COMP0087 25/26
Lecture 4: Neural Language Models

23/01/2026

comp0087.github.io

Recap: N-gram Language Model

Goal: Estimate the probability of the next word given previous words.

$P(\text{UCL} \mid \text{I am studying AI at})$

Counting (“I am … at”, “I am … at UCL”)

Estimate probability by relative frequency

Key idea: Probability is estimated from observations in data

Recap: N-gram Language Model

Downsides:

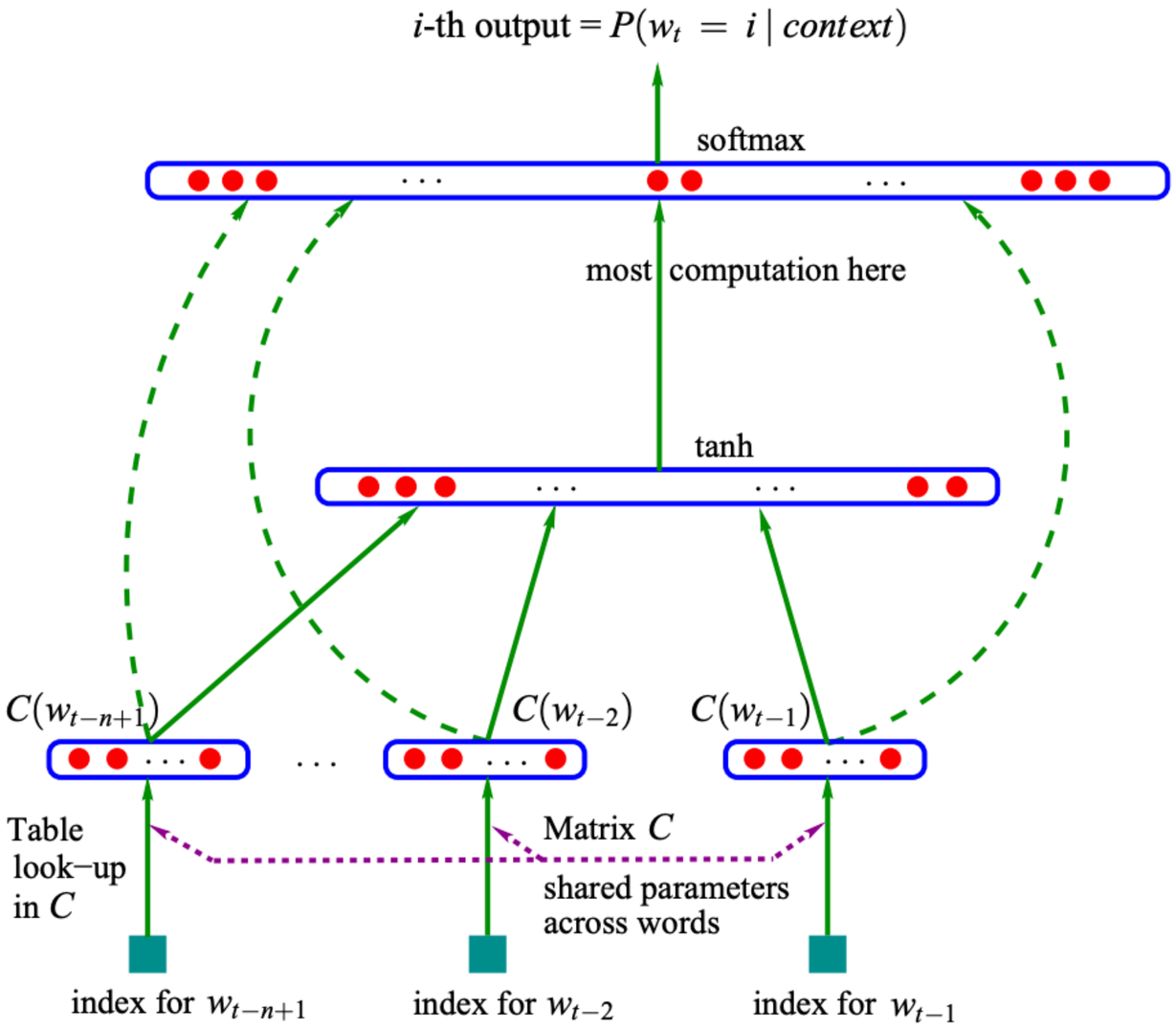
- sparsity (lots of zeros)
- poor generalisation
- high cost (with the “5-grams at most”)

Solution: Neural Nets

N-gram modelling with neural nets

- Motivation: Address sparsity and poor generalisation in count-based n-gram models.
- Simple case: Feedforward Neural Network (FNN)
- Fixed-size context

N-gram modelling with neural nets



N-gram modelling with neural nets

I am studying NLP at University College London.

Self-supervised training data

<s> I am studying NLP at University College London </s>

(<s> I am, studying)

(I am studying, NLP)

…

(at University College, London)

(University College London, </s>)

How the input is constructed

(I am studying, NLP)

Look up in dictionary

(42, 15, 308) -> 127

Word	Index
I	42
am	15
NLP	127
...	...
studying	308

How the input is constructed

Look up embedding for each word

Each word index maps to a dense vector of dimension d (e.g., d = 128)

Embedding Matrix E (size: V × d, where V = vocabulary size)

```
Word "I"        (index 42)  → E[42]  = [0.12, -0.45, 0.78, ...]
Word "am"       (index 15)  → E[15]  = [-0.33, 0.21, 0.05, ...]
Word "studying" (index 308) → E[308] = [0.67, 0.14, -0.29, ...]
```

Embedding Matrix Initialisation

From pretrained word embedding (word2vec)

- Pros: Faster convergence, better generalisation.
- Cons: Domain mismatch.

Random initialised (uniform, gaussian)

How the input is constructed

Concatenate the 3 embeddings

Input vector $x = [\ E[42]\ ;\ E[15]\ ;\ E[308]\ ]$

Final input dimension: $3 \times d$ (e.g., $3 \times 128 = 384$)

Hidden Layer Computation

Hidden layer transformation

$$\mathbf{h} = \text{activation}\left(\mathbf{W}_h\mathbf{x} + \mathbf{b}_h\right)$$

$\mathbf{W}_h$: hidden layer weight matrix (size: hidden_size × 3d)

$\mathbf{b}_h$: hidden layer bias vector (size: hidden_size)

Output Layer: Computing Word Probabilities

Project hidden state to vocabulary size

$$\mathbf{z} = \mathbf{W}_o\mathbf{h} + \mathbf{b}_o$$

$\mathbf{W}_o$: output weight matrix (size: V × hidden_size)

$\mathbf{b}_o$: output bias vector (size: V)

$\mathbf{z}$: logits (raw scores for each word in vocabulary)

Output Layer: Computing Word Probabilities

Normalize logits to a valid distribution

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

$$\hat{y}_i = P(w_i \,|\, \text{context}) = \frac{\exp(z_i)}{\sum_j^V \exp(z_j)}$$

Result: Probability distribution over entire vocabulary

Output Layer: Computing Word Probabilities

Word	Logit	Probability
NLP	2.8	0.35
hard	1.5	0.12
math	1.2	0.08
…		

Training: Loss computation

Ground truth (NLP): [0, 0, 0, …, 1, 0, 0, 0]

Cross-Entropy Loss:

$$\text{Loss} = -\sum_{i}^{V} y_i log(\hat{y}_i) = -log(\hat{y}_{\text{target}})$$

Example: Loss = –log(0.35) = 1.05

Training: gradient update

Trainable parameters $\theta = \{\mathbf{E}, \mathbf{W}_h, \mathbf{b}_h, \mathbf{W}_o, \mathbf{b}_o\}$

Update parameters: $\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}$

α: learning rate (e.g., 0.001)

Summary

Step	Operation	Output
1	Word -> Index	42
2	Index -> Embedding	**e1, e2, e3**
3	Concat	**x**=**[e1;e2;e3]**
4	Hidden later	$\mathbf{h = ReLU(W_h x + b_h)}$
5	Output layer	$\mathbf{z = W_o h + b_o}$
6	Softmax	$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$

NN-LM Performance

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Bonus

You get a learned word embedding for free.

