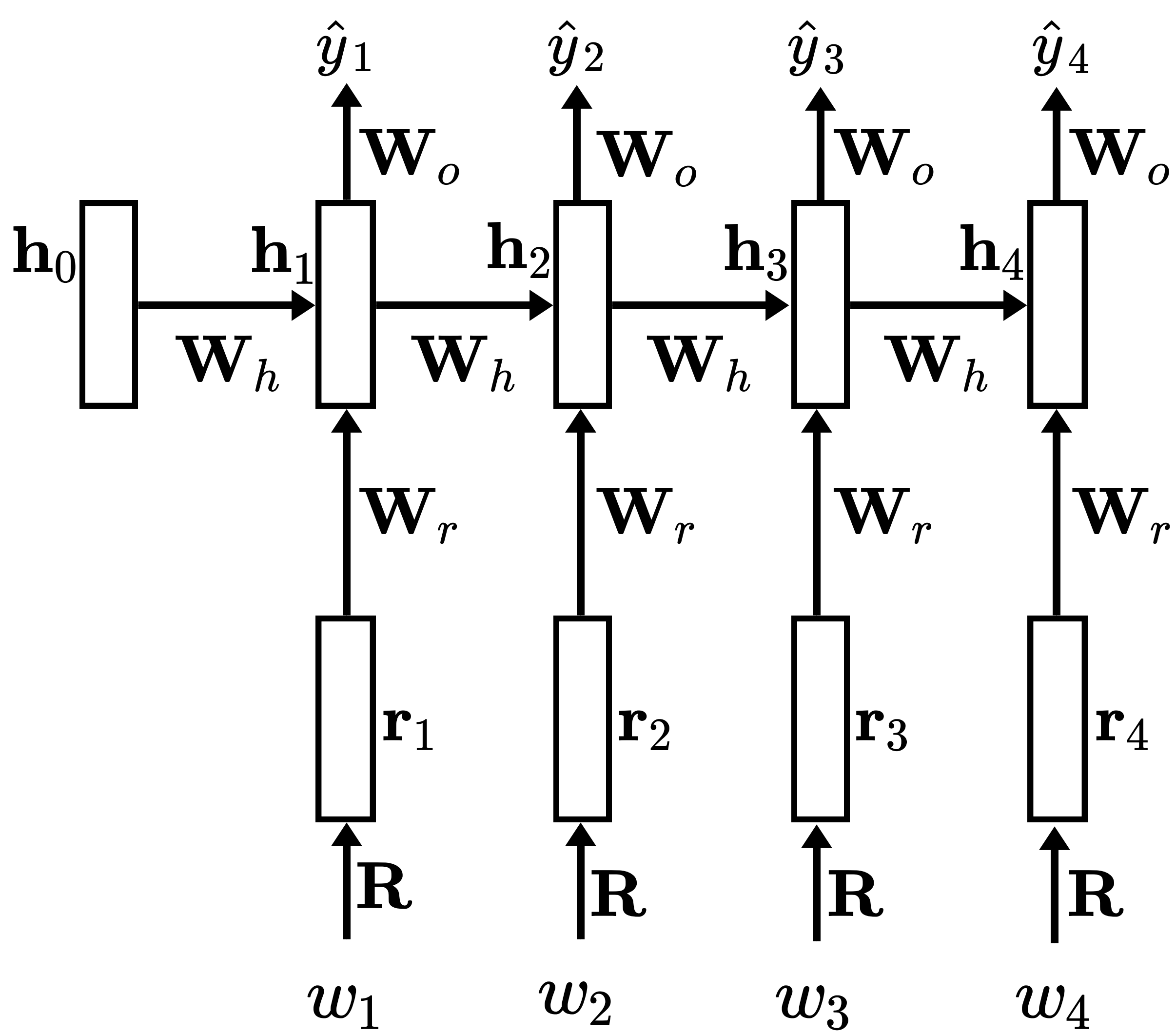


$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h_{i-1}})$$

$$\hat{y}_i = f(\mathbf{W}_o \mathbf{h_i})$$

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

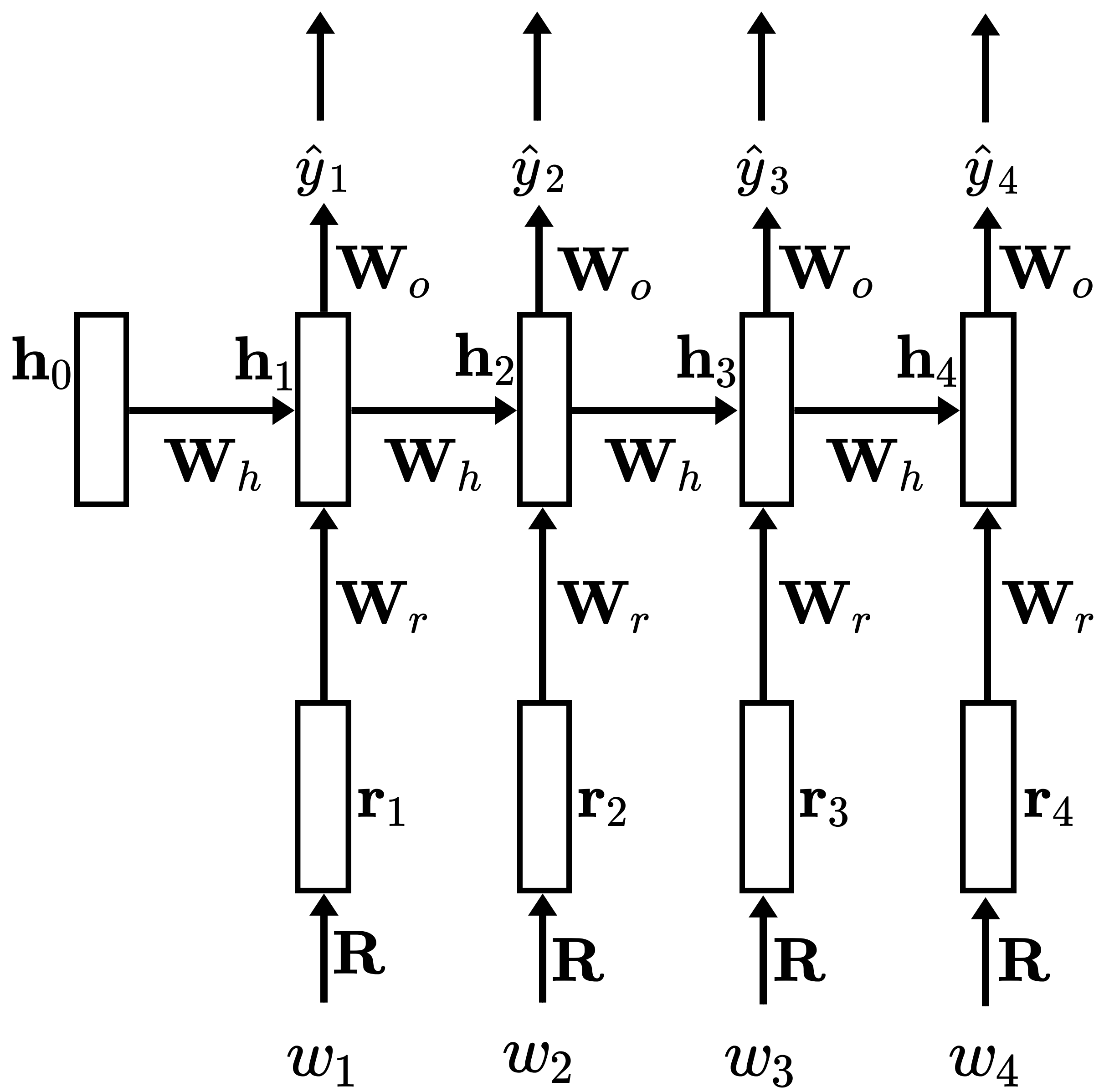


$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h_{i-1}})$$

$$\hat{y}_i = f(\mathbf{W}_o \mathbf{h_i})$$

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

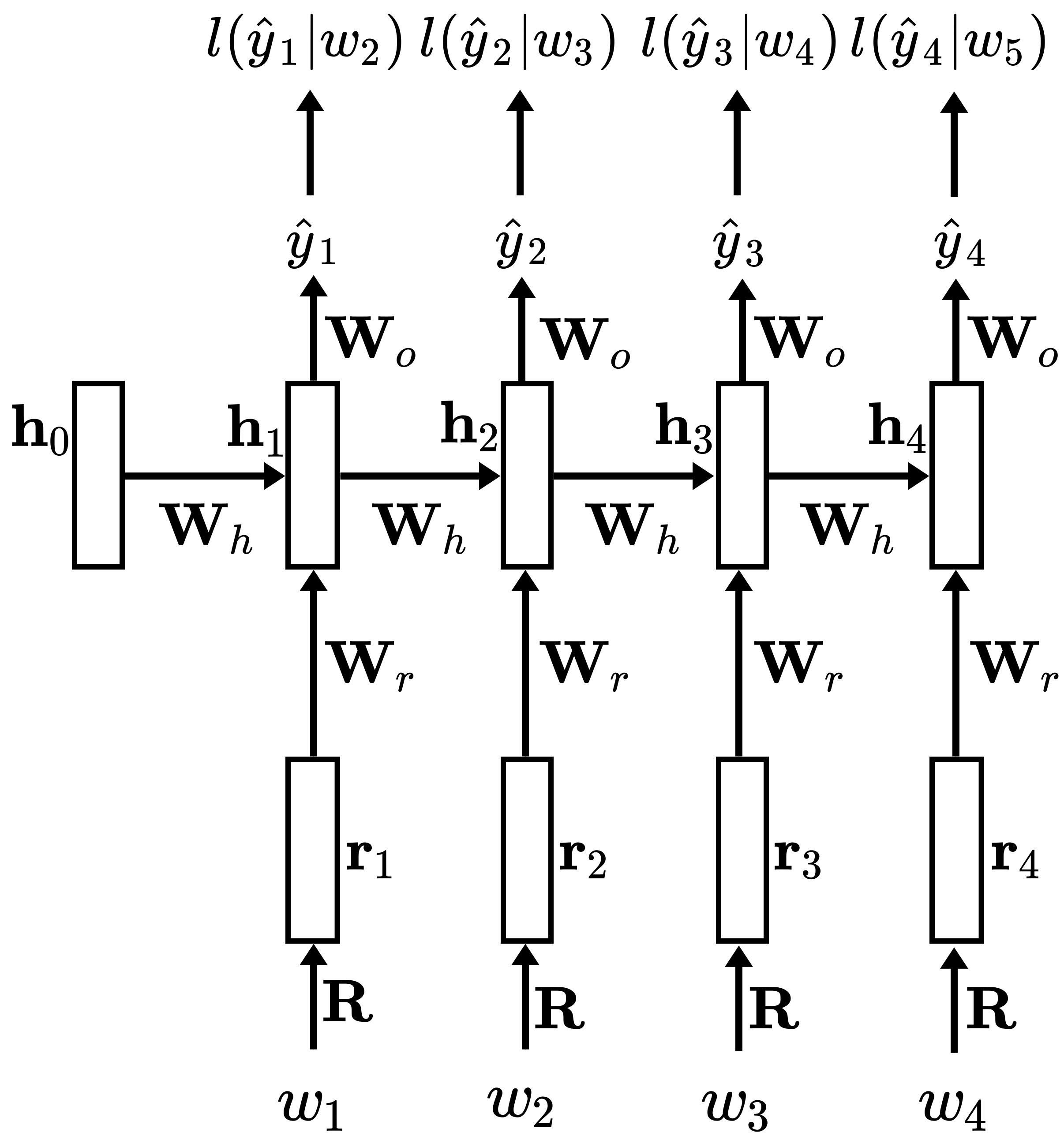


$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h_{i-1}})$$

$$\hat{y}_i = f(\mathbf{W}_o \mathbf{h_i})$$

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)

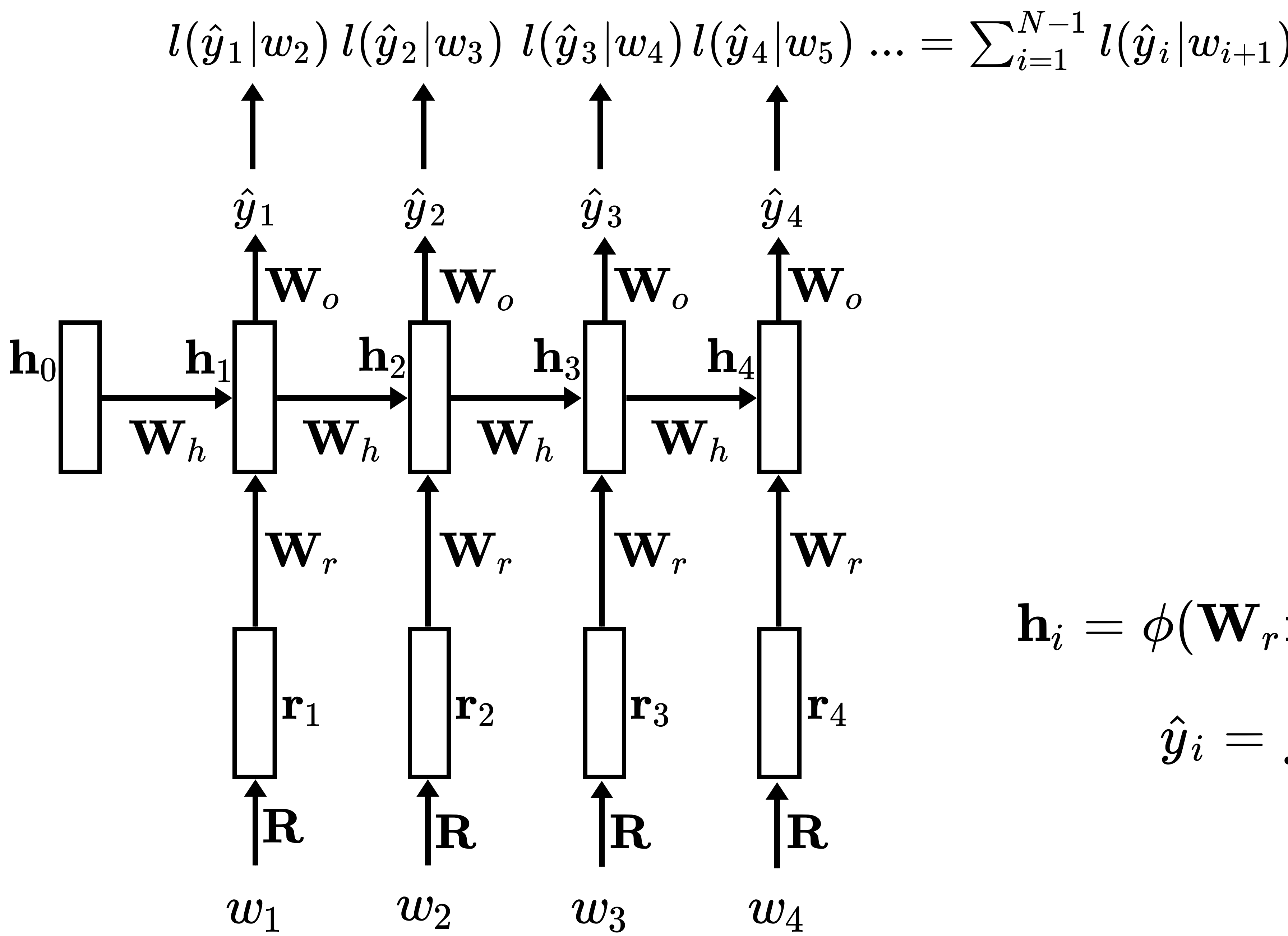


$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h}_{\mathbf{i-1}})$$

$$\hat{y}_i = f(\mathbf{W}_o \mathbf{h_i})$$

RNN Language Models

Extensions of recurrent neural network language model, (Mikolov et al, 2011)



$$\mathbf{h}_i = \phi(\mathbf{W}_r \mathbf{r}_{\mathbf{w_i}} + \mathbf{W}_h \mathbf{h}_{\mathbf{i-1}})$$

$$\hat{y}_i = f(\mathbf{W}_o \mathbf{h}_{\mathbf{i}})$$

Training RNN Language Models

- Same objective: minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^N$
- However, this time, updates go back to previous examples via reccurent links: we use **Backpropagation Through Time**

Training RNN Language Models

- Same objective: minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$
- However, this time, updates go back to previous examples via reccurent links: we use **Backpropagation Through Time**

Main issues:

- **Vanishing/exploding** gradient:
  - Architectures dedicated to this issue: **LSTM**, **GRU**

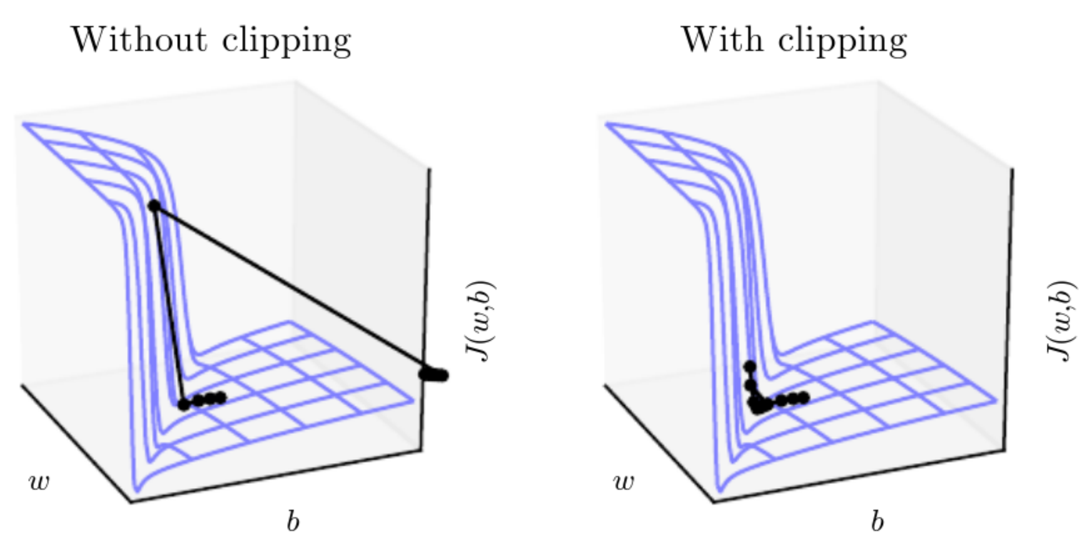
Training RNN Language Models

- Same objective: minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$
- However, this time, updates go back to previous examples via reccurent links: we use **Backpropagation Through Time**

Main issues:

- **Vanishing/exploding** gradient:
  - Architectures dedicated to this issue: **LSTM**, **GRU**
  - Gradient clipping to a constant γ

From *Deep Learning* (Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016)



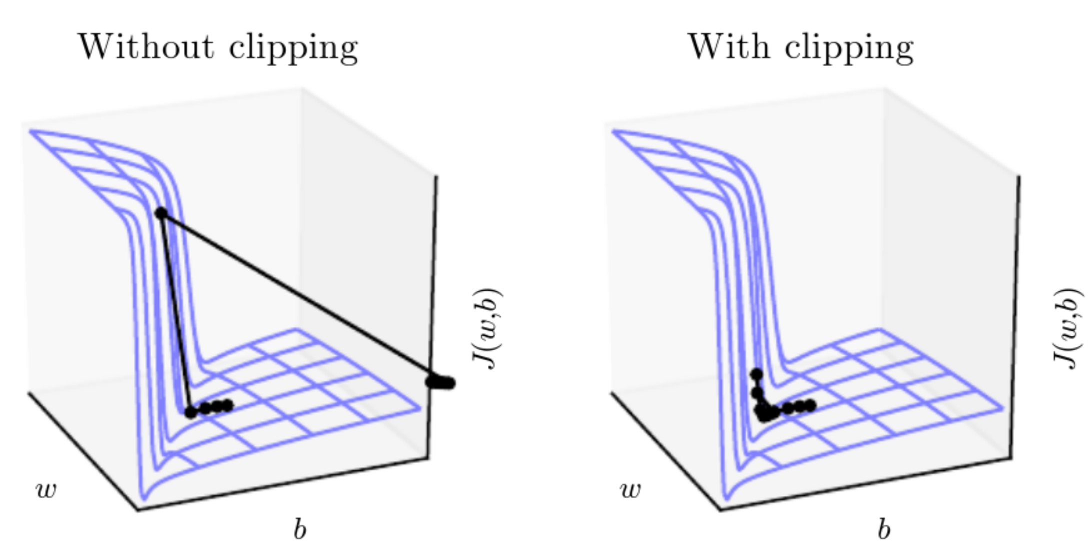
Training RNN Language Models

- Same objective: minimizing the **negative log-likelihood** on $\mathcal{D} = (w_i)_{i=1}^{N}$
- However, this time, updates go back to previous examples via reccurent links: we use **Backpropagation Through Time**