Year: 2003

The quality of learned versus pretrained?

Downside of NN-LM

I put the beer in the fridge because it was warm.

Downside of NN-LM

- Cannot model long-range dependencies
- Same core limitation as n-gram models

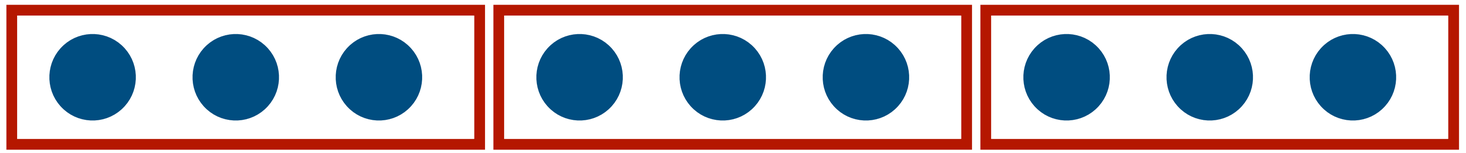
Downside of NN-LM

Scaling issue: limitation of N

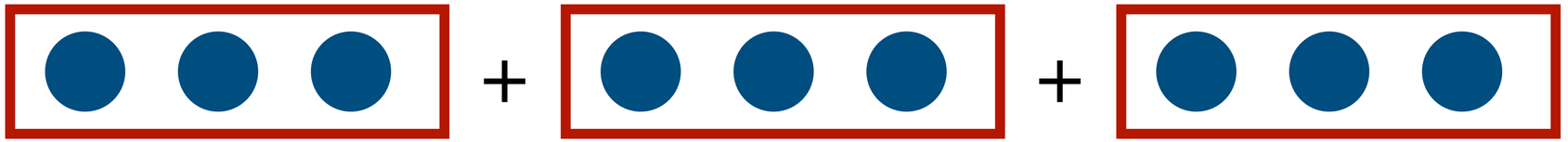
weight matrix grows rapidly

$$x \in \mathbb{R}^{nd_e} \qquad W \in \mathbb{R}^{d_h \times nd_e}$$

Downside of NN-LM



$$x \in \mathbb{R}^{3d}$$



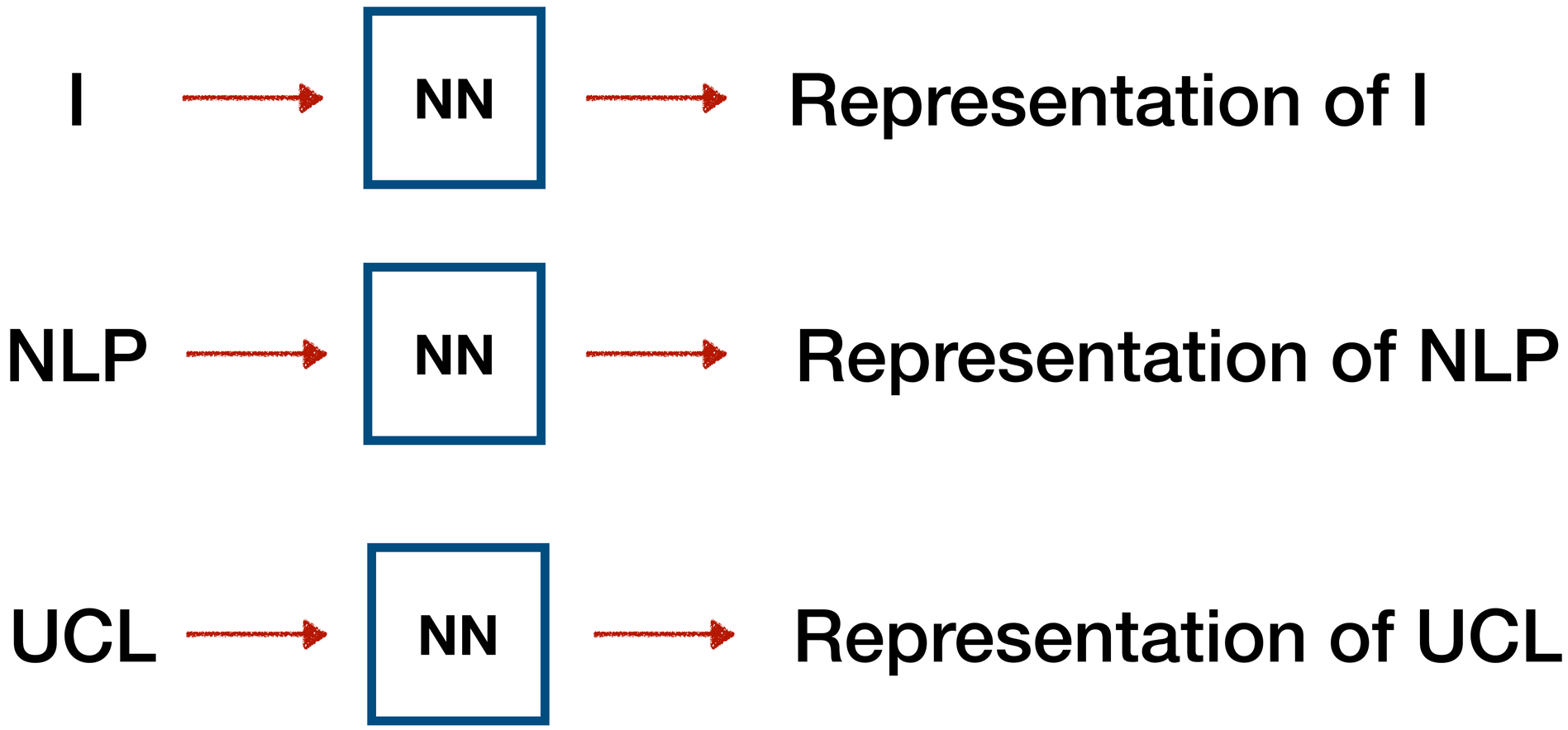
$$x \in \mathbb{R}^{d}$$

How can we dealing with variable length of input?

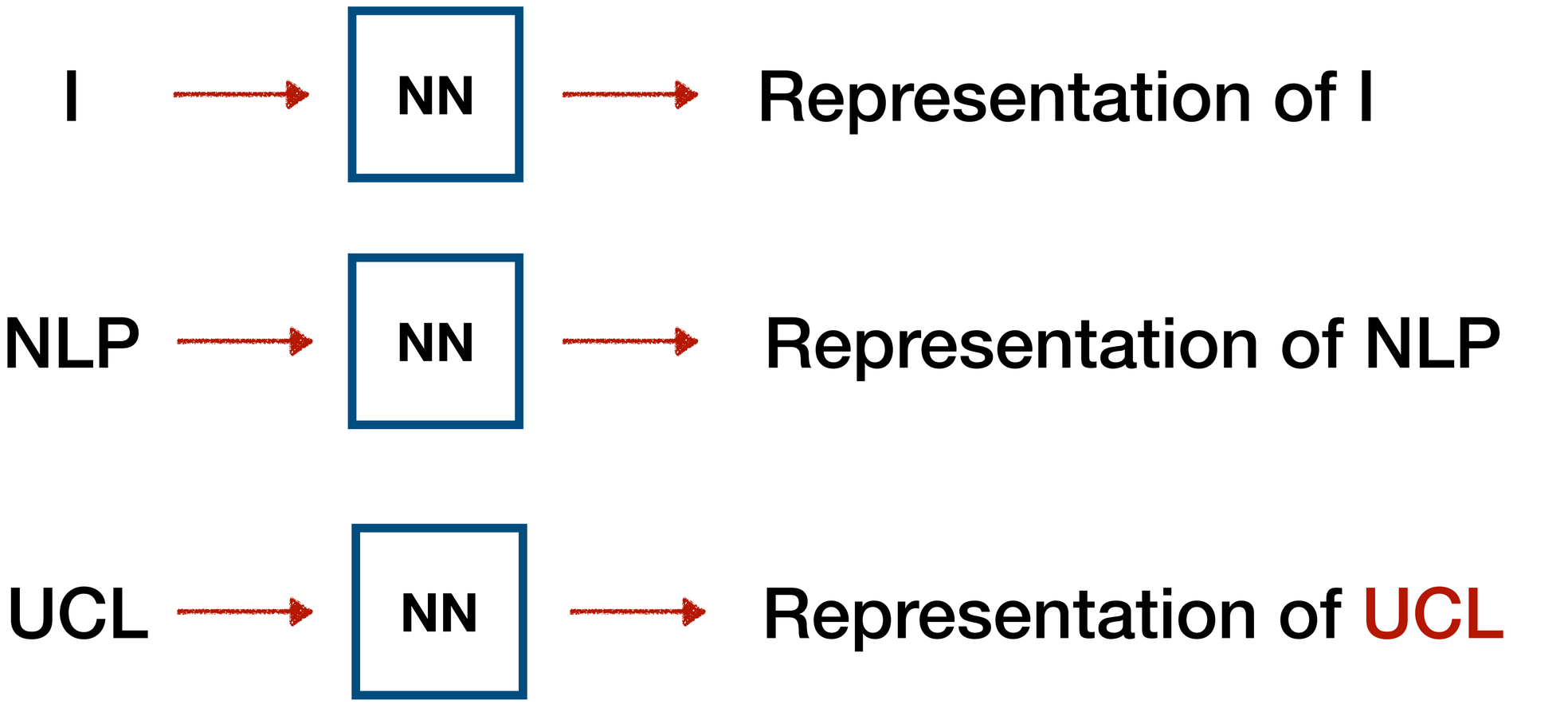
Idea

Use the same weight matrix to process information at different time steps?

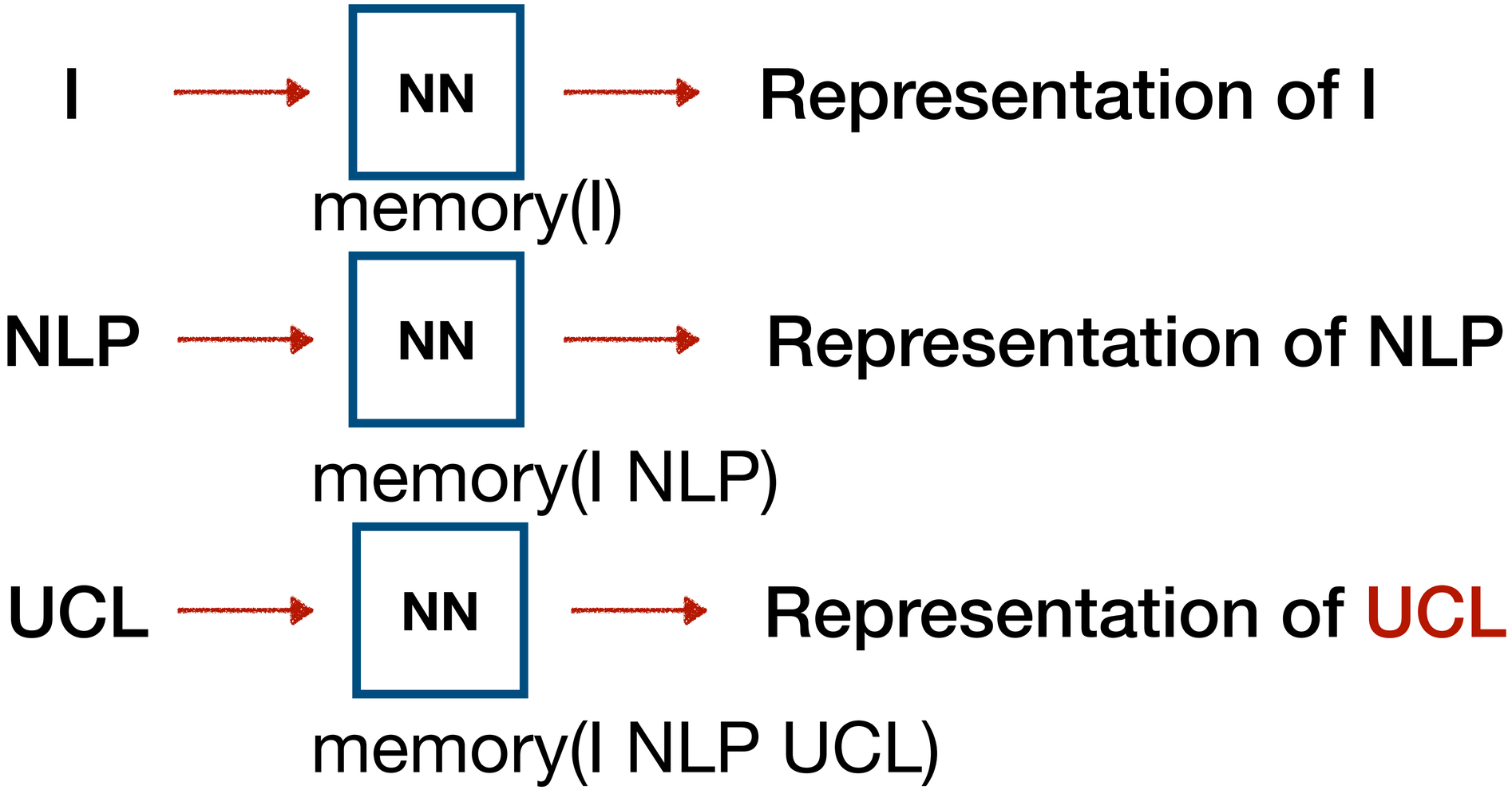
A simple idea



Problem


I
NN
Representation of I
NLP
NN
Representation of NLP
UCL
NN
Representation of UCL
No information about I/NLP