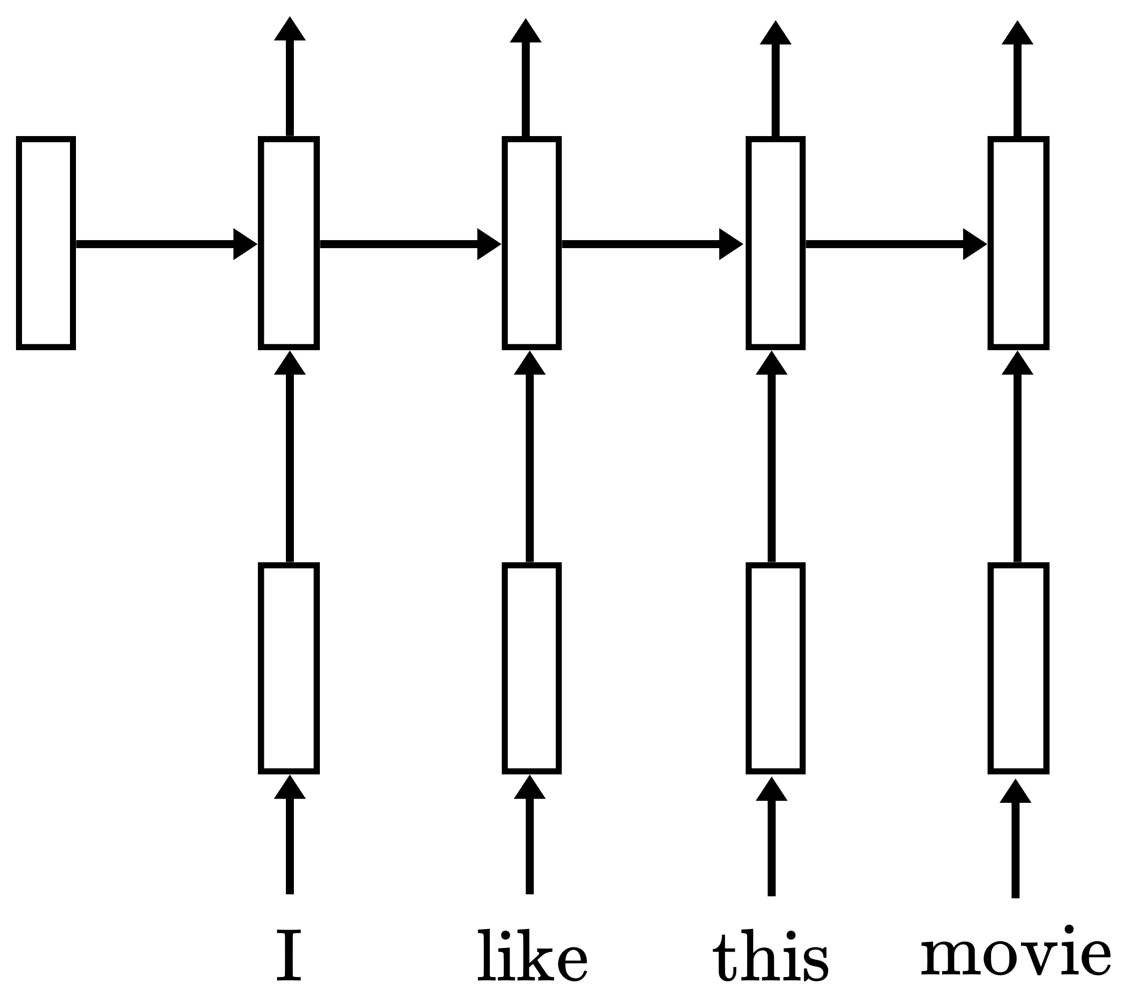
Main issues:

- **Vanishing/exploding** gradient:
  - Architectures dedicated to this issue: **LSTM**, **GRU**
  - Gradient clipping to a constant γ
- **Long term memory**:
  $\rightarrow$ bidirectional architectures !

From *Deep Learning* (Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016)

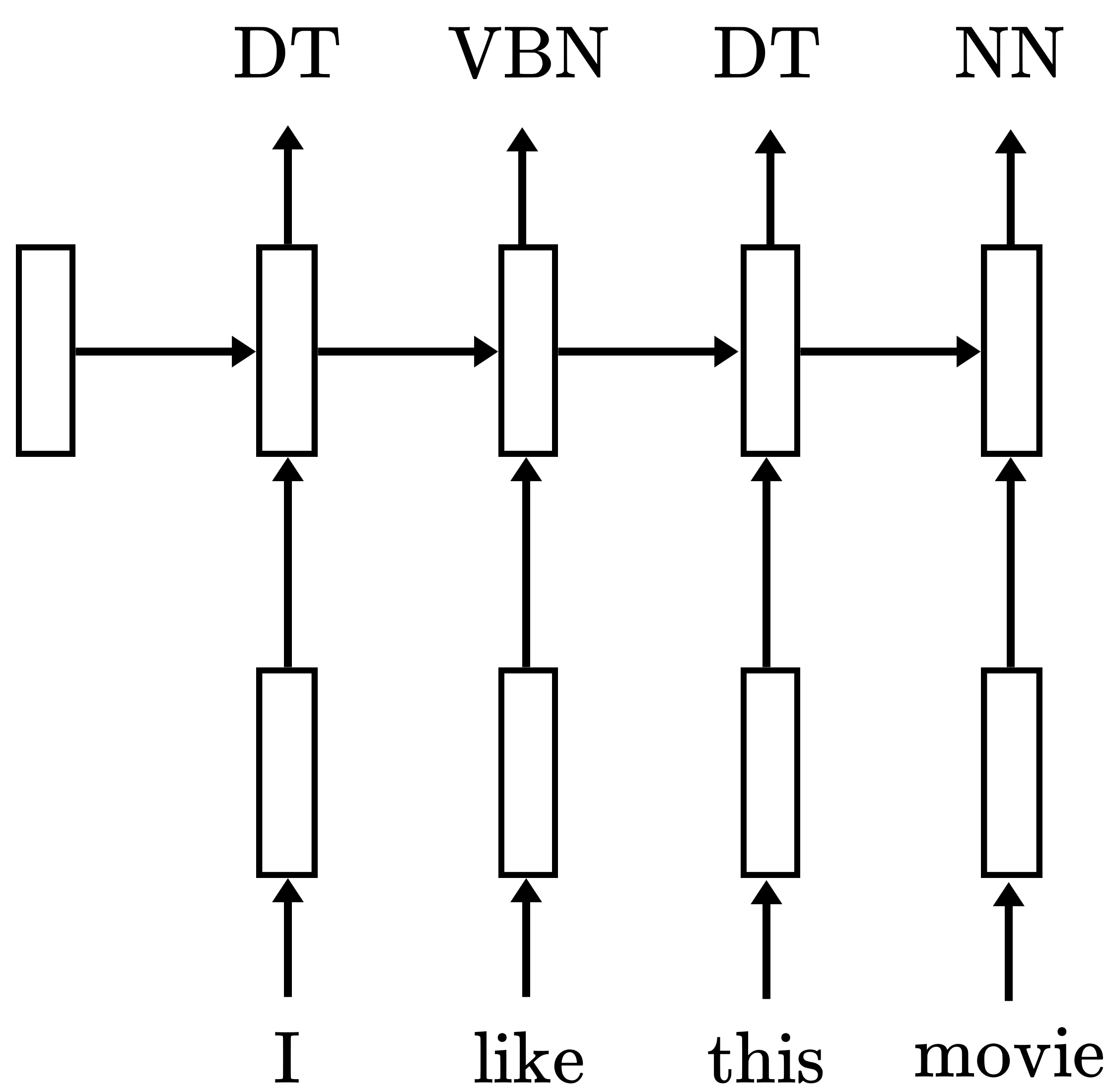


RNN: Other sequential tasks



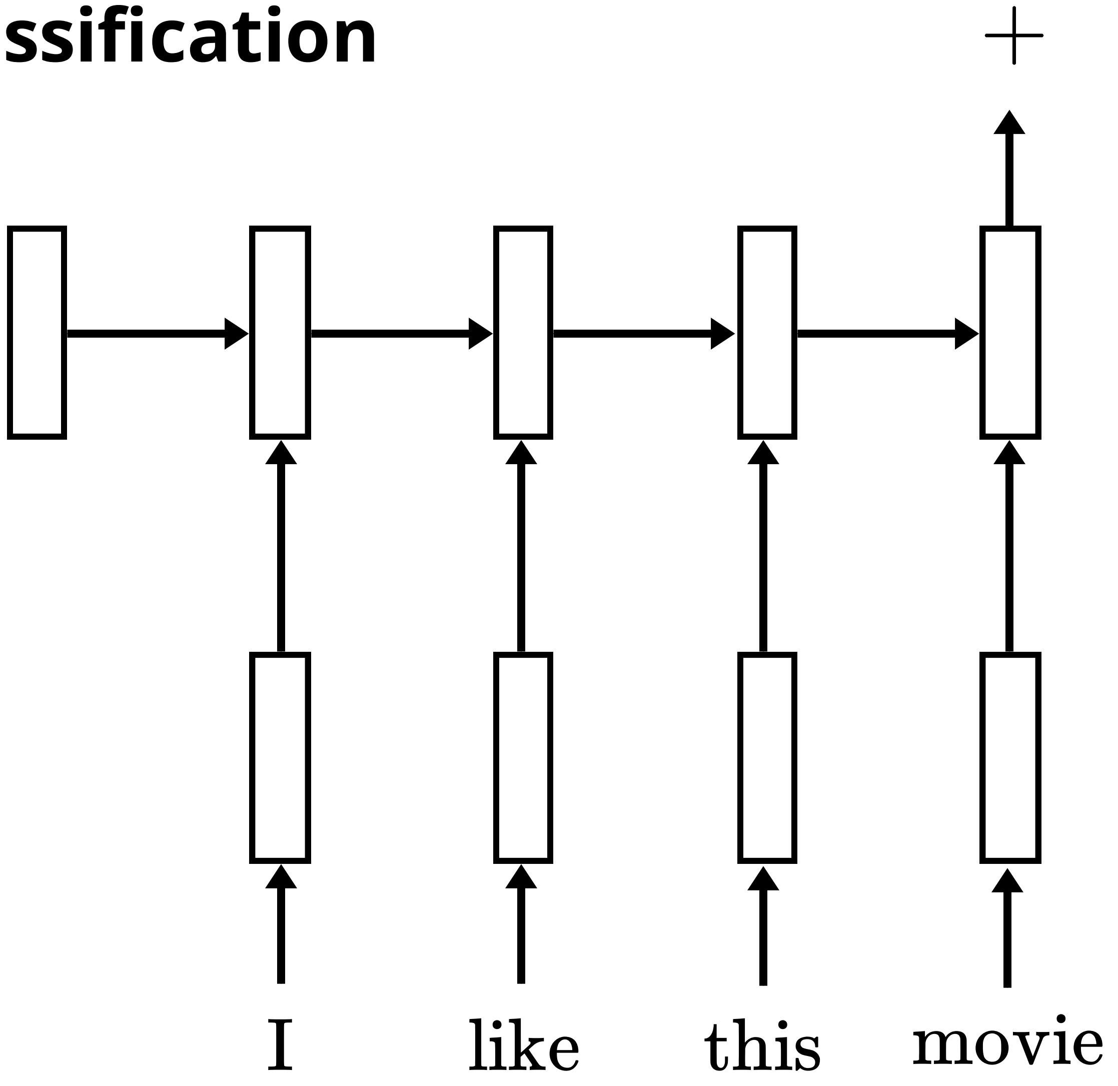
RNN: Other sequential tasks

POS Tagging



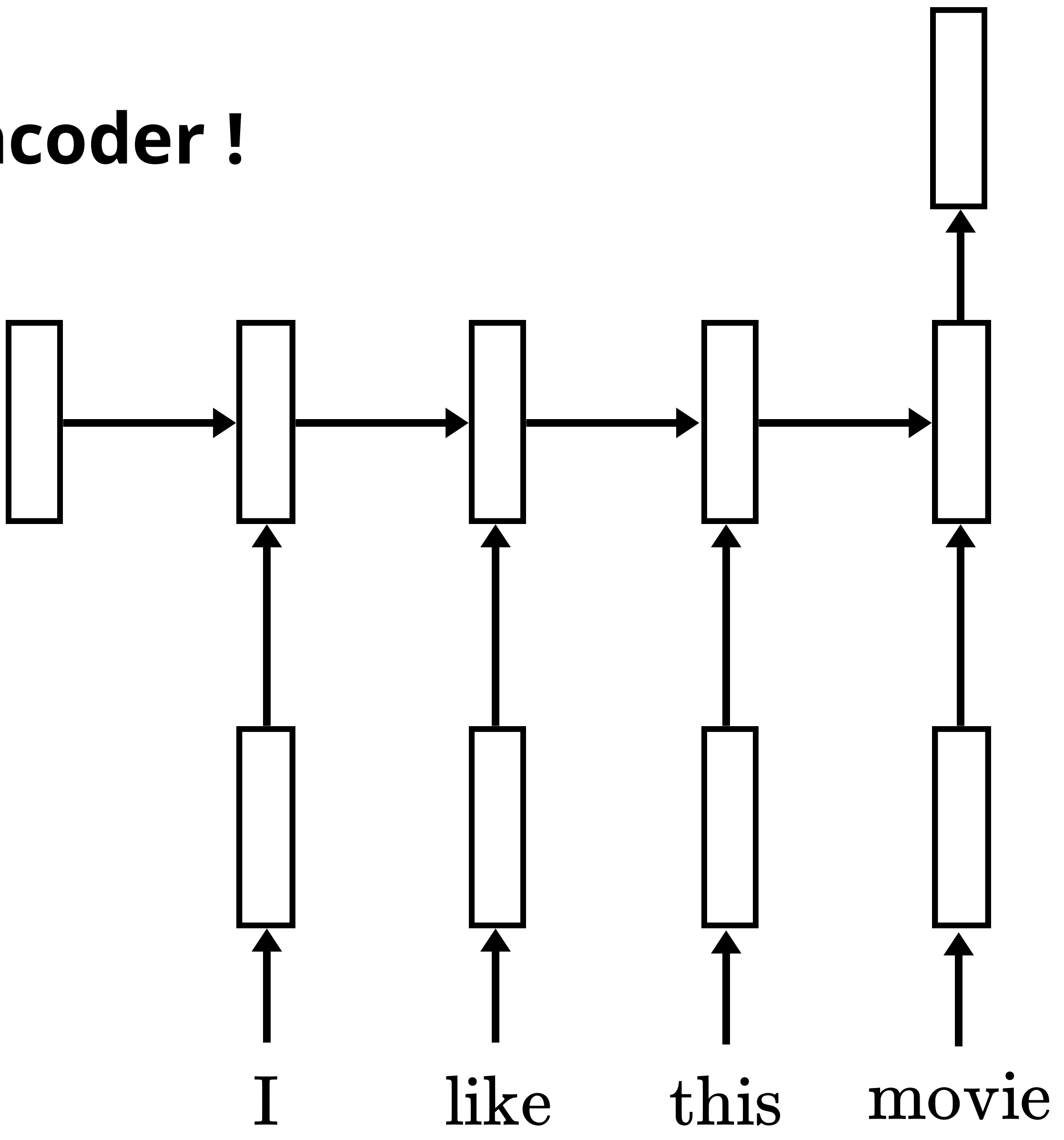
RNN: Other sequential tasks

Sentiment Classification



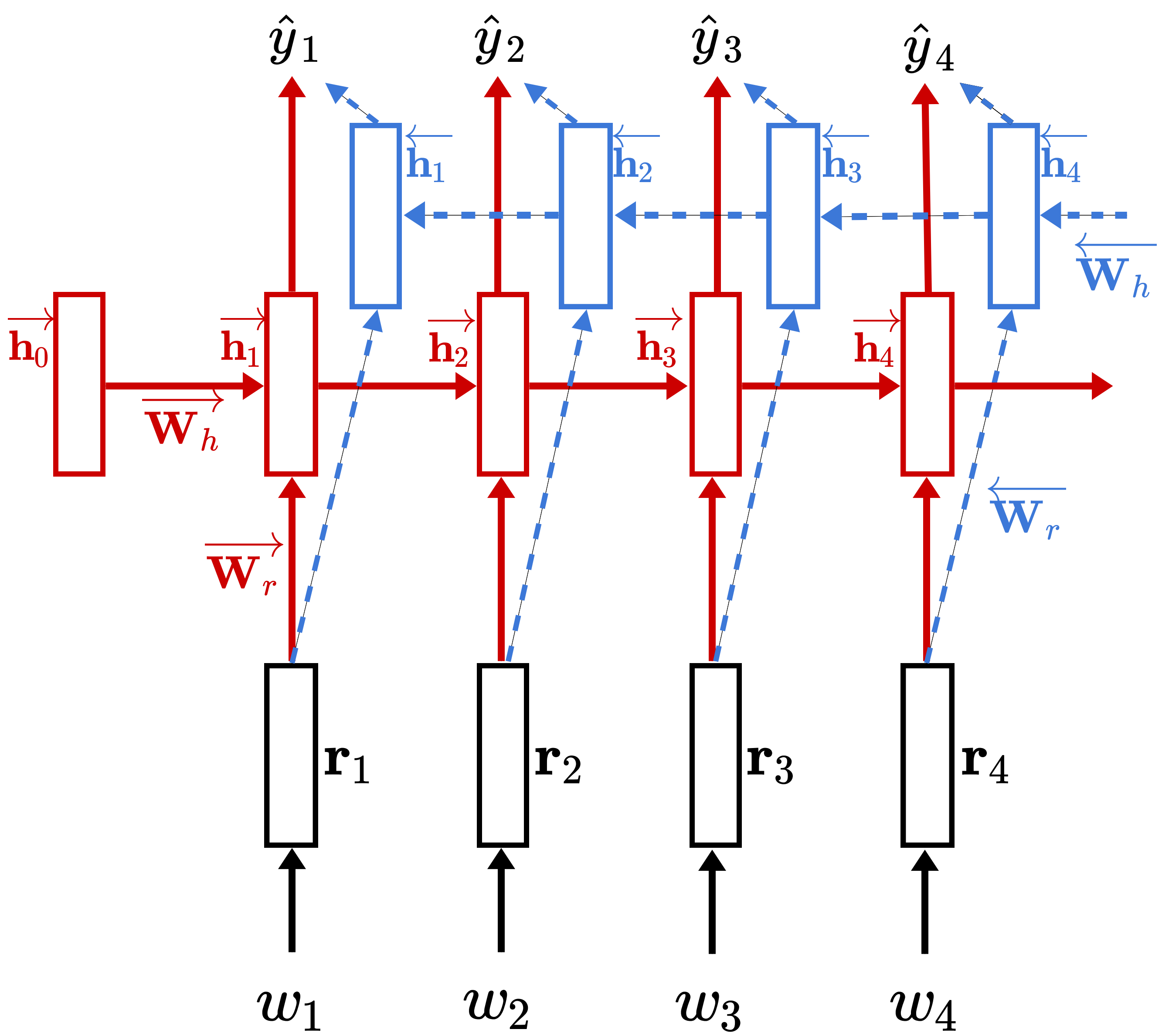
RNN: Other sequential tasks

Any kind of encoder !