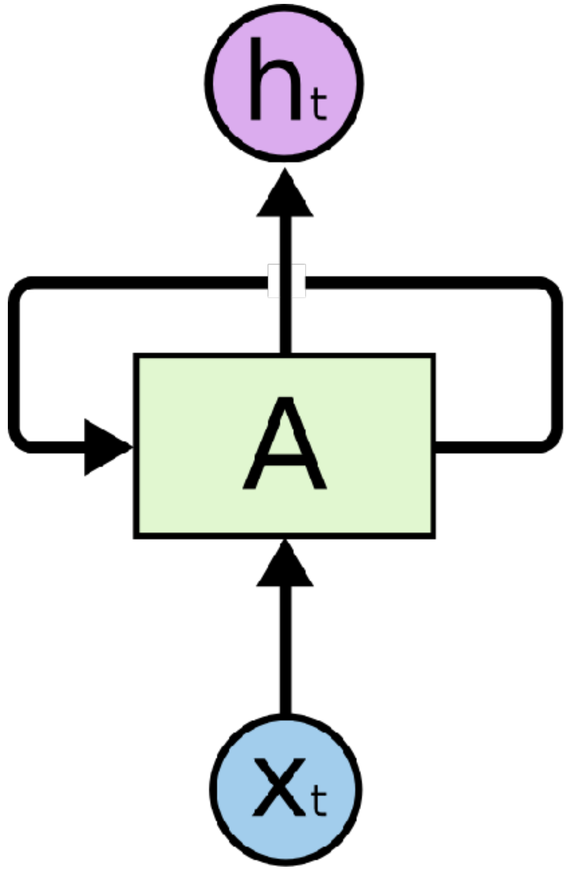
Fix


I
NN
Representation of I
memory(I)
NLP
NN
Representation of NLP
memory(I NLP)
UCL
NN
Representation of UCL
memory(I NLP UCL)

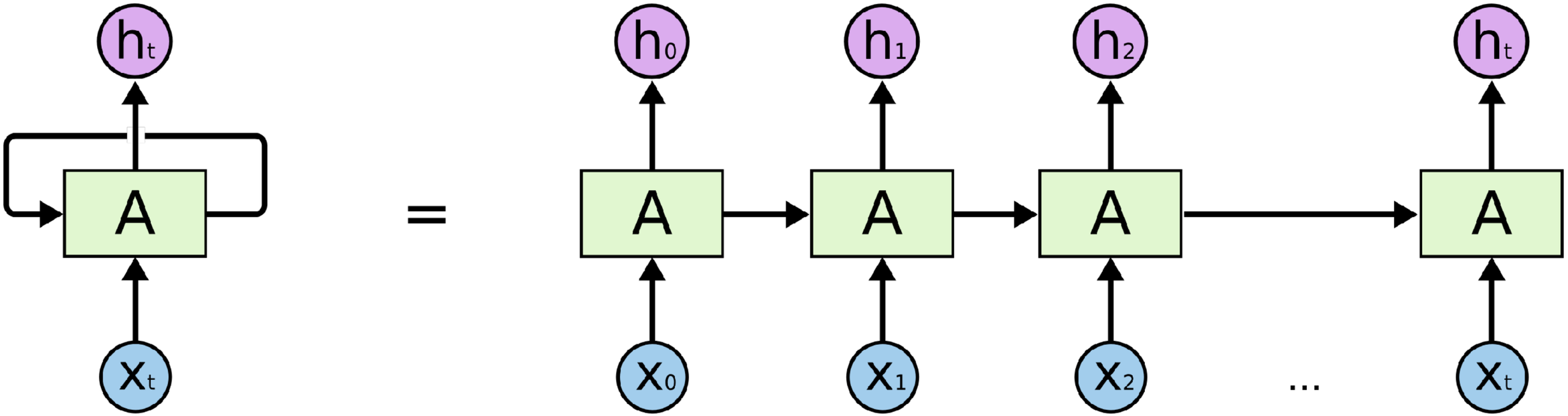
Recurrent Neural Network

- Variable-length context: Can process sequences of any length
- Hidden state memory: Maintains information from all previous words
- Parameter sharing: Same weights applied at each time step

Recurrent Neural Network



Recurrent Neural Network



Recurrent Neural Network

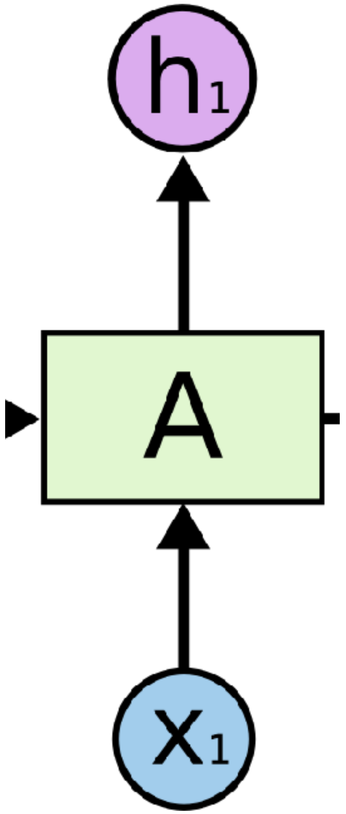
- At each time step t, RNN takes:
- Current word embedding: $\mathbf{x}_t$
- Previous hidden state: $\mathbf{h}_{t-1}$ (memory from past)
- And produces:
- New hidden state: $\mathbf{h}_t$ (updated memory)

Recurrent Neural Network

$$h_t = \tanh\left(W_h\, h_{t-1} + W_x\, x_t + b\right)$$

- $\mathbf{W}_h$: hidden-to-hidden weight matrix (size: hidden_size × hidden_size)
- $\mathbf{W}_x$: input-to-hidden weight matrix (size: hidden_size × embedding_dim)
- $\mathbf{b}$: bias vector
- $\tanh$: activation function (keeps values between -1 and 1)

Recurrent Neural Network



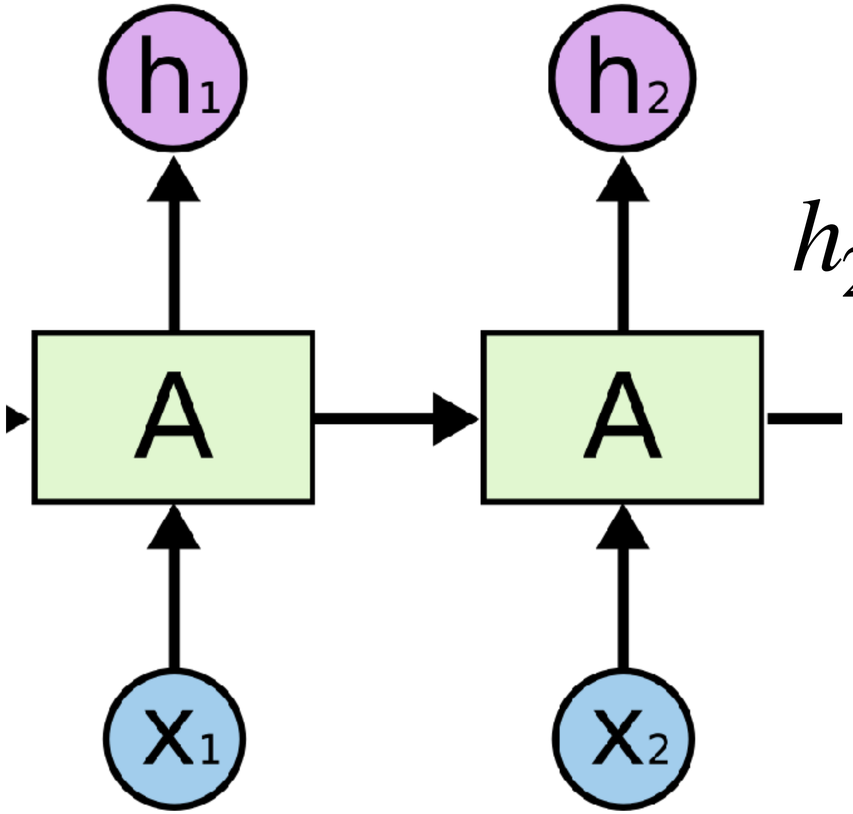
I

new word information “I”

$$h_1 = \tanh\left(W_h h_0 + W_x E["\text{I}"] + b\right)$$

previous information<s>

Recurrent Neural Network

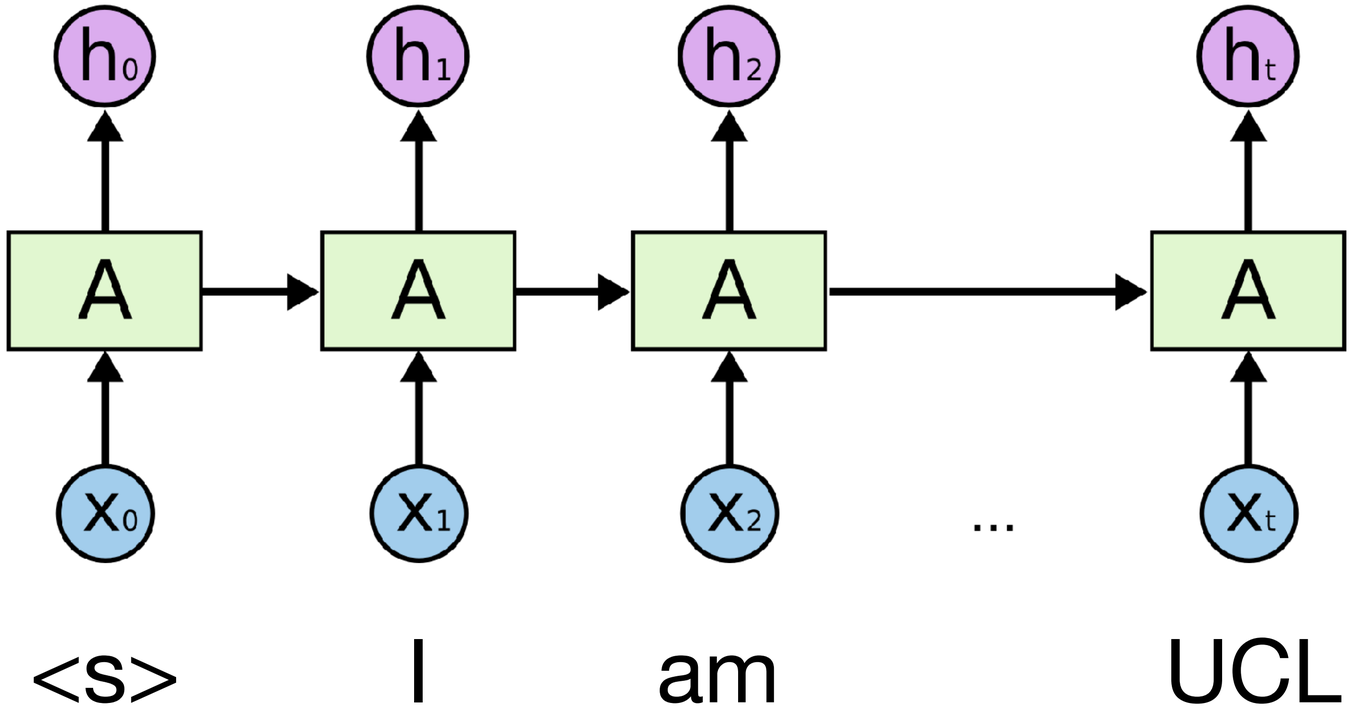


$$h_2 = \tanh\left(W_h h_1 + W_x E[\text{"am"}] + b\right)$$

previous information <s> & I

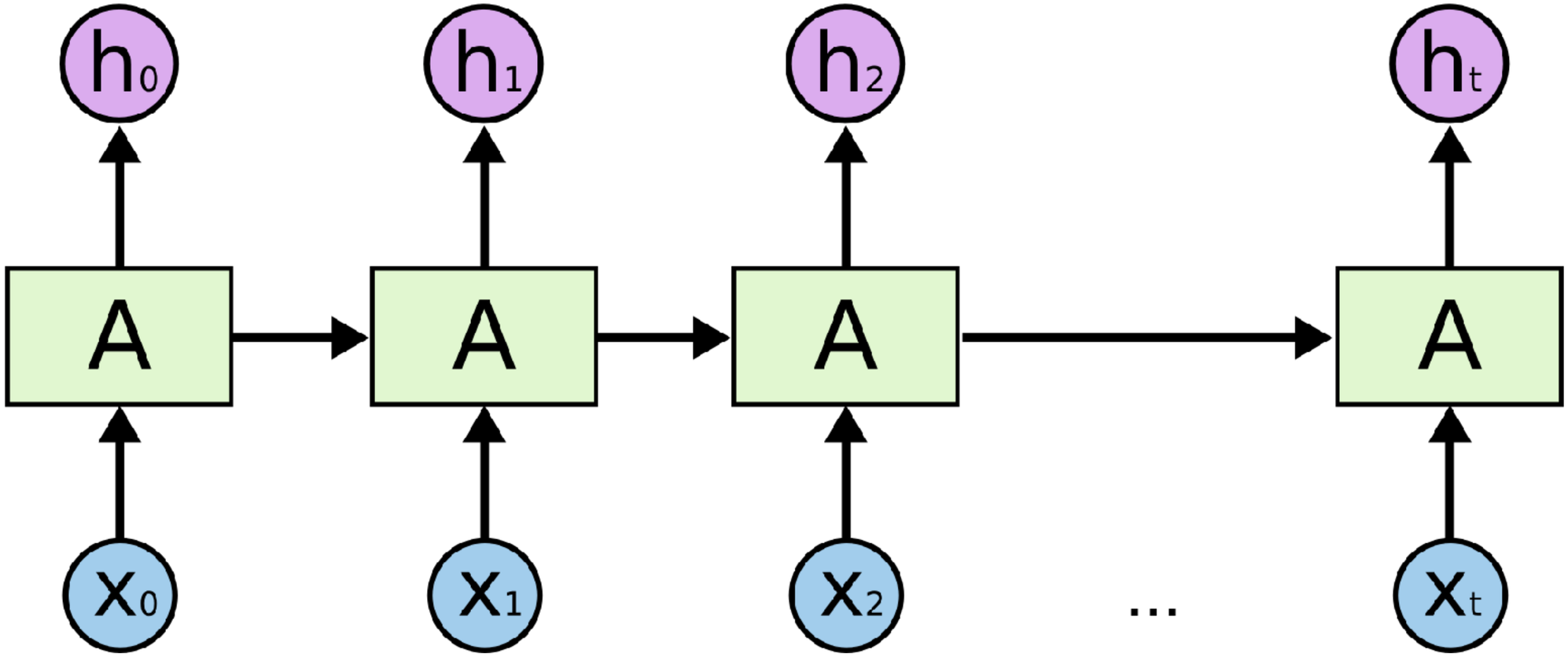
Recurrent Neural Network

$$h_t = \tanh\left(W_h h_{t-1} + W_x x_t + b\right)$$



Same weights (W_h, W_x) for each step.