Bidirectional Architectures

Bidirectional LSTMnetworks for improved phoneme classification and recognition (Graves et al, 2005)



$$\overrightarrow{\mathbf{h}_i} = \phi(\overrightarrow{\mathbf{W}_r}\mathbf{r}_{\mathbf{w_i}} + \overrightarrow{\mathbf{W}_h}\overrightarrow{\mathbf{h}_{\mathbf{i-1}}})$$

$$\overleftarrow{\mathbf{h}_i} = \phi(\overleftarrow{\mathbf{W}_r}\mathbf{r}_{\mathbf{w_i}} + \overleftarrow{\mathbf{W}_h}\overleftarrow{\mathbf{h}_{i+1}})$$

$$\hat{y}_i = f(\mathbf{W}_o[\overrightarrow{\mathbf{h}}_i : \overleftarrow{\mathbf{h}}_i])$$

- $[\overrightarrow{\mathbf{h}}_i : \overleftarrow{\mathbf{h}}_i]$ represents w_i using both contexts !
- Powerful, but you need the **entire input sequence** $\rightarrow$ It can not be used for language modeling…

Before RNNs: Sentence-based approach

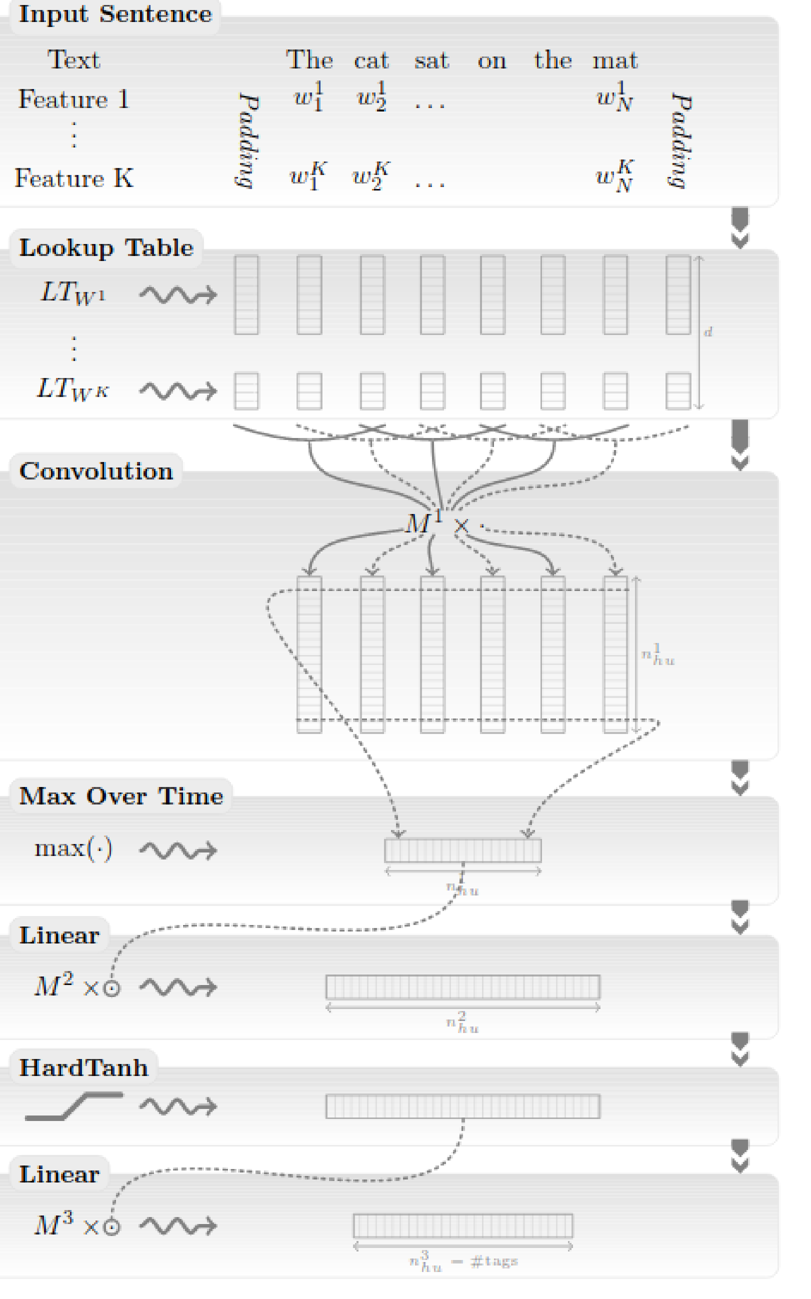
Natural language processing (almost) from scratch (Collobert et al, 2011)

- Remember: the **window-based** approach was not great for long dependancies

Before RNNs: Sentence-based approach

Natural language processing (almost) from scratch (Collobert et al, 2011)

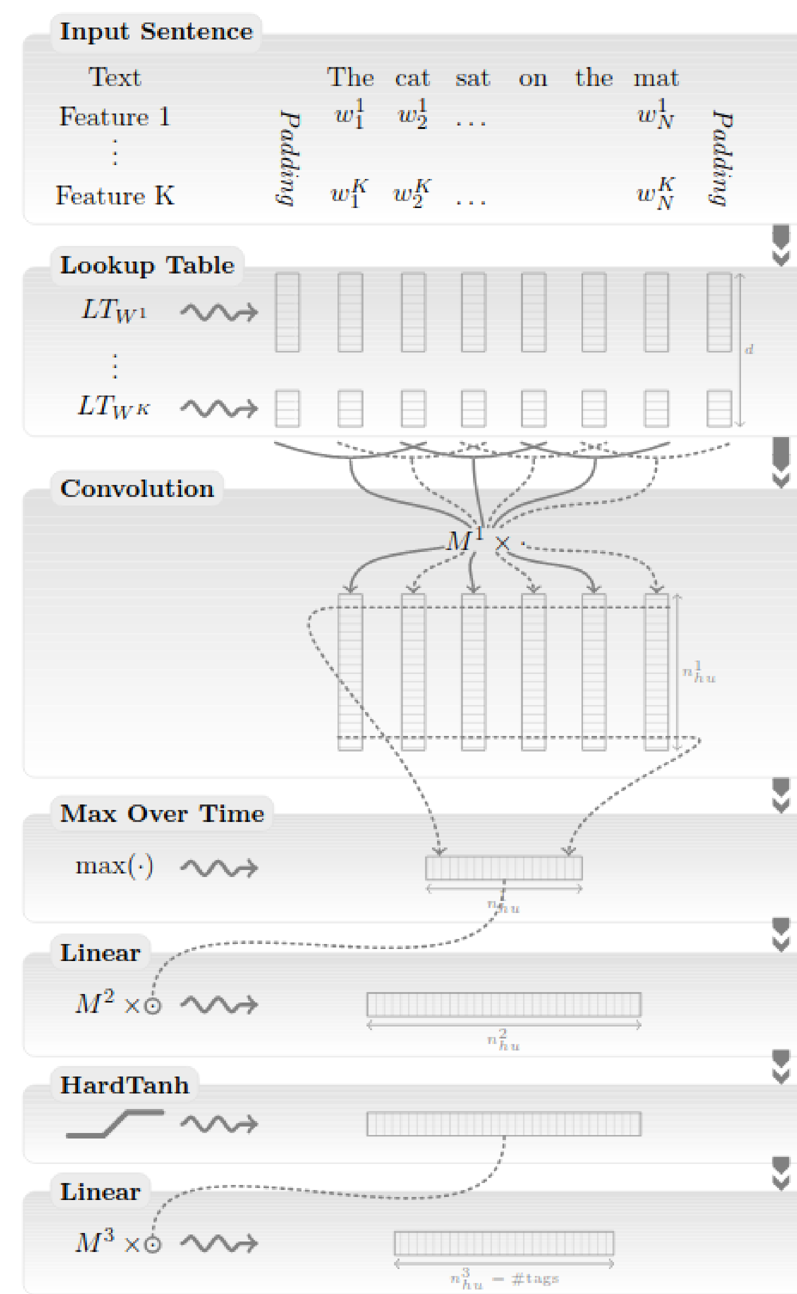
- Remember: the **window-based** approach was not great for long dependancies
- Second approach - Take the whole sentence as input: use a **convolutional approach !**



Before RNNs: Sentence-based approach

Natural language processing (almost) from scratch (Collobert et al, 2011)

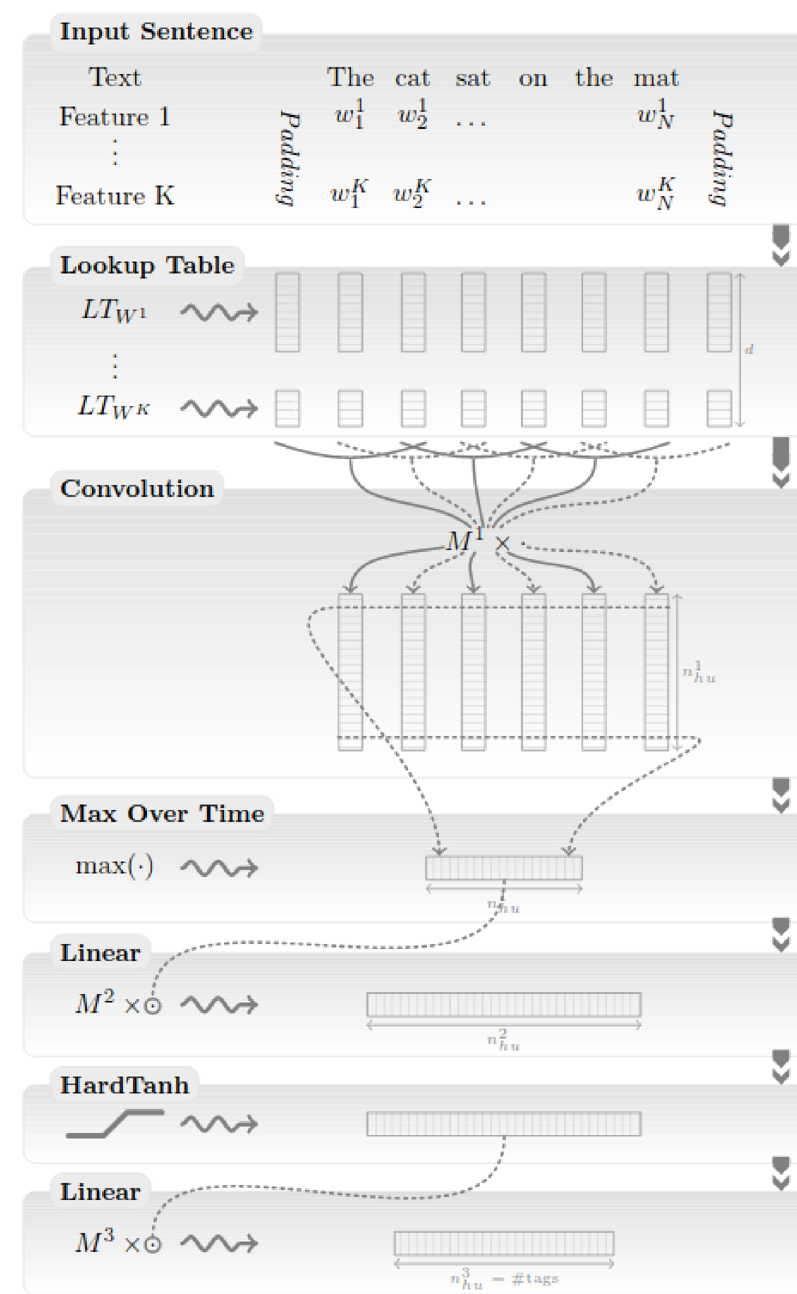
- Remember: the **window-based** approach was not great for long dependancies
- Second approach - Take the whole sentence as input: use a **convolutional approach !**
- Combines local features into a **global feature vector**



Before RNNs: Sentence-based approach

Natural language processing (almost) from scratch (Collobert et al, 2011)

- Remember: the **window-based** approach was not great for long dependancies
- Second approach - Take the whole sentence as input: use a **convolutional approach !**
- Combines local features into a **global feature vector**
- A **structured training objective** (taking in accounts dependancies between predicted tags) helps improving results



Convolutional instead of reccurent ?

- With RNNs making a prediction for a full sequence, the **last words** have way too much importance in the final hidden representation !

Convolutional instead of reccurent ?

- With RNNs making a prediction for a full sequence, the **last words** have way too much importance in the final hidden representation !
- What about coming back to a **window-based approach** ?
  - Compute representations for each window (= possible subsequence of a certain length) $\rightarrow$ **convolution**
  - Group them together into a global representation $\rightarrow$ **pooling**

Convolutional instead of reccurent ?

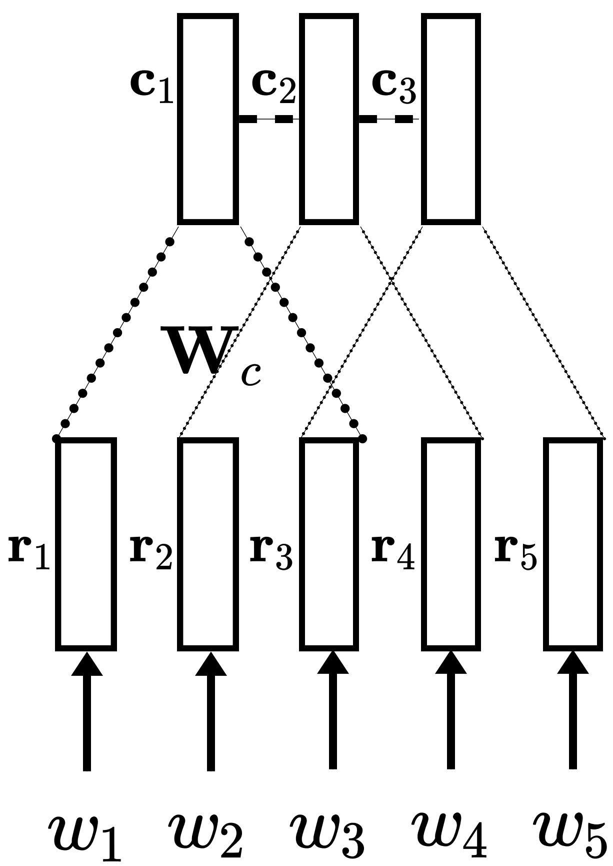
- With RNNs making a prediction for a full sequence, the **last words** have way too much importance in the final hidden representation !
- What about coming back to a **window-based approach** ?
  - Compute representations for each window (= possible subsequence of a certain length) $\rightarrow$ **convolution**
  - Group them together into a global representation $\rightarrow$ **pooling**

$$\text{Group them together into a global representation}$$

$$\downarrow$$

$$\overbrace{\text{Group them together}}, \overbrace{\text{them together into}}, \ldots, \overbrace{\text{a global representation}}$$

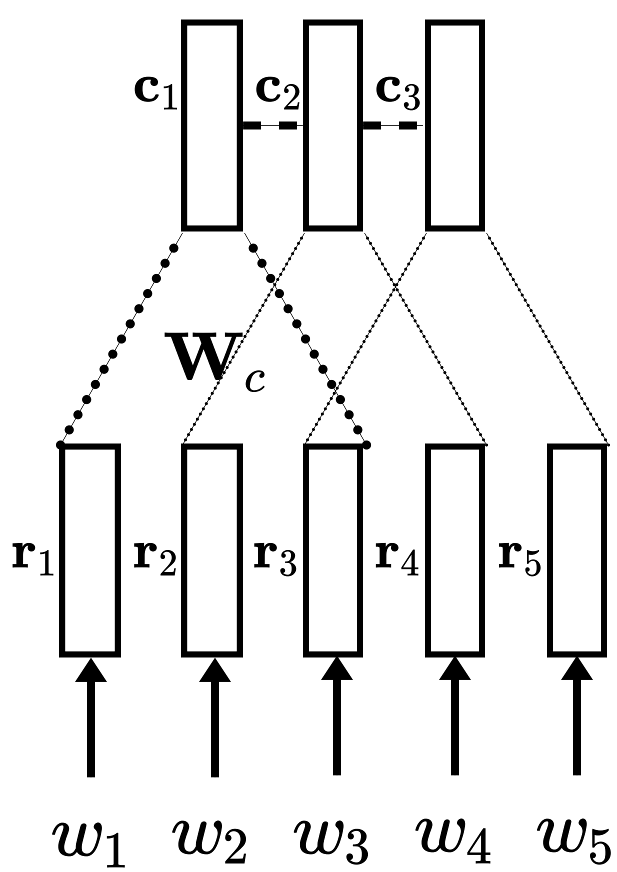
Convolutional networks for NLP



- Inputs in the same window are concatenated:

$$\mathbf{r}_{i:i+t} = [\mathbf{r}_i : .. : \mathbf{r}_t]$$

Convolutional networks for NLP



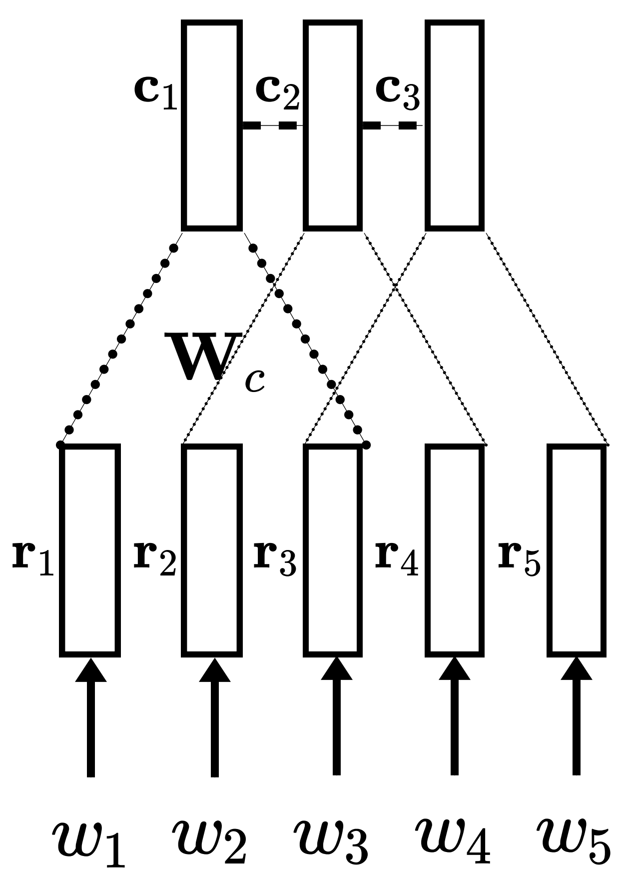
- Inputs in the same window are concatenated:

$$\mathbf{r}_{i:i+t} = [\mathbf{r}_i : .. : \mathbf{r}_t]$$

- A filter is a vector $\mathbf{w}$ which is applied to the window to obtain an individual feature:

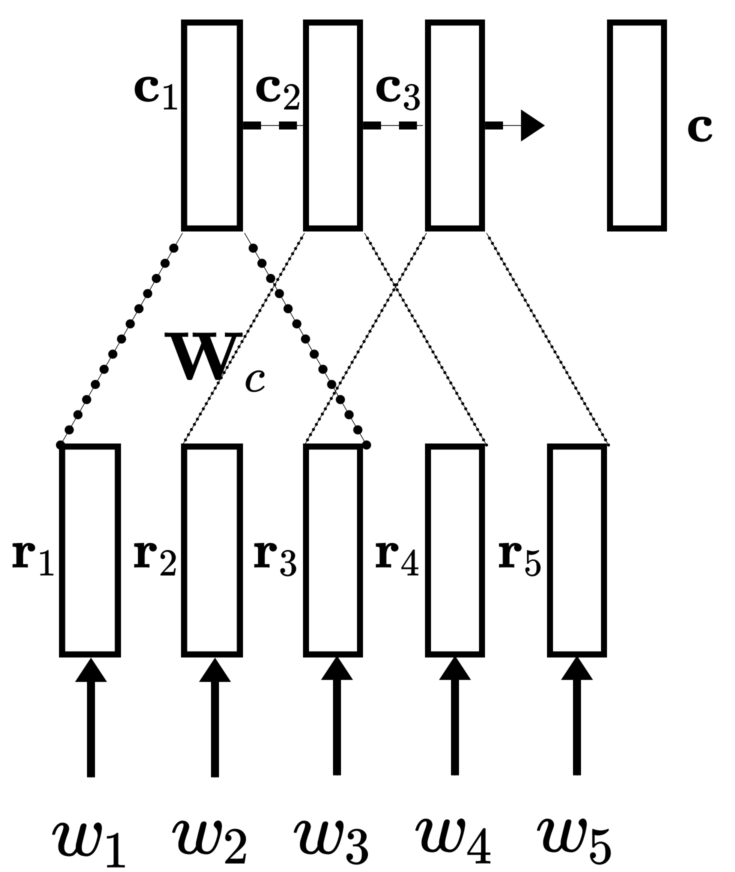
$$c_i^j = \phi(\mathbf{w}_j^T \mathbf{r}_{i:i+t})$$