RNN Language Model



RNN Language Model

After computing hidden state h_t, we can predict the next word

$$\mathbf{z} = \mathbf{W}_o\mathbf{h} + \mathbf{b}_o$$

$\mathbf{W}_o$: output weight matrix (size: V × hidden_size)

$\mathbf{b}_o$: output bias vector (size: V)

$\mathbf{z}$: logits (raw scores for each word in vocabulary)

Output Layer: Computing Word Probabilities

Normalize logits to a valid distribution

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

$$\hat{y}_i = P(w_i \,|\, \text{context}) = \frac{\exp(z_i)}{\sum_j^V \exp(z_j)}$$

Result: Probability distribution over entire vocabulary

Output Layer: Computing Word Probabilities

Time	Input	Hidden state	Prediction Target
1	I	h_1	am (0.42)
2	am	h_2	studying (0.35)
3	studying	h_3	NLP (0.28)
4	NLP	h_4	at (0.31)

$$h_3 \approx \alpha \text{Info("I")} + \beta \text{Info("am")} + \gamma \text{Info("studying")}$$

Training Objective

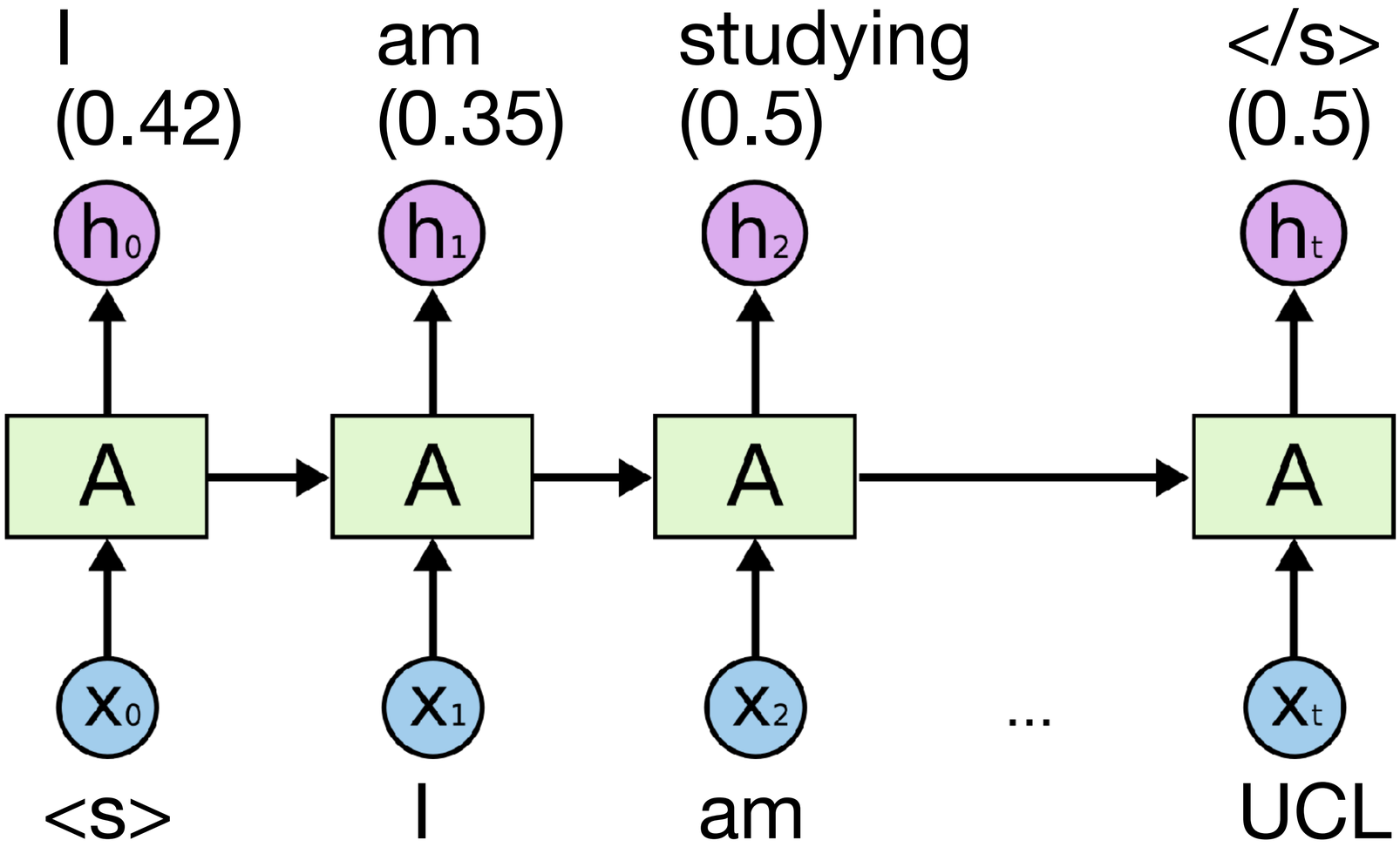
$$\text{Loss per step} = -\sum_{i}^{V} y_i log(\hat{y}_i) = -log(\hat{y}_{\text{target}})$$

$$\text{Loss} = -\frac{1}{T}\sum_{t}^{T} log(\hat{y}_{t,\text{target}})$$

Training Objective

$$L_0 = -log(\hat{y}["I"]) = -log(0.42) = 0.87$$

$$L_1 = -log(\hat{y}["am"]) = -log(0.35) = 1.05$$



Training Objective

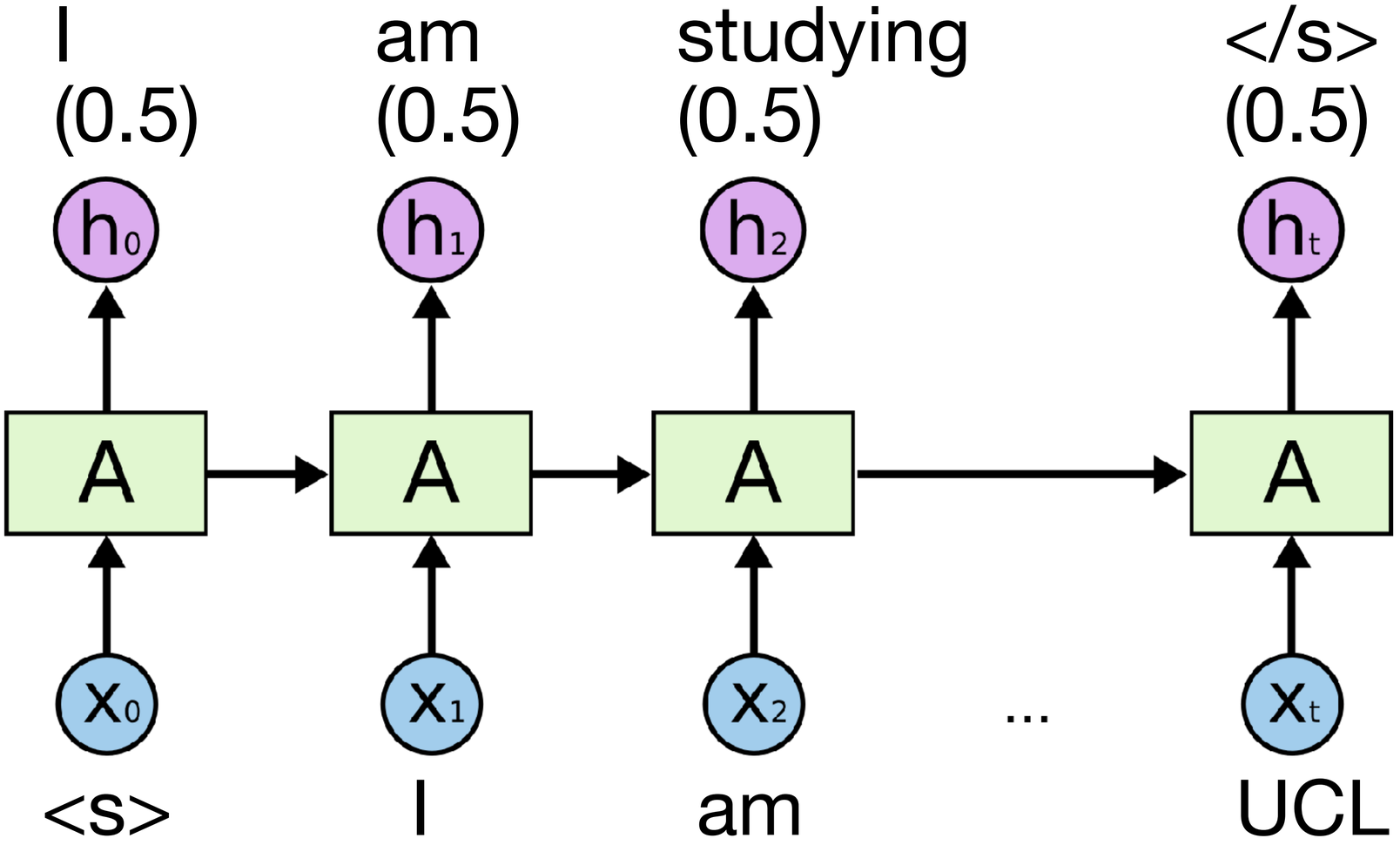
Backpropagation Through Time (BPTT):

- Gradients flow backward through time steps
- Update same weights (W_h, W_x, W_o) using accumulated gradients
- Challenge: Vanishing/exploding gradients over long sequences

RNN Language Model

- We isolate the shape of weight from the sequence length. Model weights (W_h, W_x) are independent of sequence length, making it possible to process arbitrary length of document.
- We can use the hidden state to store information that is lots of steps back, which makes long context modelling possible.

Decoder-only LM



RNN LM and GPT-2

- Autoregressive: Predict one token at a time, left-to-right
- Causal: Only use previous context, not future tokens
- Same objective: Minimize $-\log(\hat{y}_{\text{target}})$ at each position
- Only difference: recurrent versus self-attention