Convolutional networks for NLP



- Inputs in the same window are concatenated:
$$\mathbf{r}_{i:i+t} = [\mathbf{r}_i : .. : \mathbf{r}_t]$$
- A filter is a vector $\mathbf{w}$ which is applied to the window to obtain an individual feature:
$$c_i^j = \phi(\mathbf{w}_j^T \mathbf{r}_{i:i+t})$$
- The weight matrix $\mathbf{W}_c$ re-groups the d_c filters, outputting a feature vector of size d_c:
$$\mathbf{c}_i = \phi(\mathbf{W}_c^T \mathbf{r}_{i:i+t})$$

Convolutional networks for NLP

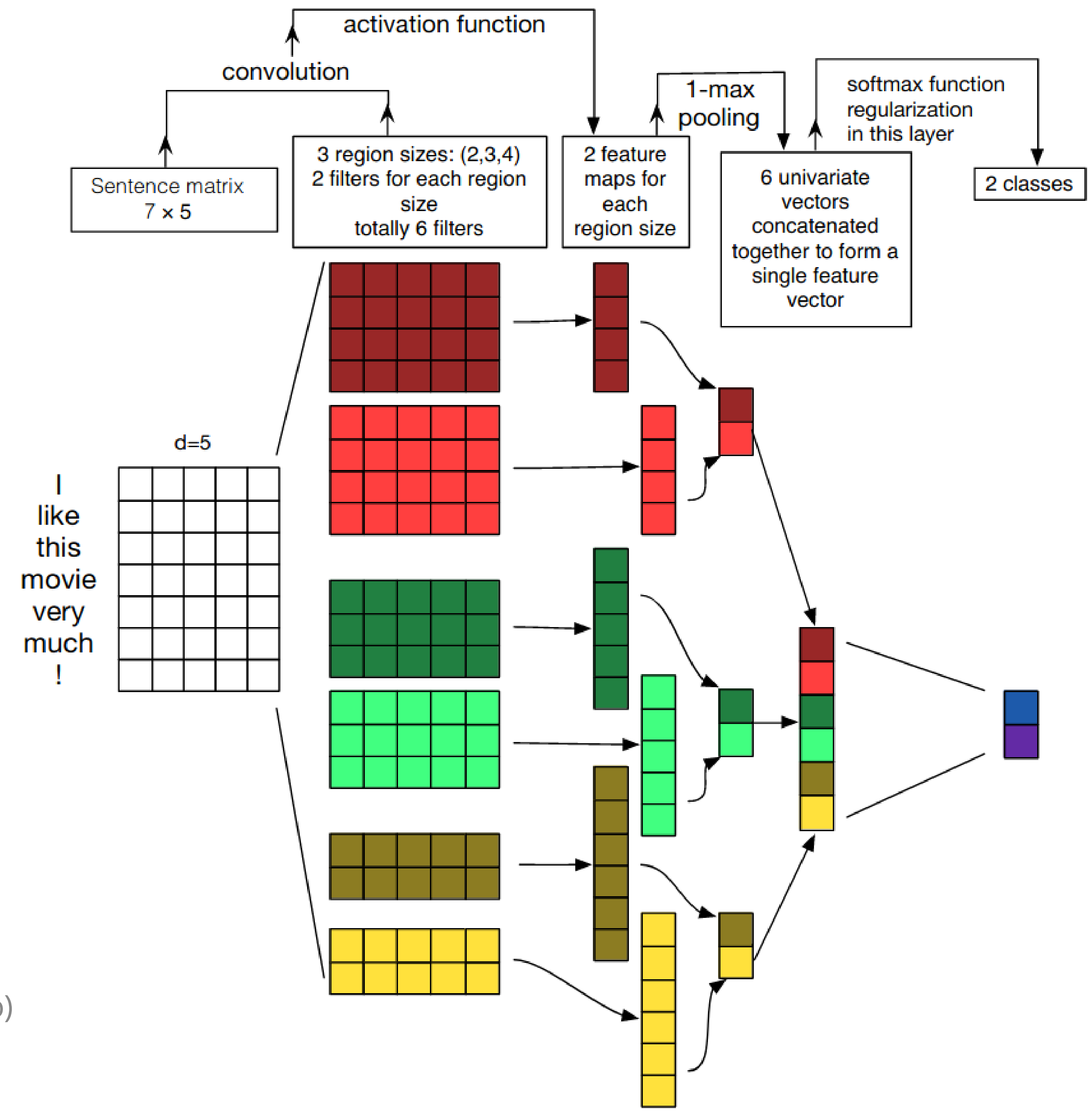


- Inputs in the same window are concatenated:
$$\mathbf{r}_{i:i+t} = [\mathbf{r}_i : .. : \mathbf{r}_t]$$
- A filter is a vector $\mathbf{w}$ which is applied to the window to obtain an individual feature:
$$c_i^j = \phi(\mathbf{w}_j^T \mathbf{r}_{i:i+t})$$
- The weight matrix $\mathbf{W}_c$ re-groups the d_c filters, outputting a feature vector of size d_c:
$$\mathbf{c}_i = \phi(\mathbf{W}_c^T \mathbf{r}_{i:i+t})$$
- With max-pooling, the idea is to capture the most important activation over time:
$$\mathbf{c} = max(\{\mathbf{c}_i\}_{i=1}^{n-t})$$

CNN for NLP: Applications

- Convolutional networks are lighter, faster - and we can apply filters on **windows of various** sizes !

- The architecture became popular for NLP tasks with *Convolutional Neural Networks for Sentence Classification* (Kim, 2014)

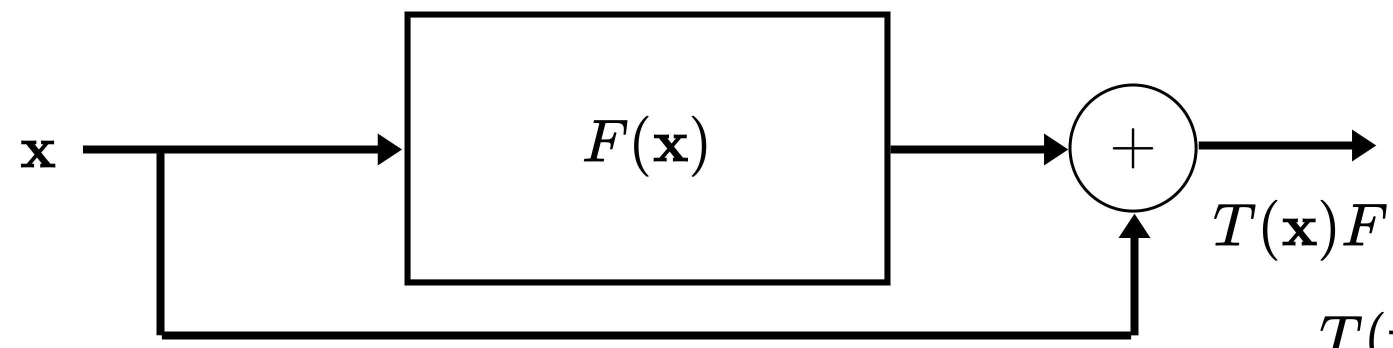


From "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification" (Ye Zhang, Byron Wallace, 2015)

Supplementary innovations used in NLP

- **Gating/Skipping**, similar to LSTM/GRU, but vertically:

  Highway Networks, (Srivastava et al, 2015)



$$T(\mathbf{x})F(\mathbf{x}) + (1 - T(\mathbf{x}))\mathbf{x}$$

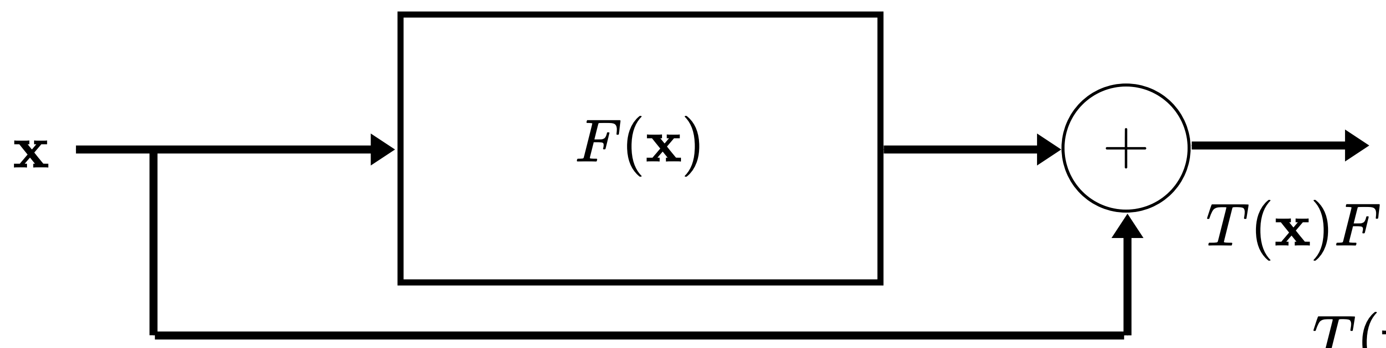
$$T(\mathbf{x}) = \phi(\mathbf{W}_T\mathbf{x})$$

$\rightarrow$ Very useful for deep networks

Supplementary innovations used in NLP

- **Gating/Skipping**, similar to LSTM/GRU, but vertically:

  Highway Networks, (Srivastava et al, 2015)



$$T(\mathbf{x})F(\mathbf{x}) + (1 - T(\mathbf{x}))\mathbf{x}$$

$$T(\mathbf{x}) = \phi(\mathbf{W}_T\mathbf{x})$$

  $\rightarrow$ Very useful for deep networks

- **Batch Normalization**: Scaling the output of the activation function to have zero mean and unit variance - *Batch normalization: Accelerating deep network training by reducing internal covariate shift* (Ioffe et al, 2015)
  - Make models less sensible to parameter initialization
  - Reduce the difficulty with learning rate tuning

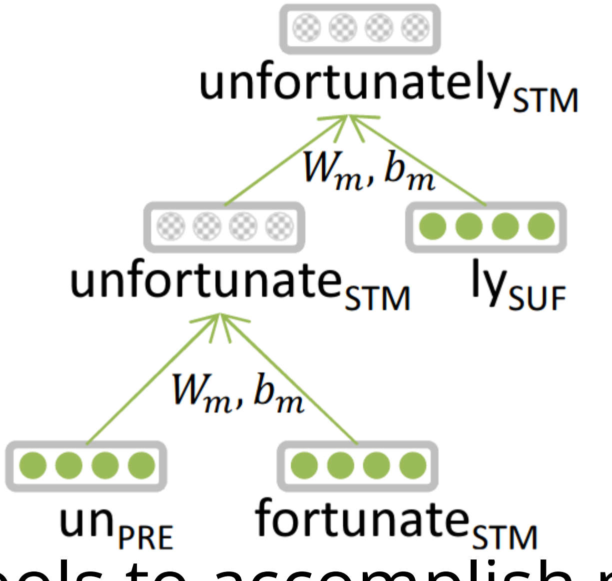
Subwords models

- Common issues with word-based models:
  - The vocabulary is **closed -** especially an issue when **spelling varies**
  - We are missing a lot of information **linking words**

Subwords models

- Common issues with word-based models:
  - The vocabulary is **closed -** especially an issue when **spelling varies**
  - We are missing a lot of information **linking words**
- Linguistically motivated decomposition:
  - **Phonemes** (Distinctive features in audio), **morphemes** (smallest semantic unit)

    Better Word Representations with Recursive Neural Networks forMorphology, (Luong et al, 2013)



$\rightarrow$ While there exist tools to accomplish morphological decomposition, it remains **costly** to obtain

Subwords models

- The goal here is to obtain better word representation **from subwords**
  $\rightarrow$ An easier alternative is to use character n-grams !

$$\text{Unfortunately}$$

$$\downarrow$$

$$\overbrace{\text{Unf}}, \overbrace{\text{nfo}}, \ldots, \overbrace{\text{tel}}, \overbrace{\text{ely}}$$

Subwords models

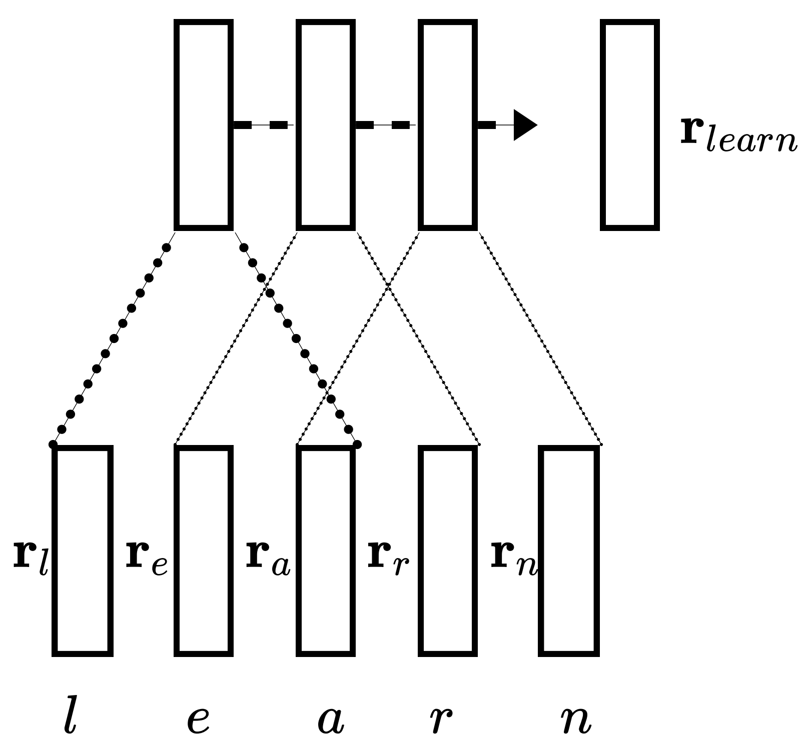
- The goal here is to obtain better word representation **from subwords**
  $\rightarrow$ An easier alternative is to use character n-grams !

$$\text{Unfortunately}$$

$$\downarrow$$

$$\overbrace{\text{Unf}}, \overbrace{\text{nfo}}, \ldots, \overbrace{\text{tel}}, \overbrace{\text{ely}}$$

- We can obtain word representations with a convolution, or a Bi-LSTM:



Compare the different architectures:

From Characters to Words to in Between: Do We Capture Morphology ? (Vania et al, 2017)

Subwords models : Applications

- With convolutions, to POS Tagging:

  Learning Character-level Representations for Part-of-Speech Tagging (Dos Santos et al, 2014)

Subwords models : Applications

- With convolutions, to POS Tagging:
  Learning Character-level Representations for Part-of-Speech Tagging (Dos Santos et al, 2014)

- With Bi-LSTM, to LM/POS Tagging:
  Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation (Ling et al, 2015)

Subwords models : Applications

- With convolutions, to POS Tagging:

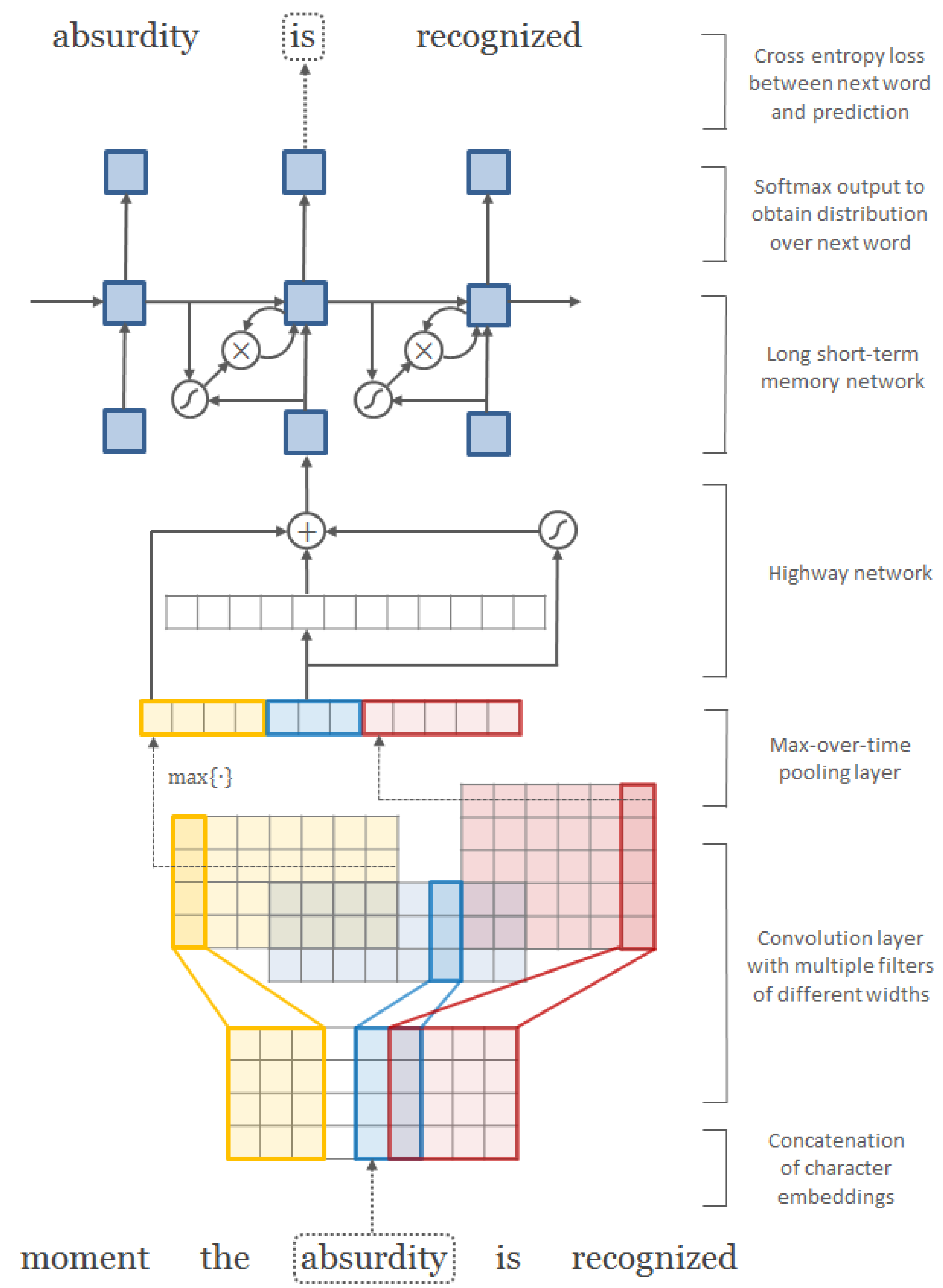
  Learning Character-level Representations for Part-of-Speech Tagging (Dos Santos et al, 2014)

- With Bi-LSTM, to LM/POS Tagging:

  Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation (Ling et al, 2015)

- With convolutions, to LM:

  Character-Aware Neural Language Models (Kim et al, 2015)



Parenthesis: Statistical MT

- Research on Machine Translation (**MT**) began eary (1950s)
  - System were mainly **rule-based**

Parenthesis: Statistical MT

- Research on Machine Translation (**MT**) began eary (1950s)
  - System were mainly **rule-based**
- Before neural approaches, MT was essentially **Statistical** MT
  - Data-driven: based on **parallel corpora**
  - Working at the sentence-level, and requiring **word alignement**
  - Mainly focused on **high-resource** languages

Parenthesis: Statistical MT

- Research on Machine Translation (**MT**) began eary (1950s)
  - System were mainly **rule-based**
- Before neural approaches, MT was essentially **Statistical** MT
  - Data-driven: based on **parallel corpora**
  - Working at the sentence-level, and requiring **word alignement**
  - Mainly focused on **high-resource** languages

$\rightarrow$ Assuming a sentence in a language f that we want to translate as a sentence in a target language e. We are looking for:

$$\text{argmax}_e P(e|f) = \text{argmax}_e \underbrace{P(f|e)}_{\text{Translation model}} \underbrace{P(e)}_{\text{Language model}}$$

Parenthesis: Statistical MT

- Research on Machine Translation (**MT**) began eary (1950s)
  - System were mainly **rule-based**
- Before neural approaches, MT was essentially **Statistical** MT
  - Data-driven: based on **parallel corpora**
  - Working at the sentence-level, and requiring **word alignement**
  - Mainly focused on **high-resource** languages

$\rightarrow$ Assuming a sentence in a language f that we want to translate as a sentence in a target language e. We are looking for:

$$\mathrm{argmax}_e P(e|f) = \mathrm{argmax}_e \underbrace{P(f|e)}_{\text{Translation model}} \underbrace{P(e)}_{\text{Language model}}$$

$\rightarrow$ Evaluation is done with **BLEU** score (measures n-gram overlap)

Statistical MT: Many components

The main translation models were **phrase-based**:

Statistical MT: Many components

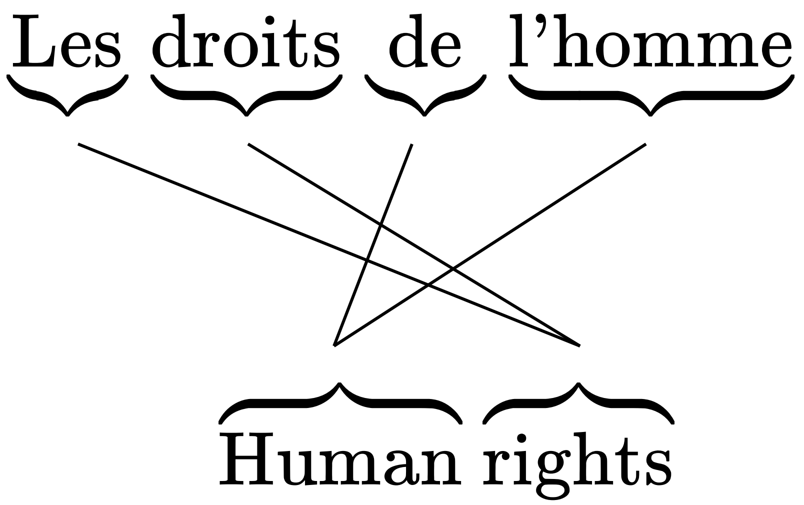
The main translation models were **phrase-based**:

- Segment the parallel sentences in **phrases**

Statistical MT: Many components

The main translation models were **phrase-based**:

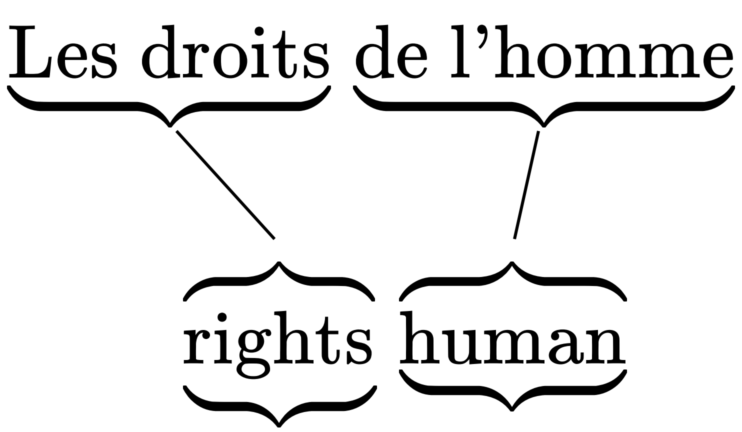
- Segment the parallel sentences in **phrases**
- Use **alignement** probabilities learnt with a separate model



Statistical MT: Many components

The main translation models were **phrase-based**:

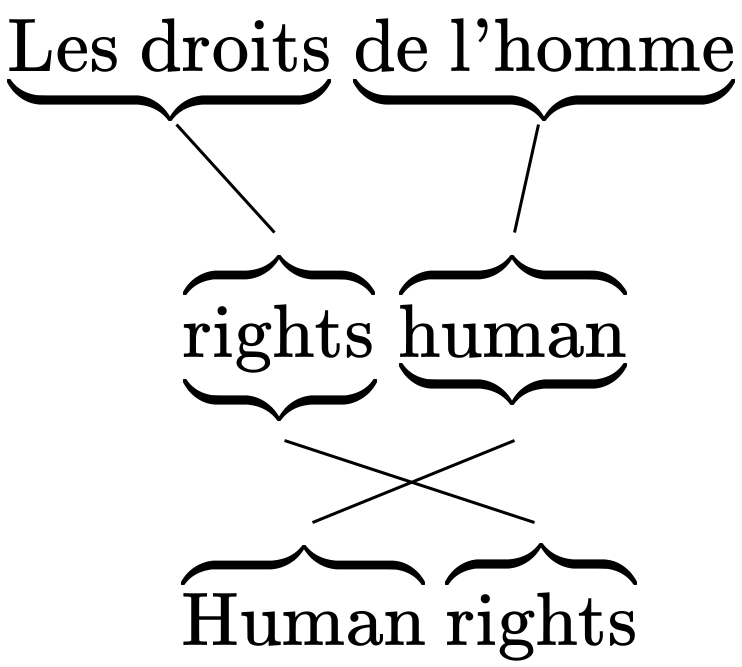
- Segment the parallel sentences in **phrases**
- Use **alignement** probabilities learnt with a separate model
- Generating a translation (**decoding**) implies:
  - Using **translation probabilities** to obtain target sentences



Statistical MT: Many components

The main translation models were **phrase-based**:

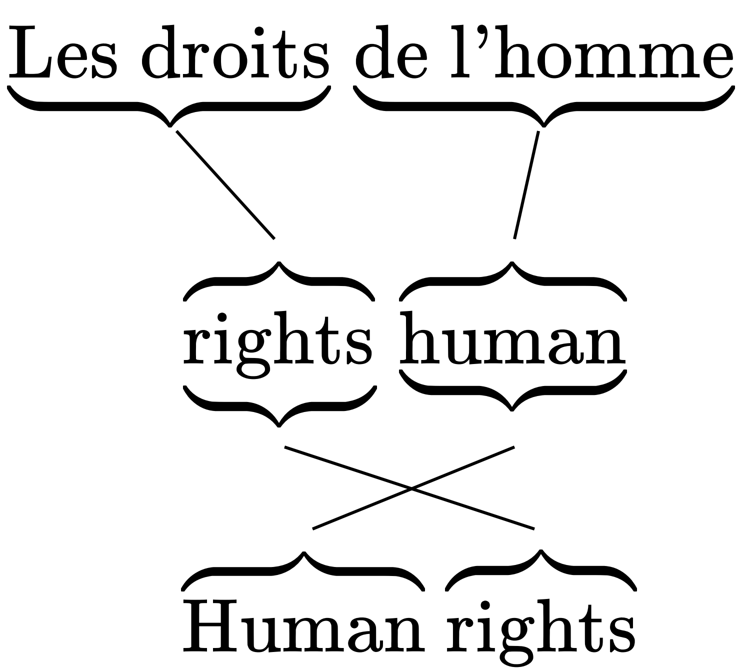
- Segment the parallel sentences in **phrases**
- Use **alignement** probabilities learnt with a separate model
- Generating a translation (**decoding**) implies:
  - Using **translation probabilities** to obtain target sentences
  - Using a **reordering** model along the language model



Statistical MT: Many components

The main translation models were **phrase-based**:

- Segment the parallel sentences in **phrases**
- Use **alignement** probabilities learnt with a separate model
- Generating a translation (**decoding**) implies:
  - Using **translation probabilities** to obtain target sentences
  - Using a **reordering** model along the language model



$\rightarrow$ Each of these (and others) steps implies many complex design choices !

Besides, *feature engineering*, supplementary *linguistic resources*, systems to maintain and update... *for every different language pair* !

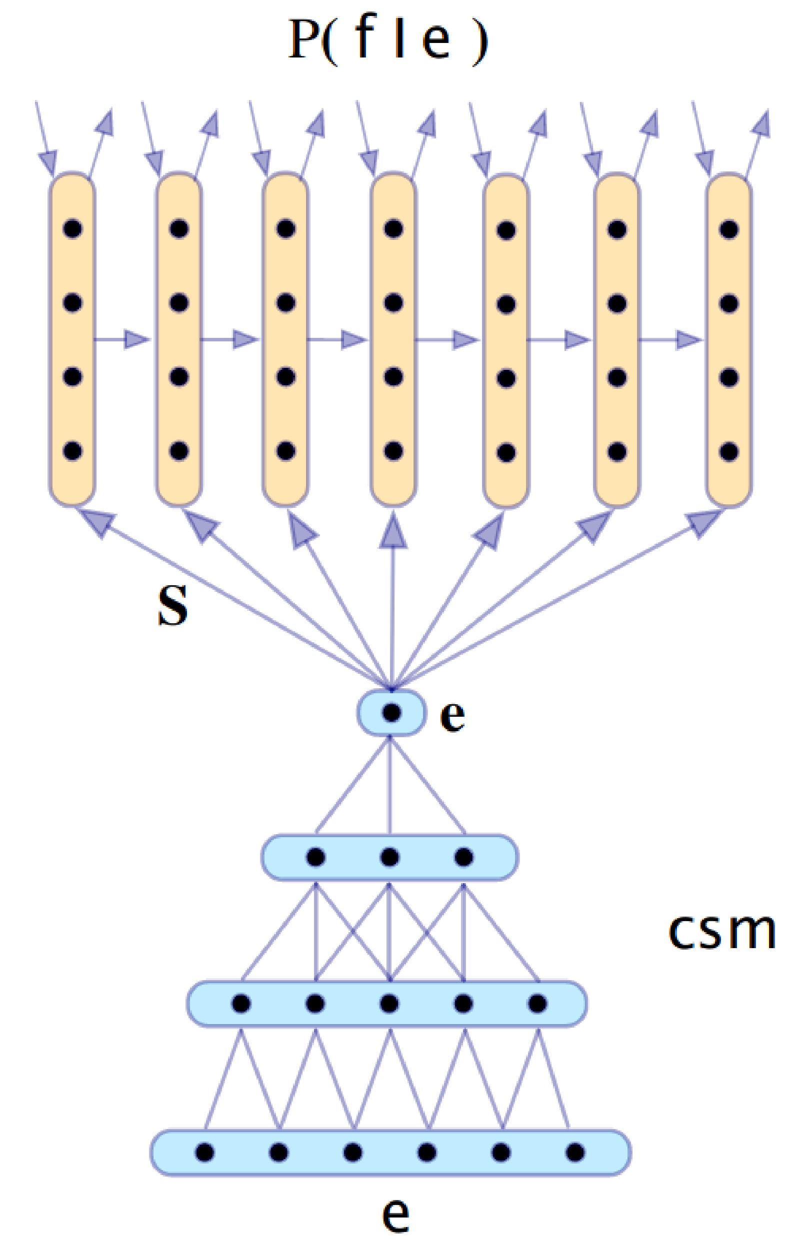
$\rightarrow$ Many other difficulties: using context, long-range dependancies, ...

Towards Sequence-to-sequence models

Recurrent Continuous Translation Models (Kalchbrenner et al, 2013)

One of the first neural model **for end-to-end machine translation**:

- The input sentence is encoded into **one global representation** via convolution
- The representation is decoded into the target sentence with a RNN



Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

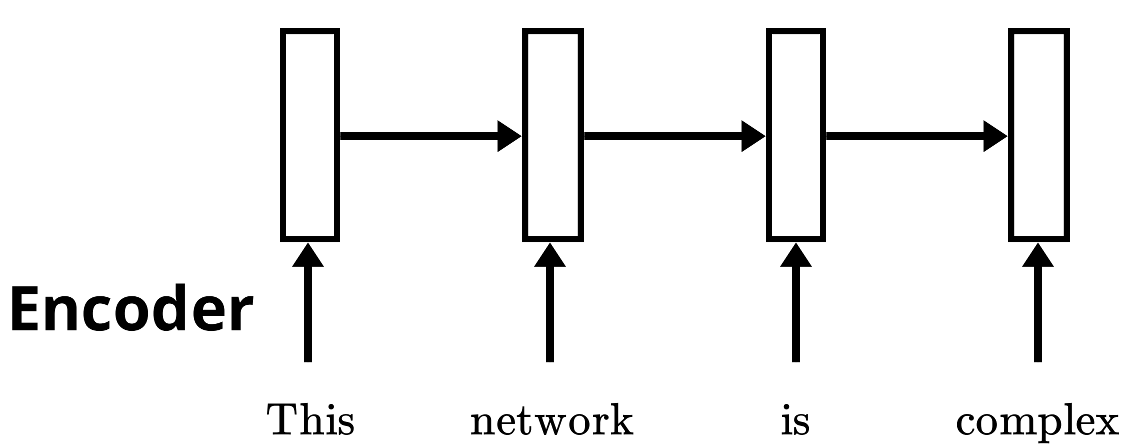
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation



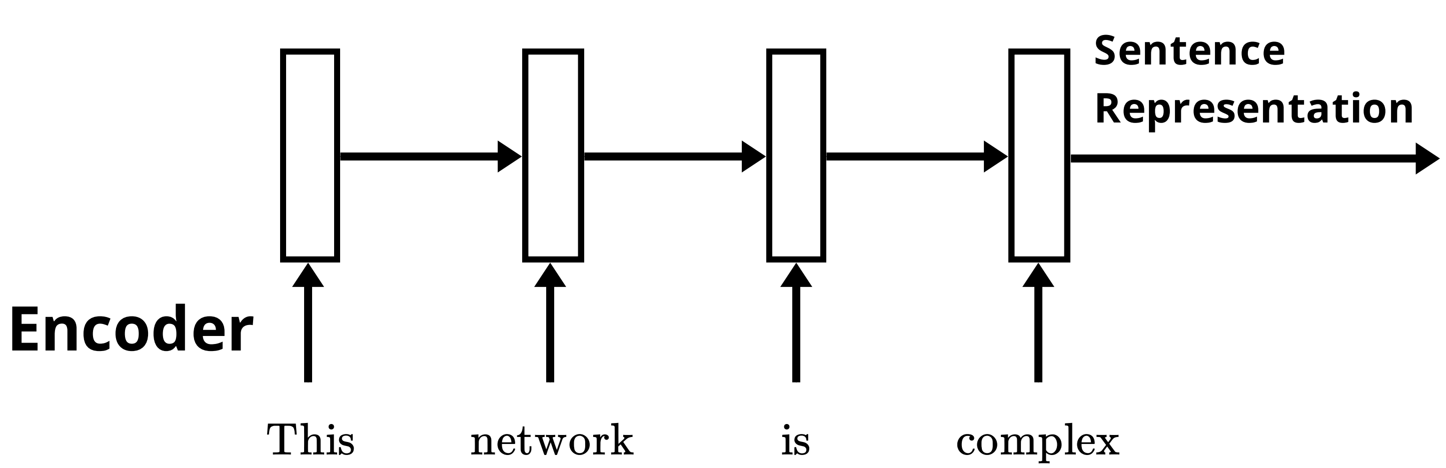
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation



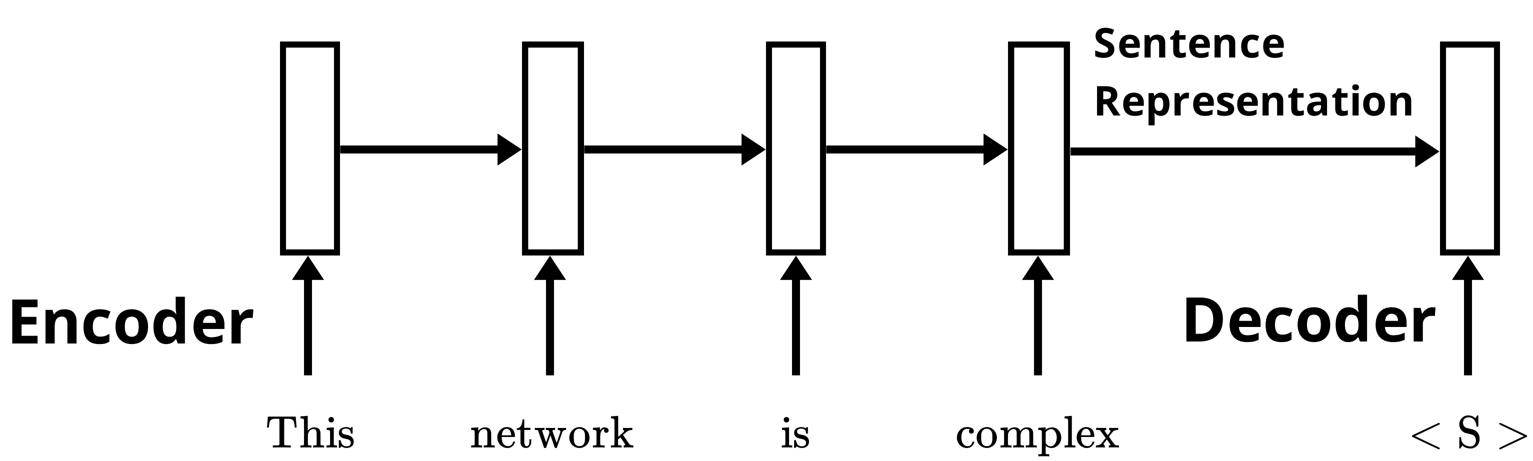
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation
- A decoder, generating a word at each step



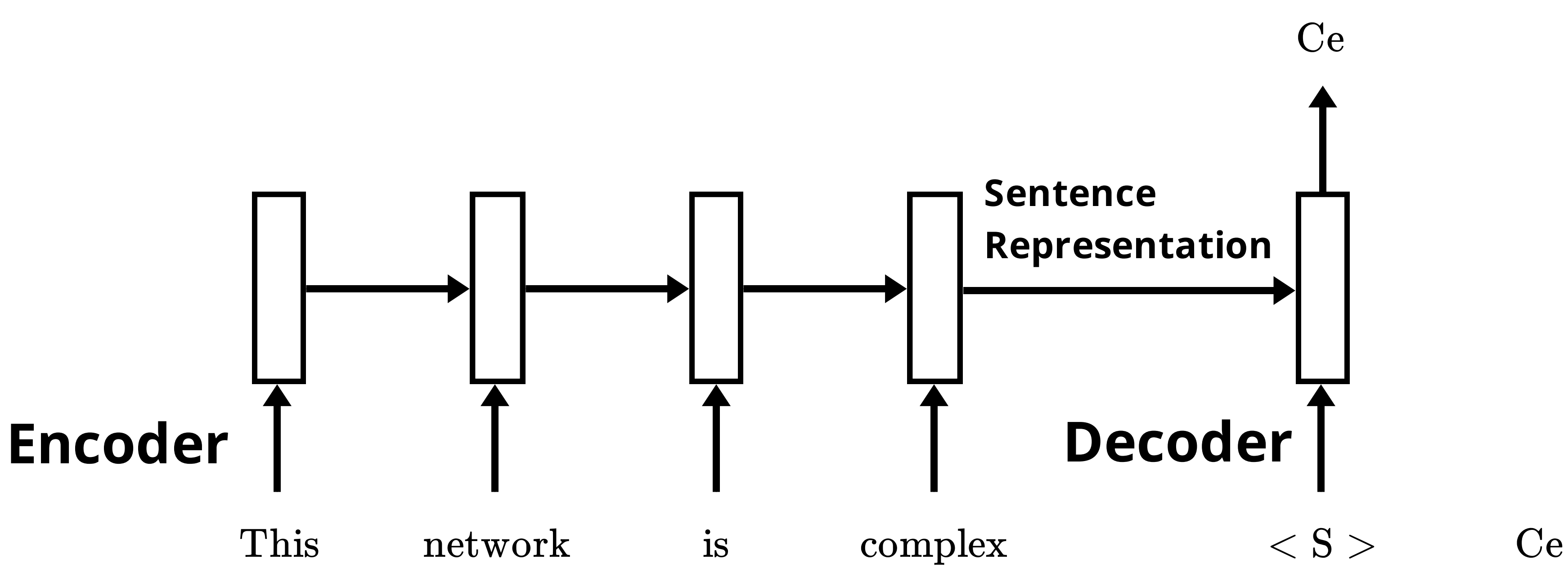
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation
- A decoder, generating a word at each step



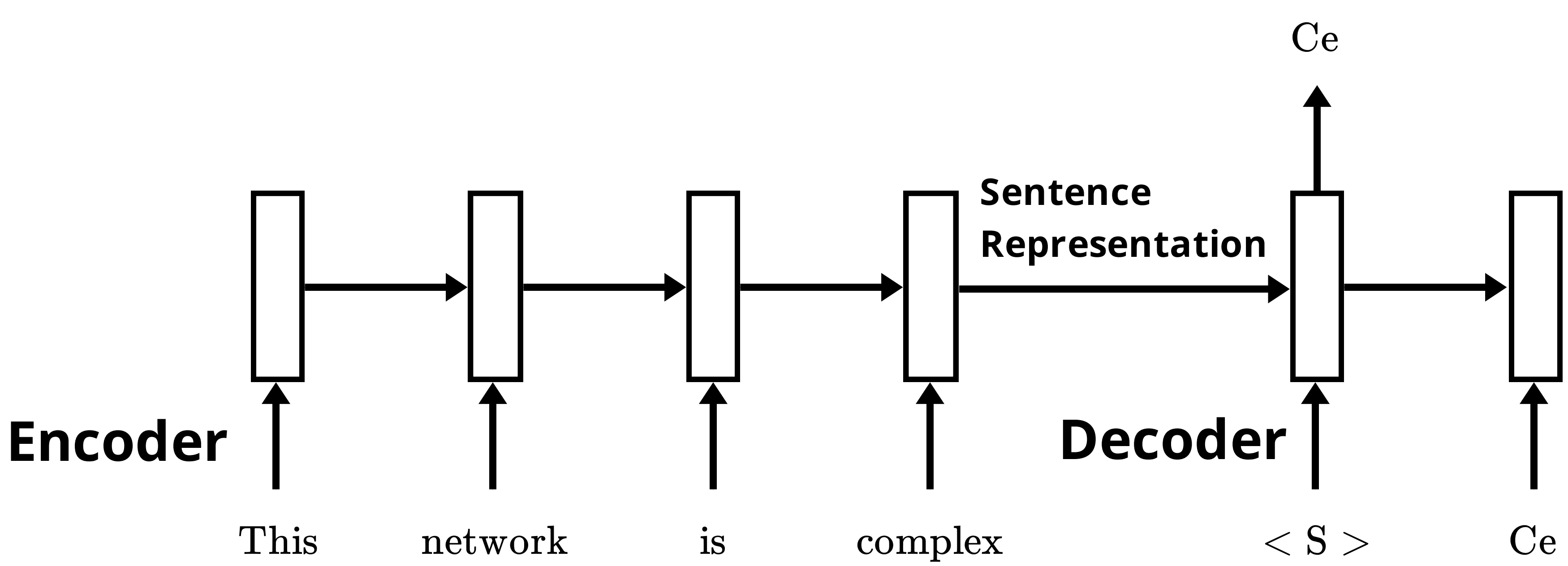
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation
- A decoder, generating a word at each step



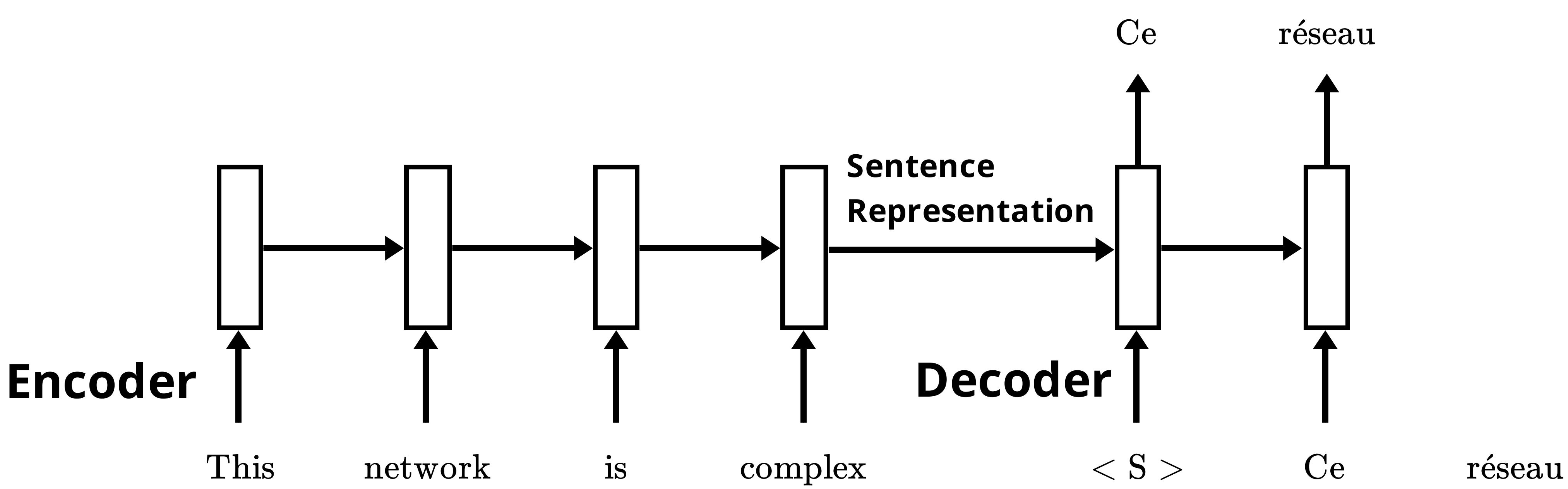
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation
- A decoder, generating a word at each step



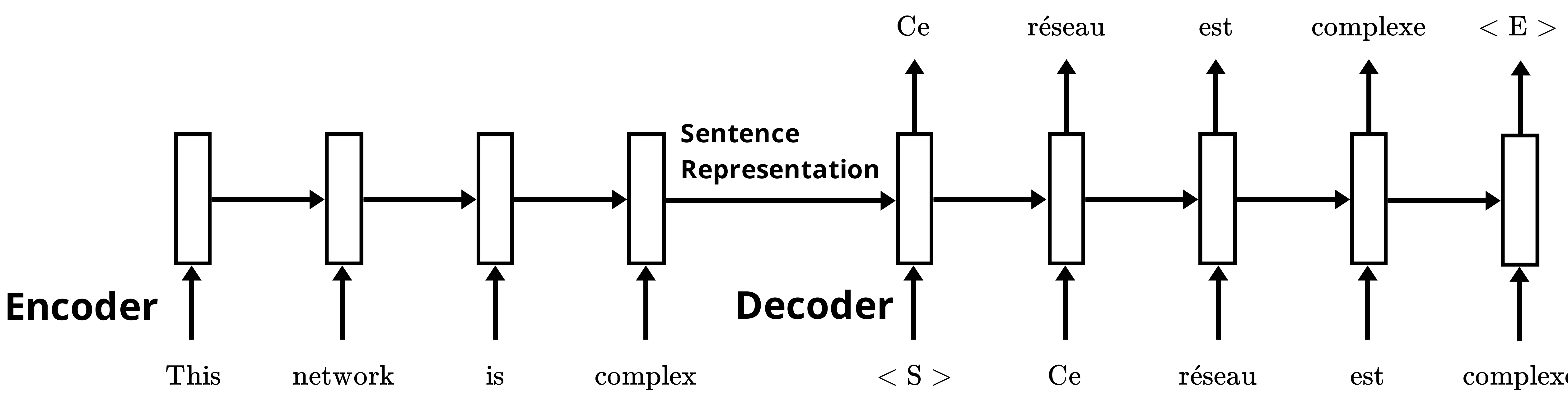
Sequence-to-sequence models for NMT

Sequence to Sequence Learning with Neural Networks, (Sutskever et al, 2014)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, (Cho et al, 2014)

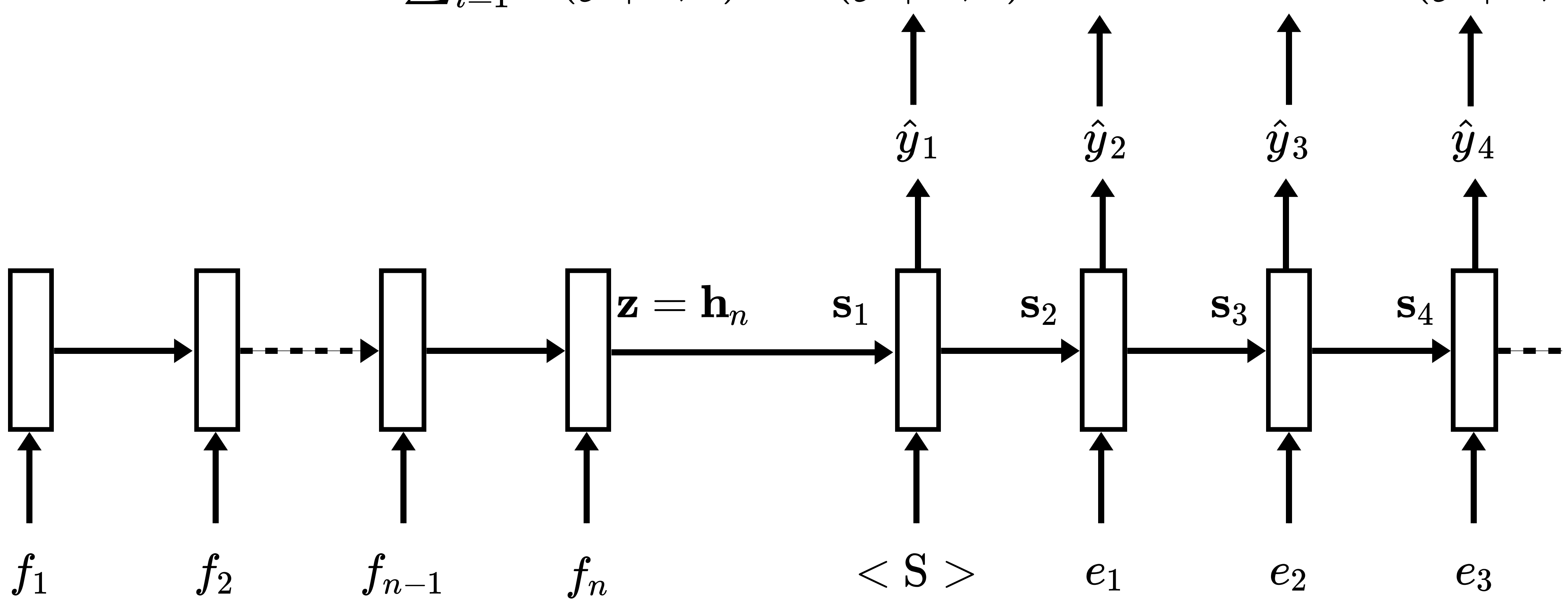
Sequence-to-sequence (Seq2seq) models are based on 2 RNNs:

- An encoder, outputting a sentence representation
- A decoder, generating a word at each step



Training Seq2seq models for NMT

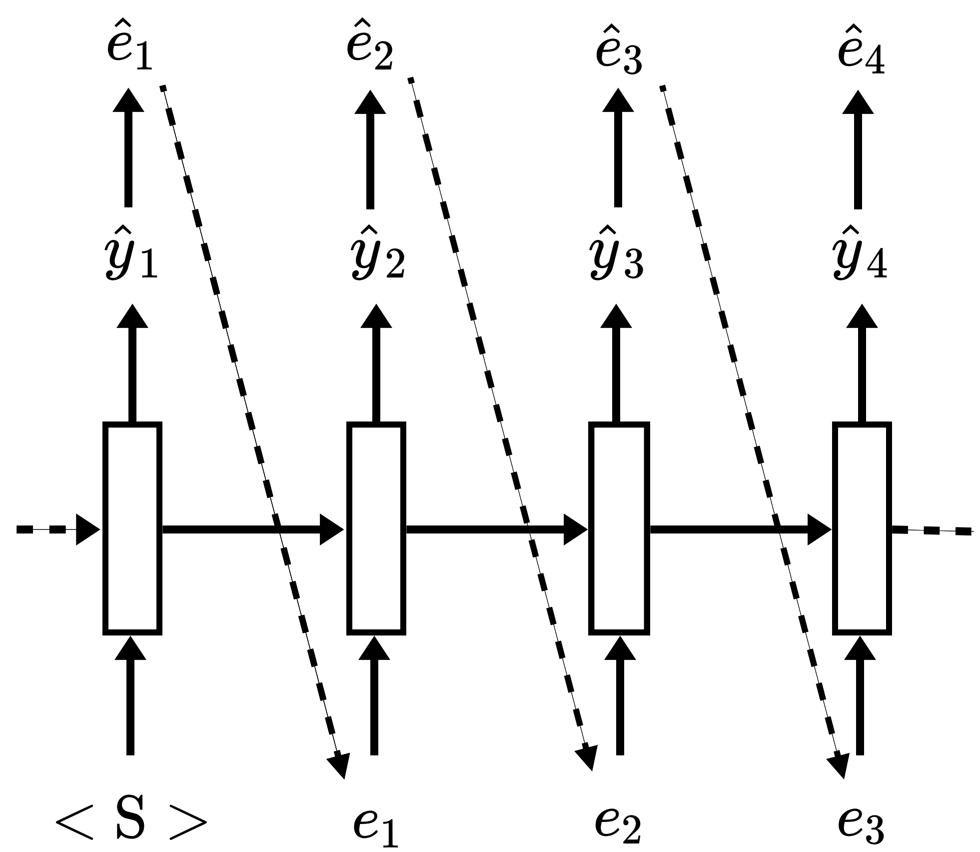
$$\sum_{i=1}^{N-1} l(\hat{y}_i | e_i, \mathbf{z}) = \; l(\hat{y}_1|e_1,\mathbf{z}) + ... \quad + \quad ... \quad + l(\hat{y}_4|e_1,..,e_4,\mathbf{z}) + ...$$



This NMT model can be seen as a conditional language model: we optimize $P(e|f) = P(e_1|f)P(e_2|e_1,f)...$

- As usual, we minimize the **negative log-likelihood**
- The model is optimized **end-to-end** with backpropagation
- During training, we give the model the **true inputs** (target sentence)

Greedy decoding of Seq2seq models



Greedy decoding:

$\rightarrow$ Taking the argmax at each step of the decoder and use it as next input, until the end symbol $<\mathrm{E}>$ is generated

- It does not take the **structure** of the target sentence into account
- If there is a **mistake** during decoding, impossible to go back
- But it is impossible to keep track of all possibilities ($|\mathcal{V}|$ is very large !)

Beam search for Seq2seq decoding

Intuition: Always keep track of the k most probable partial sequences of outputs (hypotheses) and use them to generate the next output

Beam search for Seq2seq decoding

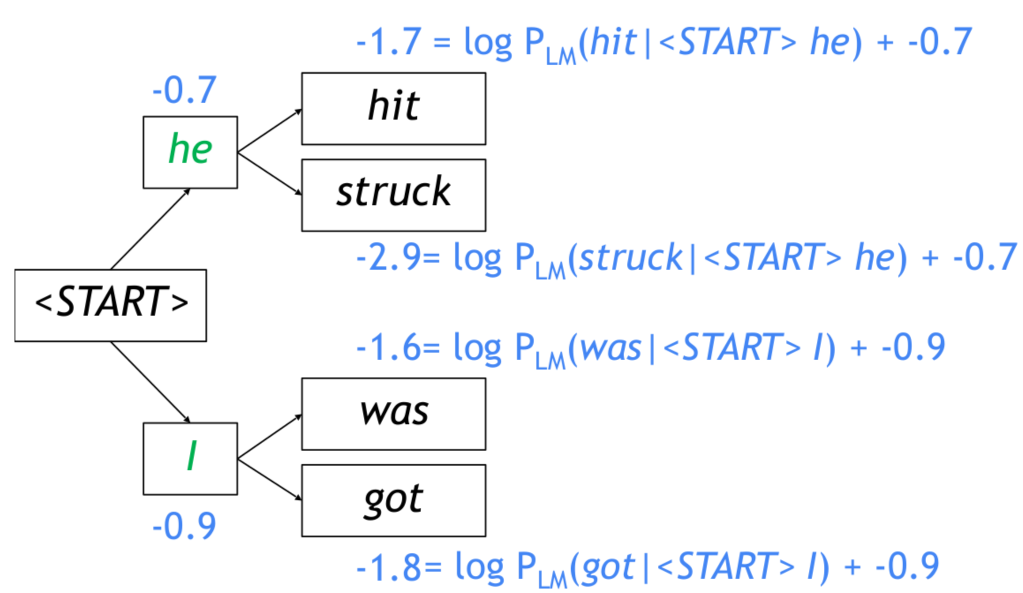
Intuition: Always keep track of the k most probable partial sequences of outputs (hypotheses) and use them to generate the next output

- k is the beam size (in practice around 5 to 10)

Beam search for Seq2seq decoding

Intuition: Always keep track of the k most probable partial sequences of outputs (hypotheses) and use them to generate the next output

- k is the beam size (in practice around 5 to 10)
- Search for the k top-scoring hypotheses at each step

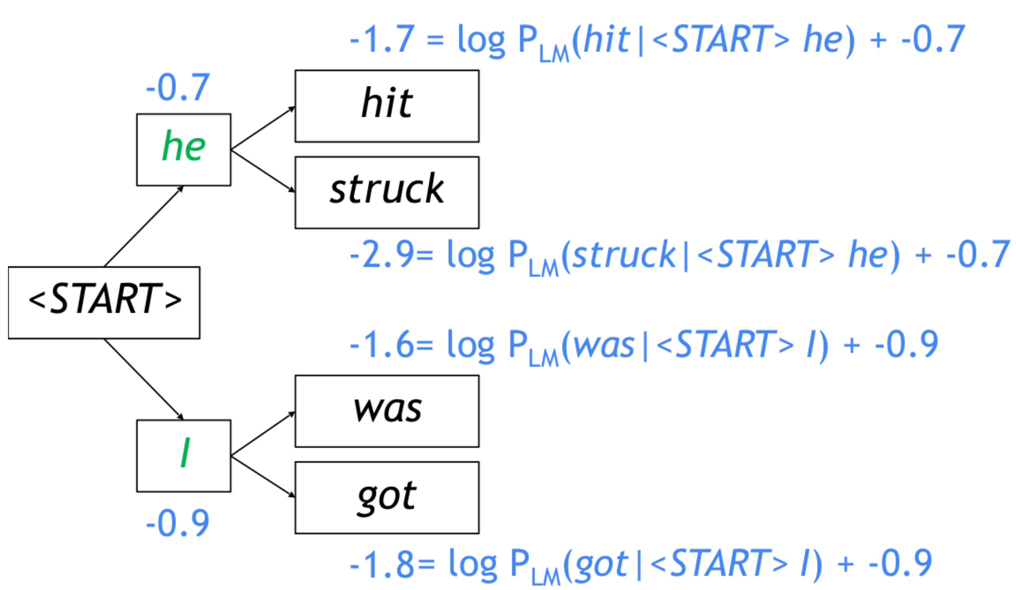


From Lecture 8, CS224N, Stanford University (Abigail See, Matthew Lamm)

Beam search for Seq2seq decoding

Intuition: Always keep track of the k most probable partial sequences of outputs (hypotheses) and use them to generate the next output

- k is the beam size (in practice around 5 to 10)
- Search for the k top-scoring hypotheses at each step



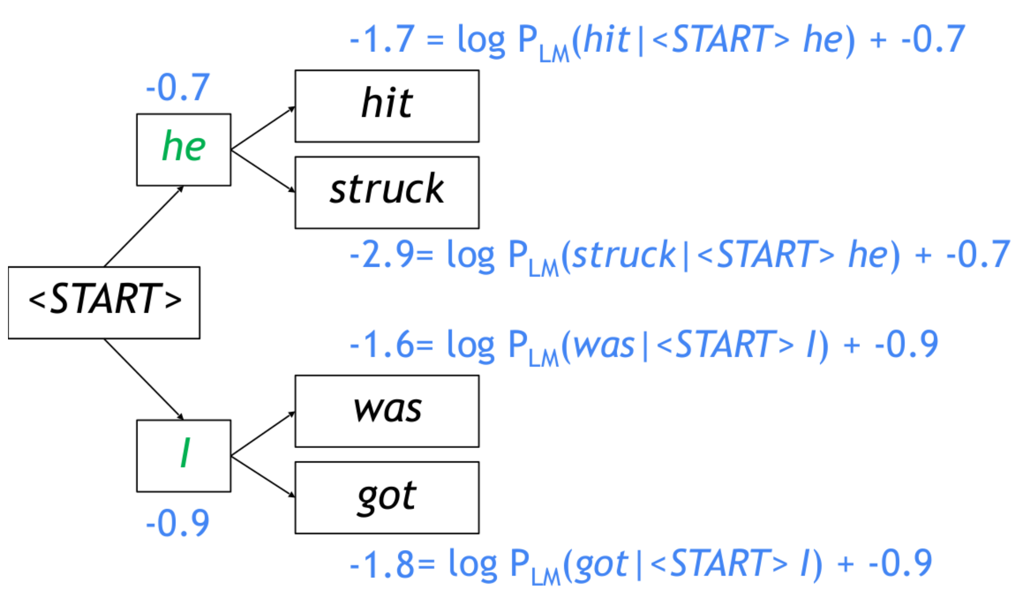
From Lecture 8, CS224N, Stanford University (Abigail See, Matthew Lamm)

- The search is not optimal, but it is very efficient !

Beam search for Seq2seq decoding

Intuition: Always keep track of the k most probable partial sequences of outputs (hypotheses) and use them to generate the next output

- k is the beam size (in practice around 5 to 10)
- Search for the k top-scoring hypotheses at each step



From Lecture 8, CS224N, Stanford University (Abigail See, Matthew Lamm)

- The search is not optimal, but it is very efficient !
- We may decode until we reach a limit length, or until we generate a limit number of completed sequences

Seq2seq models: Assesment

- NMT with Sequence-to-sequence models provided **huge gains in BLEU** compared to SMT, for many language pairs.

Seq2seq models: Assesment

- NMT with Sequence-to-sequence models provided **huge gains in BLEU** compared to SMT, for many language pairs.
- Translation are more **fluent** and use the context of the input sentence better.

Seq2seq models: Assesment

- NMT with Sequence-to-sequence models provided **huge gains in BLEU** compared to SMT, for many language pairs.
- Translation are more **fluent** and use the context of the input sentence better.
- Models are **far easier to train and maintain** and the same method is used for different language pairs.

Seq2seq models: Assesment

- NMT with Sequence-to-sequence models provided **huge gains in BLEU** compared to SMT, for many language pairs.
- Translation are more **fluent** and use the context of the input sentence better.
- Models are **far easier to train and maintain** and the same method is used for different language pairs.

- However, as often with neural models, NMT is less interpretable and difficult to control.

Seq2seq models: Assesment

- NMT with Sequence-to-sequence models provided **huge gains in BLEU** compared to SMT, for many language pairs.
- Translation are more **fluent** and use the context of the input sentence better.
- Models are **far easier to train and maintain** and the same method is used for different language pairs.

- However, as often with neural models, NMT is less interpretable and difficult to control.
- Besides, some problems are still there:
  - The large ($\rightarrow$ slow training) and closed **output vocabulary**
  - **Long-term dependencies**
  - It does not work well on **low-resource** language pairs

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !
  - Start $\mathcal{V}$ as the set of characters

```
5 low
2 lowest
6 newest
3 widest
```

$\longrightarrow \mathcal{V}_0 = \{\text{l,o,w,e,r,n,w,s,t,i,d}\}$

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !
  - Start $\mathcal{V}$ as the set of characters
  - Add **the most frequent pair of elements** of $\mathcal{V}$ as a new n-gram element

```
5 low
2 lowest
6 newest
3 widest
```

$$\mathcal{V}_0 = \{\mathrm{l,o,w,e,r,n,w,s,t,i,d}\}$$

$$\downarrow$$

$$\mathcal{V}_1 = \{\mathrm{l,o,w,e,r,n,w,s,t,i,d,}\mathbf{es}\}$$

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !
  - Start $\mathcal{V}$ as the set of characters
  - Add **the most frequent pair of elements** of $\mathcal{V}$ as a new n-gram element

```
5 low
2 lowest
6 newest
3 widest
```

$$\mathcal{V}_0 = \{\mathrm{l,o,w,e,r,n,w,s,t,i,d}\}$$

$$\mathcal{V}_1 = \{\mathrm{l,o,w,e,r,n,w,s,t,i,d},\mathbf{es}\}$$

$$\mathcal{V}_2 = \{\mathrm{l,o,w,e,r,n,w,s,t,i,d,es},\mathbf{est}\}$$

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !
  - Start $\mathcal{V}$ as the set of characters
  - Add **the most frequent pair of elements** of $\mathcal{V}$ as a new n-gram element
  - Process until a target size is reached for $\mathcal{V}$

```
5 low
2 lowest
6 newest
3 widest
```

$$\mathcal{V}_0 = \{\text{l,o,w,e,r,n,w,s,t,i,d}\}$$

$$\mathcal{V}_1 = \{\text{l,o,w,e,r,n,w,s,t,i,d,}\textbf{es}\}$$

$$\mathcal{V}_2 = \{\text{l,o,w,e,r,n,w,s,t,i,d,es,}\textbf{est}\}$$

$$\mathcal{V}_3 = \{\text{l,o,w,e,r,n,w,s,t,i,d,es,est,}\textbf{low}\}$$

Parenthesis: Byte Pair Encoding

Neural Machine Translation of Rare Words with Subword Units, (Seinrich et al, 2016)

- Byte Pair Encoding (BPE) is a **word segmentation algorithm** allowing to use a NMT model with an **open vocabulary** !
    - Start $\mathcal{V}$ as the set of characters
    - Add **the most frequent pair of elements** of $\mathcal{V}$ as a new n-gram element
    - Process until a target size is reached for $\mathcal{V}$

```
5 low
2 lowest
6 newest
3 widest
```

$$\mathcal{V}_0 = \{\text{l,o,w,e,r,n,w,s,t,i,d}\}$$

$$\mathcal{V}_1 = \{\text{l,o,w,e,r,n,w,s,t,i,d,}\textbf{es}\}$$

$$\mathcal{V}_2 = \{\text{l,o,w,e,r,n,w,s,t,i,d,es,}\textbf{est}\}$$

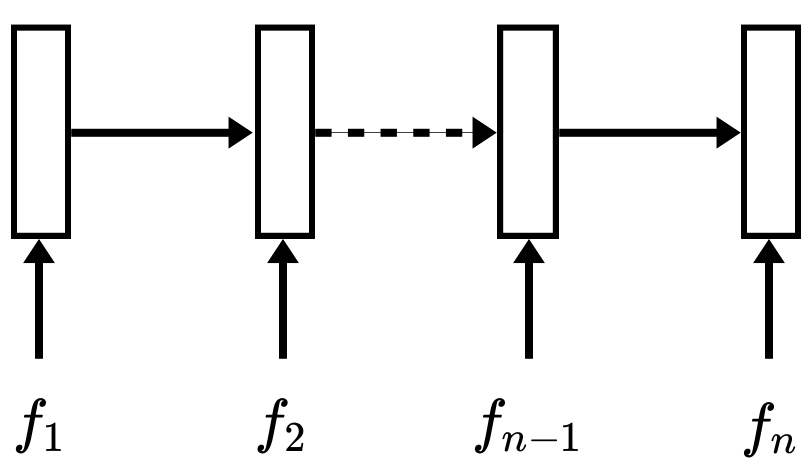
$$\mathcal{V}_3 = \{\text{l,o,w,e,r,n,w,s,t,i,d,es,est,}\textbf{low}\}$$

- Variant: **wordpiece/sentencepiece** - Tokenize in order to greedily reduce perplexity

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

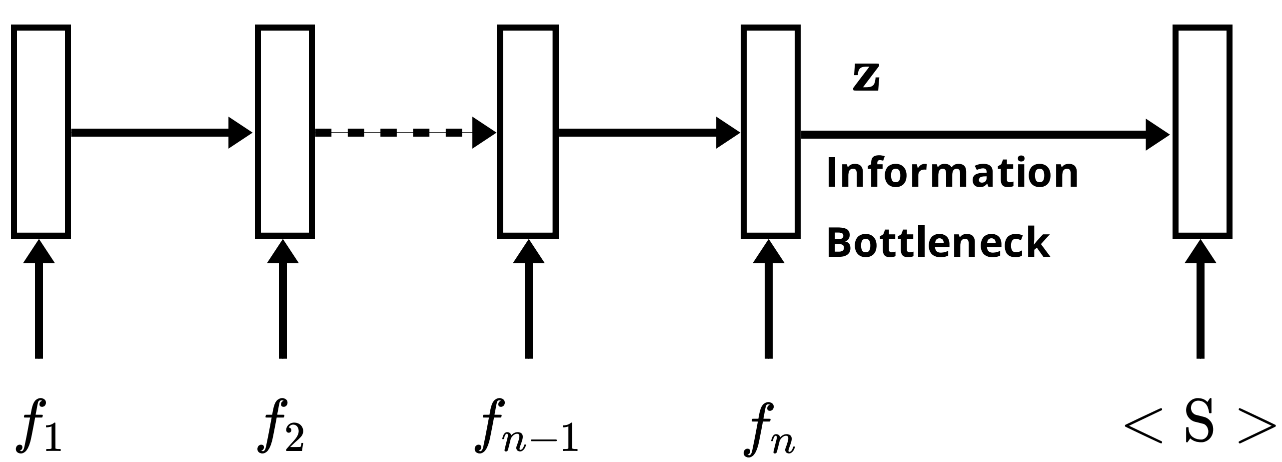
$\rightarrow$ **Attention** mechanism !



Seq2seq models: with Attention

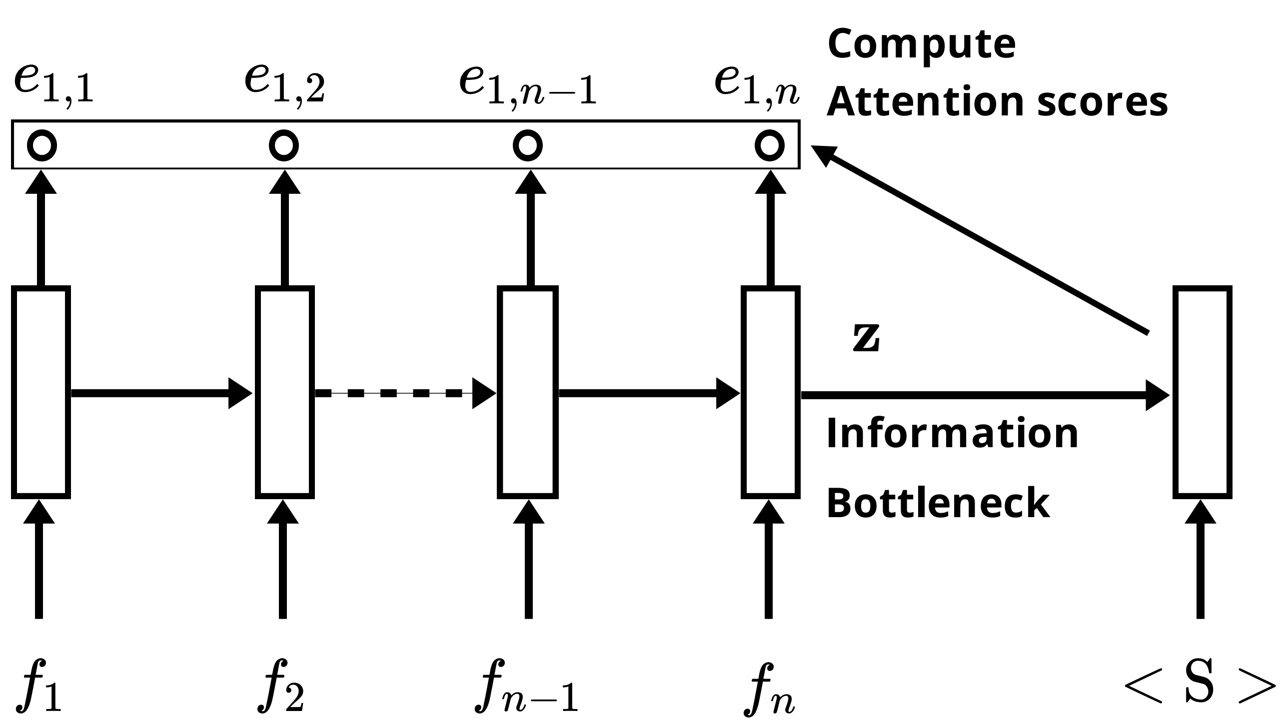
Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

$\rightarrow$ **Attention** mechanism !



Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...



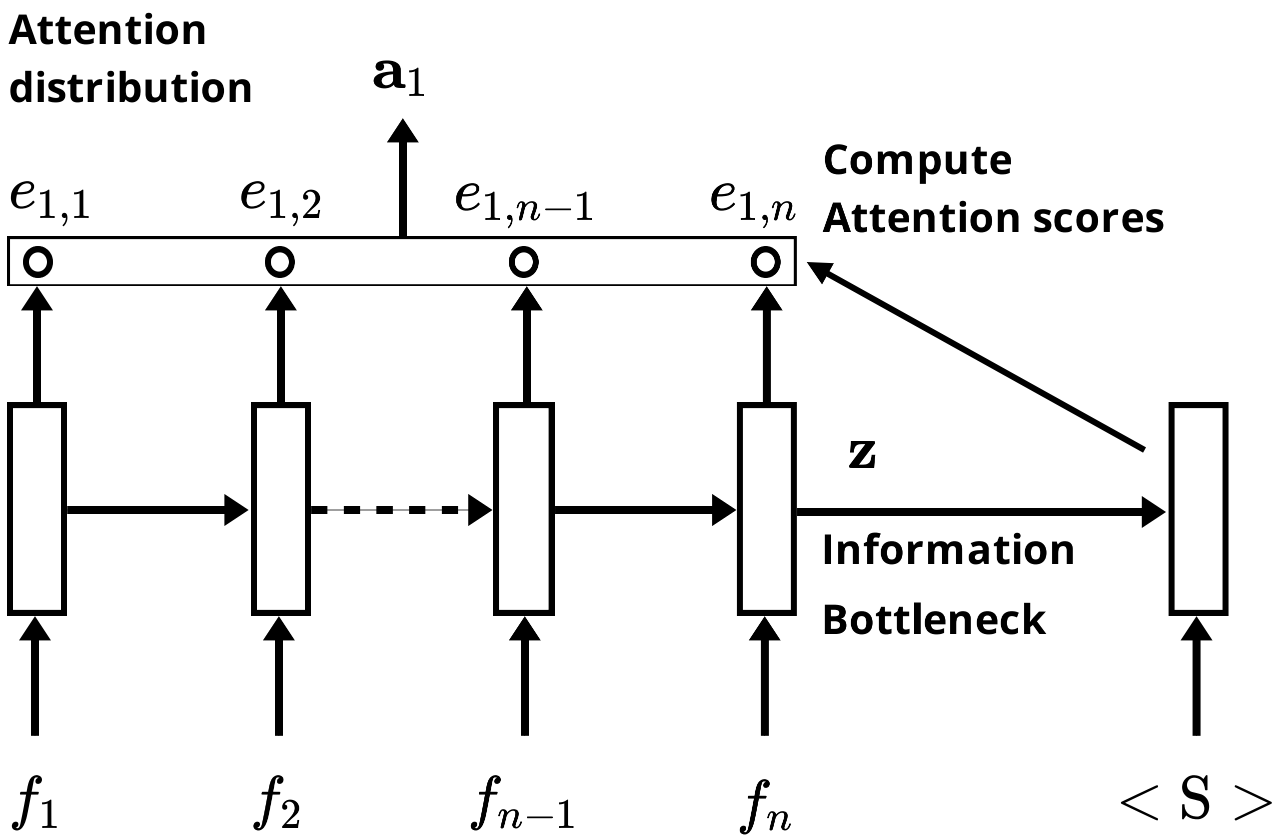
$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:

$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

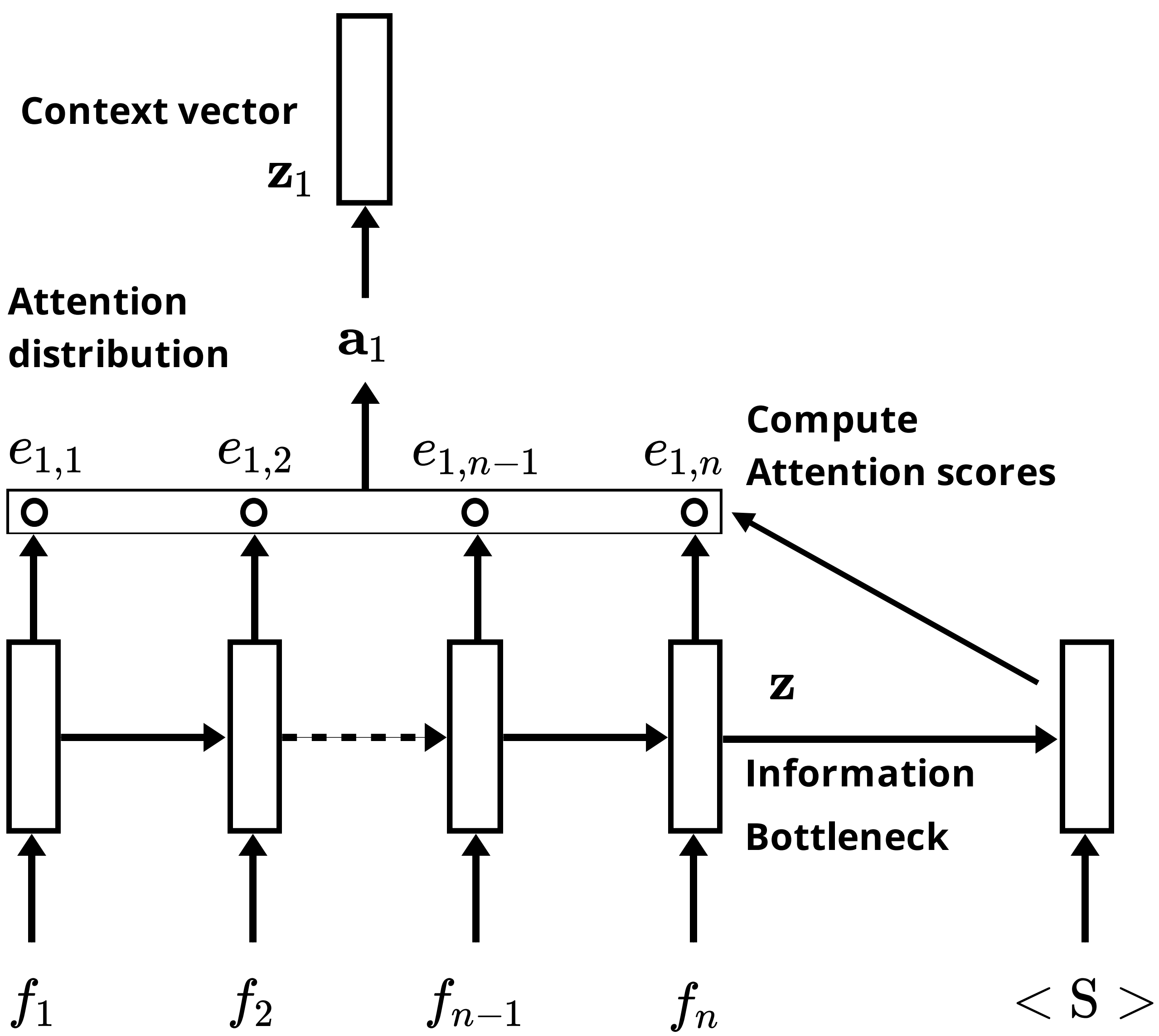


$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

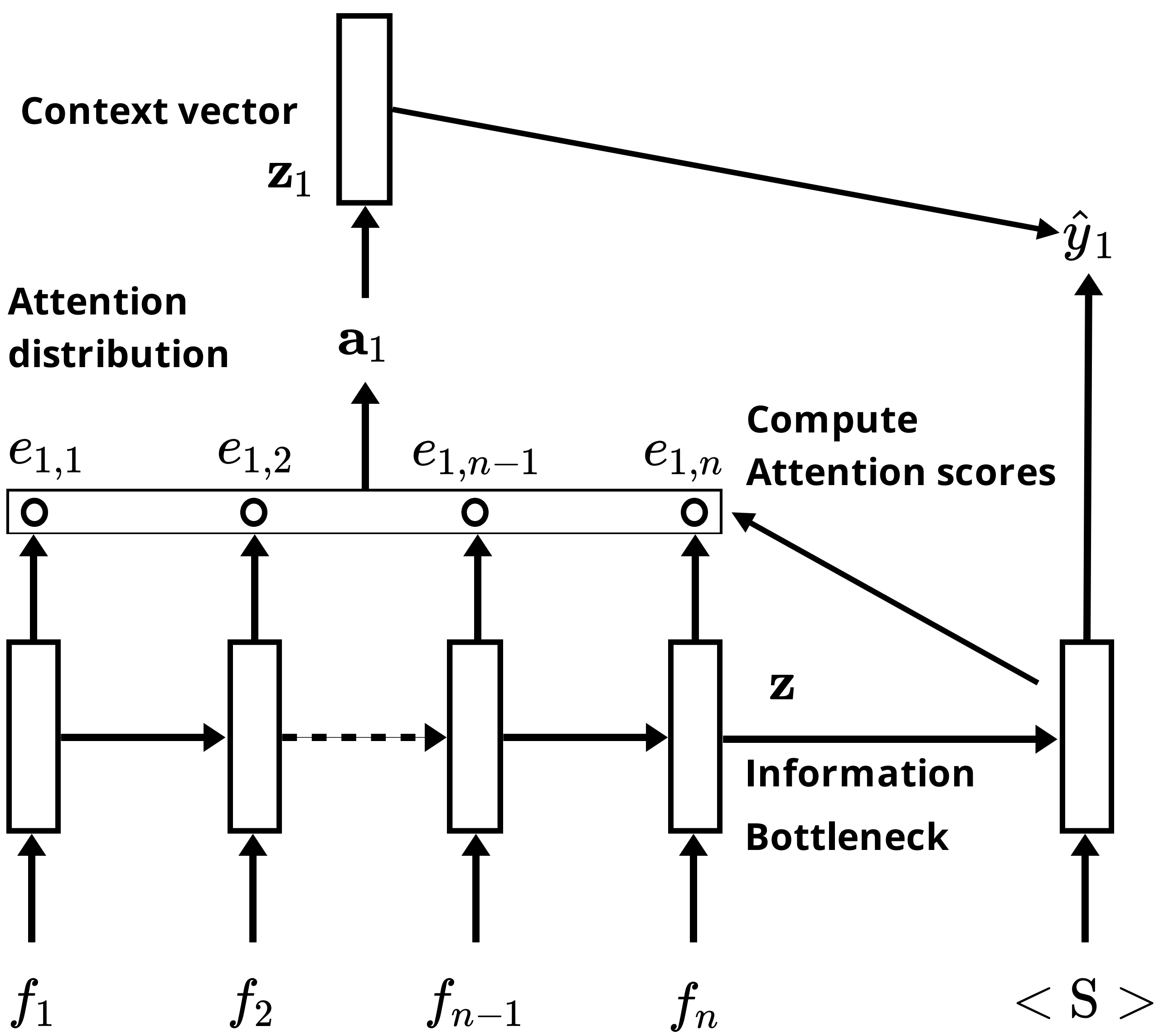


$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$
- The context vector is the combination of hidden states:
$$\mathbf{z}_i = \sum_{j=1}^{n} a_{i,j} \mathbf{h}_j$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

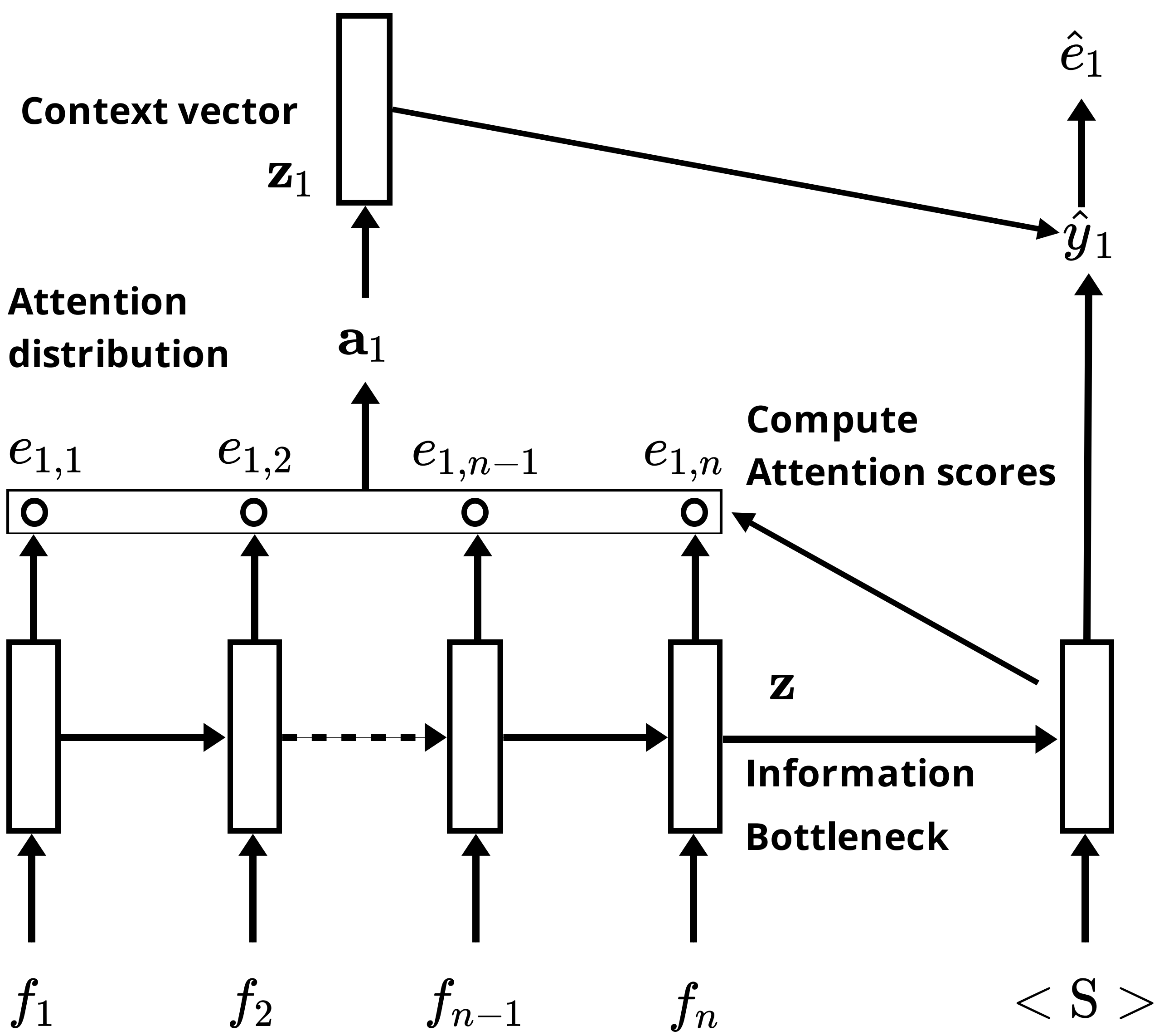


$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$
- The context vector is the combination of hidden states:
$$\mathbf{z}_i = \sum_{j=1}^{n} a_{i,j} \mathbf{h}_j$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

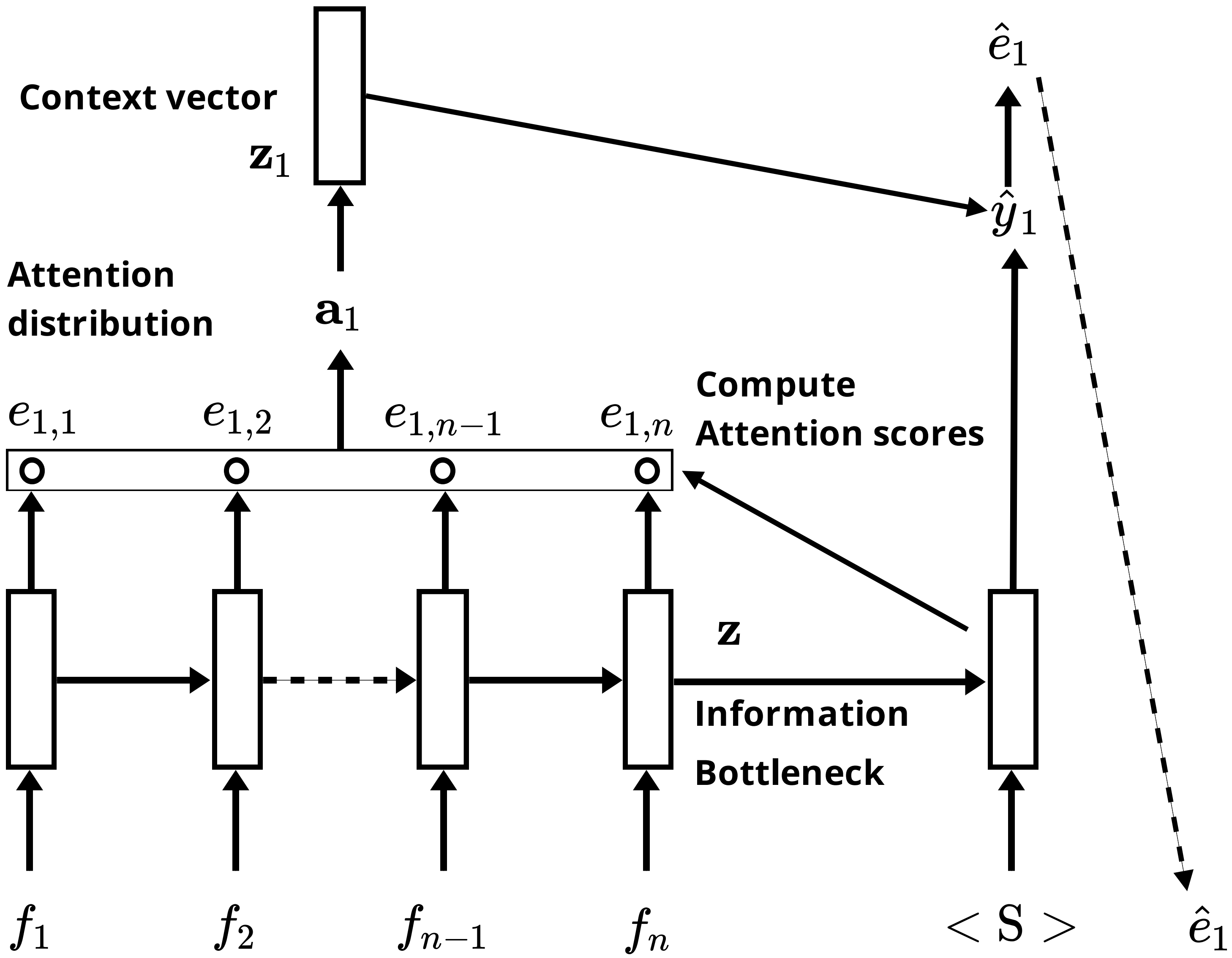


$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$
- The context vector is the combination of hidden states:
$$\mathbf{z}_i = \sum_{j=1}^{n} a_{i,j}\mathbf{h}_j$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...

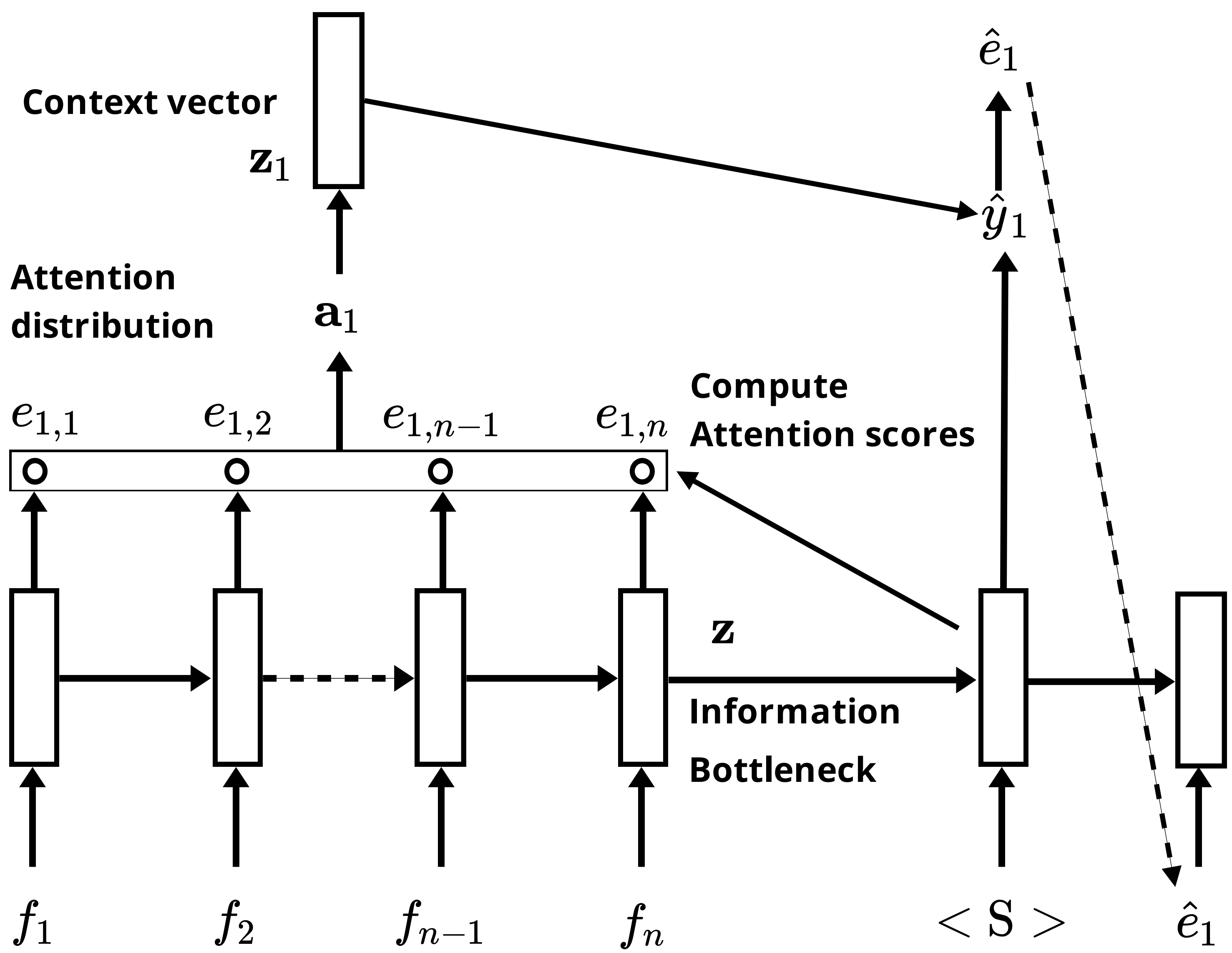


$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$
- The context vector is the combination of hidden states:
$$\mathbf{z}_i = \sum_{j=1}^{n} a_{i,j} \mathbf{h}_j$$

Seq2seq models: with Attention

Difficulty: encode a sentence in a fixed-sized vector ? Besides, long-term dependencies are not well considered during decoding...



$\rightarrow$ **Attention** mechanism !

- Scores are computed with an alignement function f:
$$s_{i,j} = f(\mathbf{s}_i, \mathbf{h}_j)$$
- These scores give the attention weights:
$$\mathbf{a}_i = softmax(\mathbf{s}_i)$$
- The context vector is the combination of hidden states:
$$\mathbf{z}_i = \sum_{j=1}^{n} a_{i,j} \mathbf{h}_j$$

Attention for NMT

Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et al, 2014)

- Attention largely improves NMT performance !

Attention for NMT

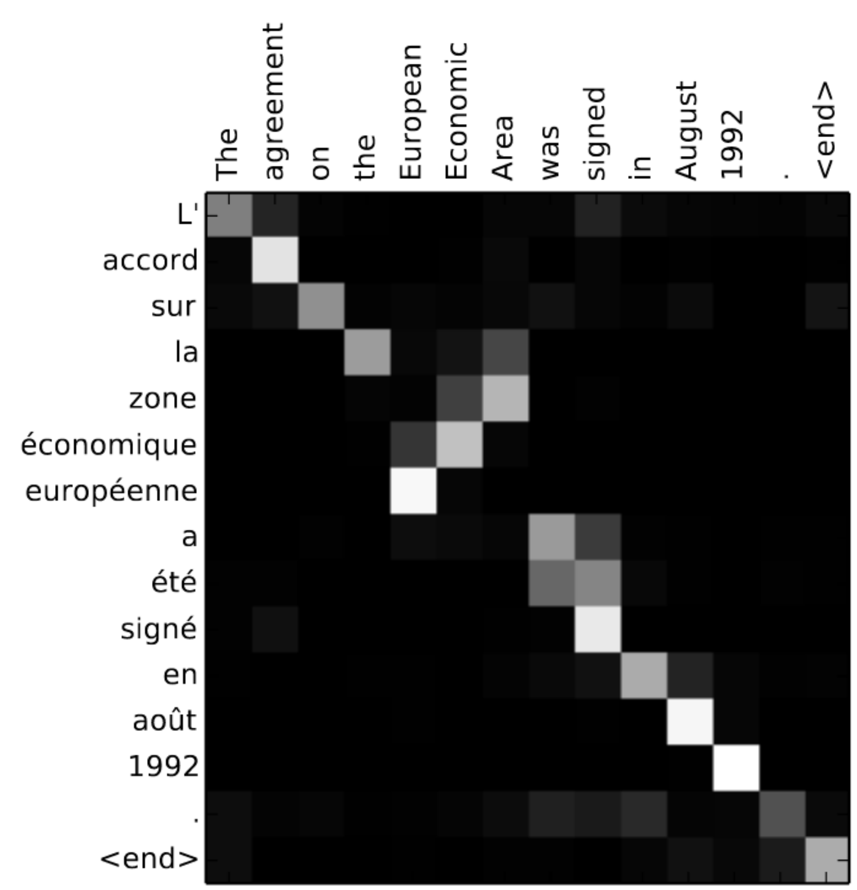
Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et al, 2014)

- Attention largely improves NMT performance !
- Besides solving the information bottleneck problem, it allows the model to reflect **far longer dependencies**.

Attention for NMT

Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et al, 2014)

- Attention largely improves NMT performance !
- Besides solving the information bottleneck problem, it allows the model to reflect **far longer dependencies**.
- Attention gives **soft alignement** on generated translations:



NMT as Catalyst

- **Sequence to sequence** models can be applied to any tasks where the input is a sequence and the output can be modeled as a sequence !

NMT as Catalyst

- **Sequence to sequence** models can be applied to any tasks where the input is a sequence and the output can be modeled as a sequence !
- Similarly, **Attention** is a general technique, that can be applied within other models and to many other tasks, allowing to obtain a *fixed sized* representation from an *arbitrary number* of representations

NMT as Catalyst

- **Sequence to sequence** models can be applied to any tasks where the input is a sequence and the output can be modeled as a sequence !
- Similarly, **Attention** is a general technique, that can be applied within other models and to many other tasks, allowing to obtain a *fixed sized* representation from an *arbitrary number* of representations

$\rightarrow$ Neural Machine Translation has been a **catalyst** for the devellopment of deep learning techniques for NLP - and Seq2seq/Attention have been applied to many other tasks:

- Summarization (Document → Summary)
- Dialogue (Previous utterances → Next utterance)
- Parsing (Input text → Output parse sequence)
- ....

$\rightarrow$ Many other improvements have been made these last years, - some of these tasks are now catalysts themselves.

Acknowledgements & References

From slides by **Alexandre Allauzen** and the **CS224N** Stanford class

Some references:

- *The Deep Learning Book*, Chapters 9, 10 and 12.4

- *Speech and Language Processing*, Chapters 7,9 and 10 - but look at the rest !
- Section 1:
  - *A Bit of Progress in Language Modeling, (J. Goodman, 2001)*
  - *Improved backing-off for n-gram language modeling (Kneser et al, 1995)*
- Section 2:
  - *Strategies for training large vocab-ulary neural language models (Chen et al, 2016)*
  - *Efficient softmax approximation for gpus (Grave et al, 2016)*
- Section 3:
  - *On the difficulty of training Recurrent Neural Networks, (Pascanu et al, 2013)*
  - *Understanding LSTM Networks*
  - *An empirical exploration of recurrent network architectures (Jozefowicz et al, 2015)*