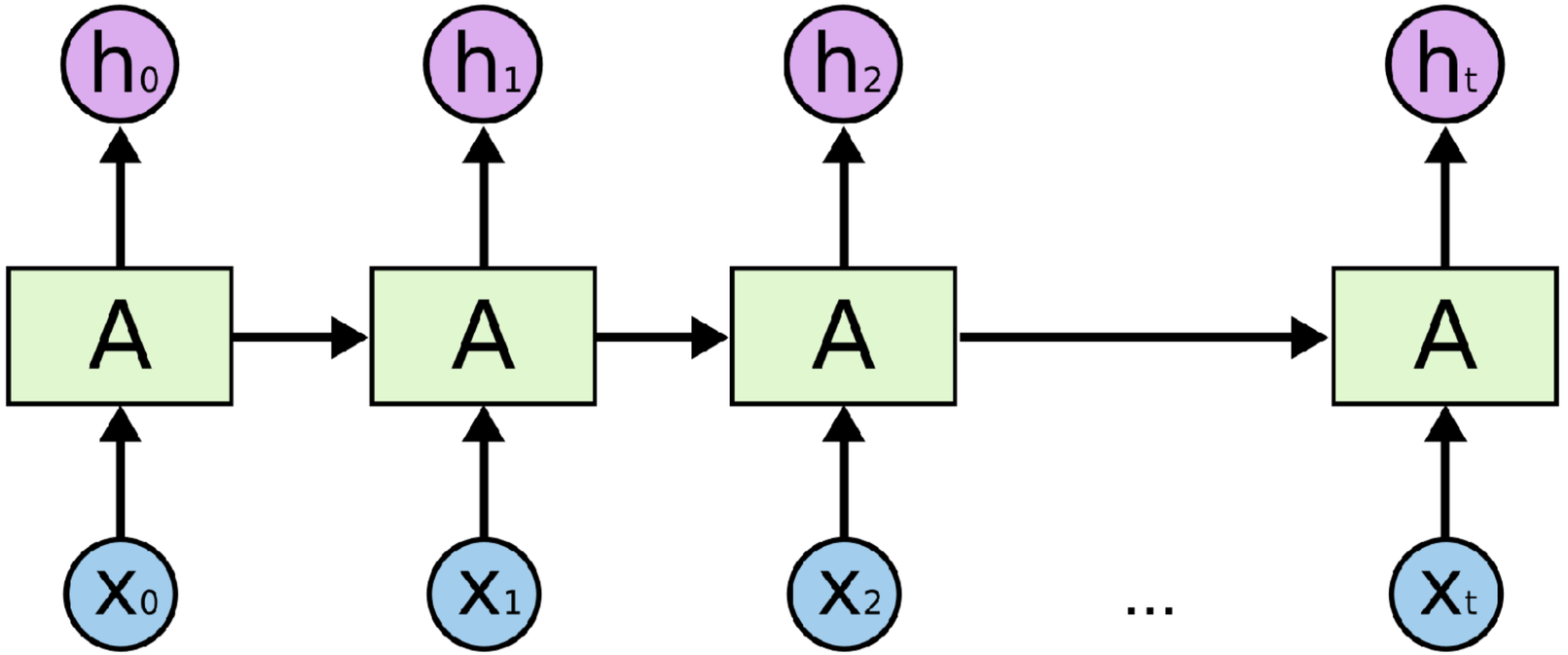
Limitation of RNN LM

I am a student

4 tokens

Je sui étudiant

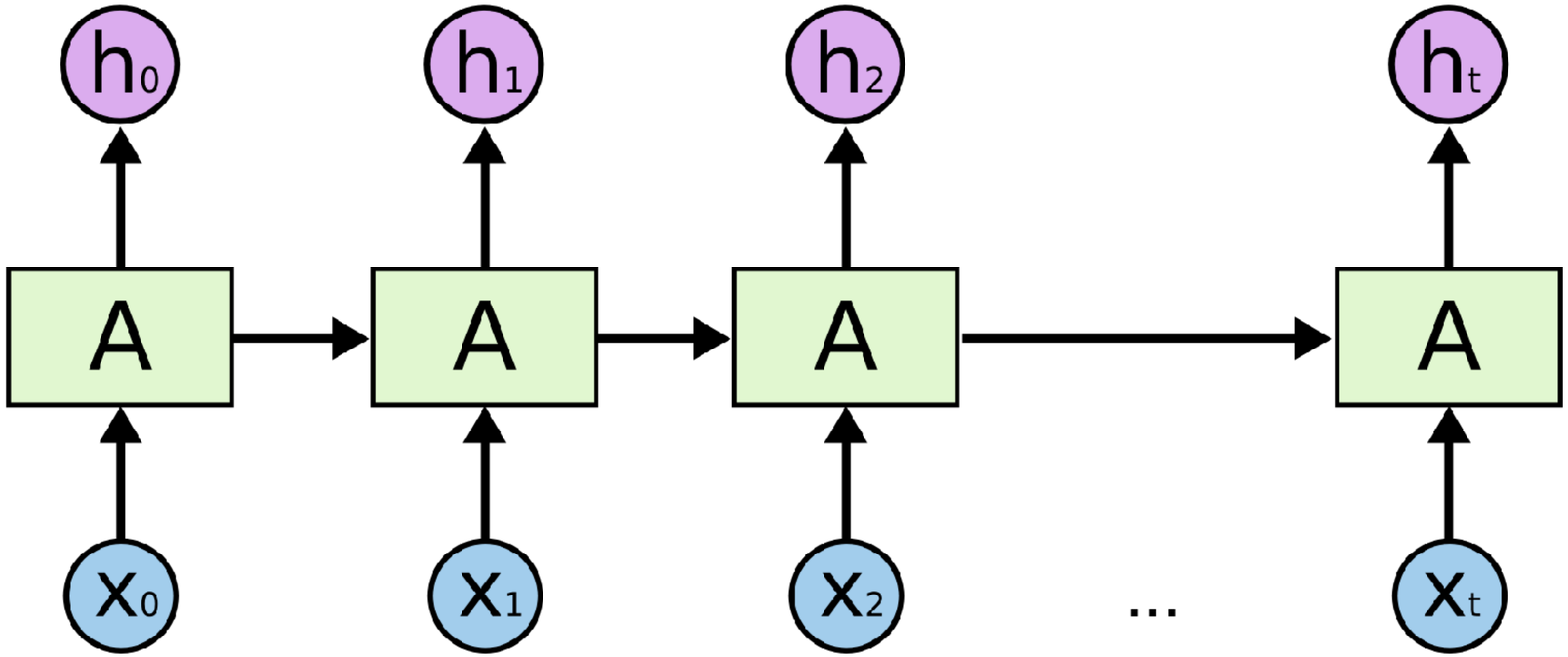
3 tokens



Limitation of RNN LM

Modern solution:

<s> English: I am a student. French: Je sui étudiant </s>



Translation Task

I am a student

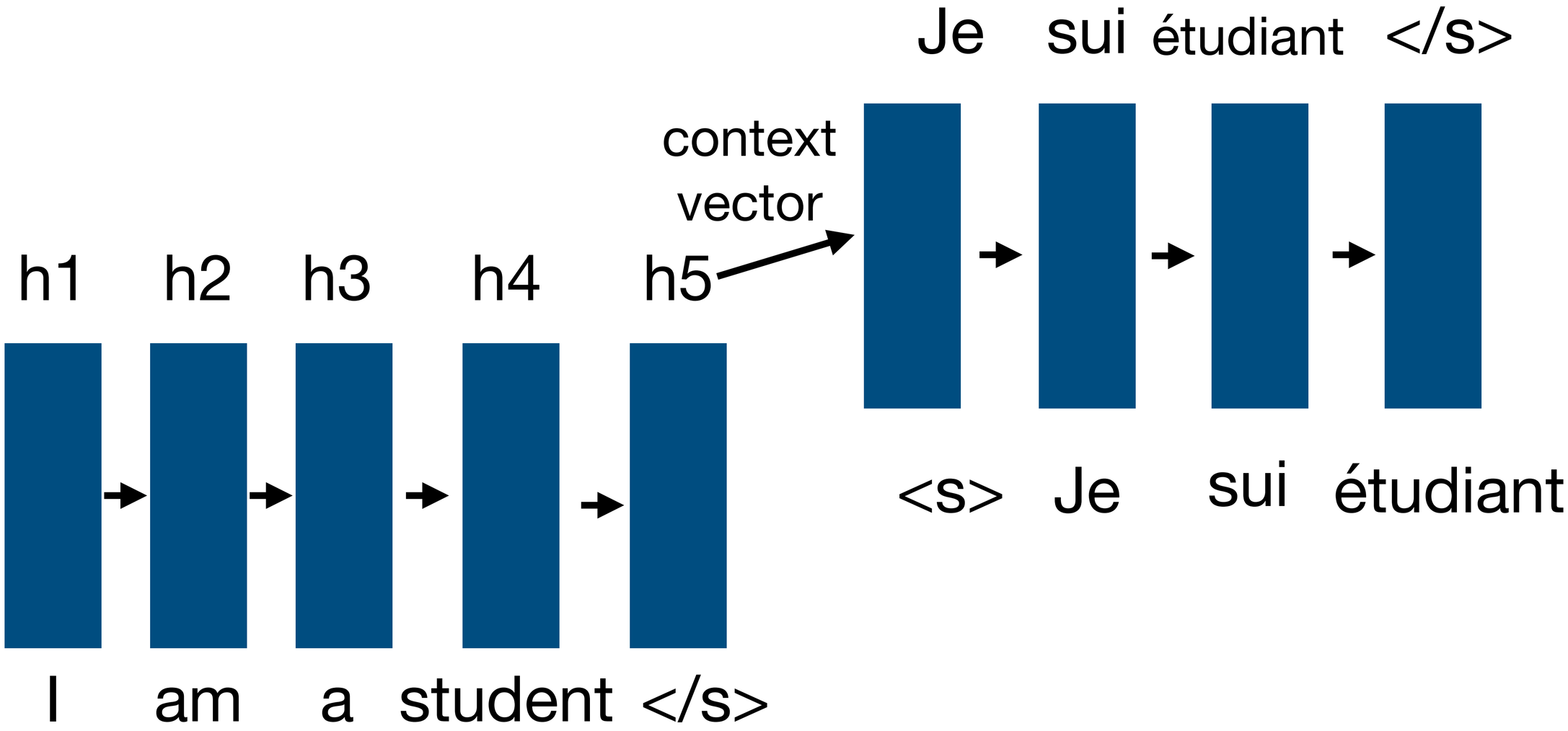
Je sui étudiant

- Different languages, different input/output length
- Similar semantics for input and output

Sequence-to-sequence model

- Encoder RNN: Process input sequence, compress into context vector
- Decoder RNN: Generate output sequence from context vector

Sequence-to-sequence model



Sequence-to-sequence model

Encoder RNN (processes source sentence):

- At each time step t in the source:

$$h_t^{enc} = \tanh\left(W_h^{enc}\, h_{t-1}^{enc} + W_x^{enc}\, x_t + b^{enc}\right)$$

Context vector (final encoder state):

$$c = h_T^{enc}$$

T is the length of source sentence

Sequence-to-sequence model

Decoder RNN (generates target sentence):

Initialisation of decoder RNN state $s_0 = c$

At each time step t in the target:

$$s_t = \tanh\left(W_h^{dec} \, s_{t-1}^{dec} + W_x^{dec} \, y_{t-1} + b^{dec} \right)$$

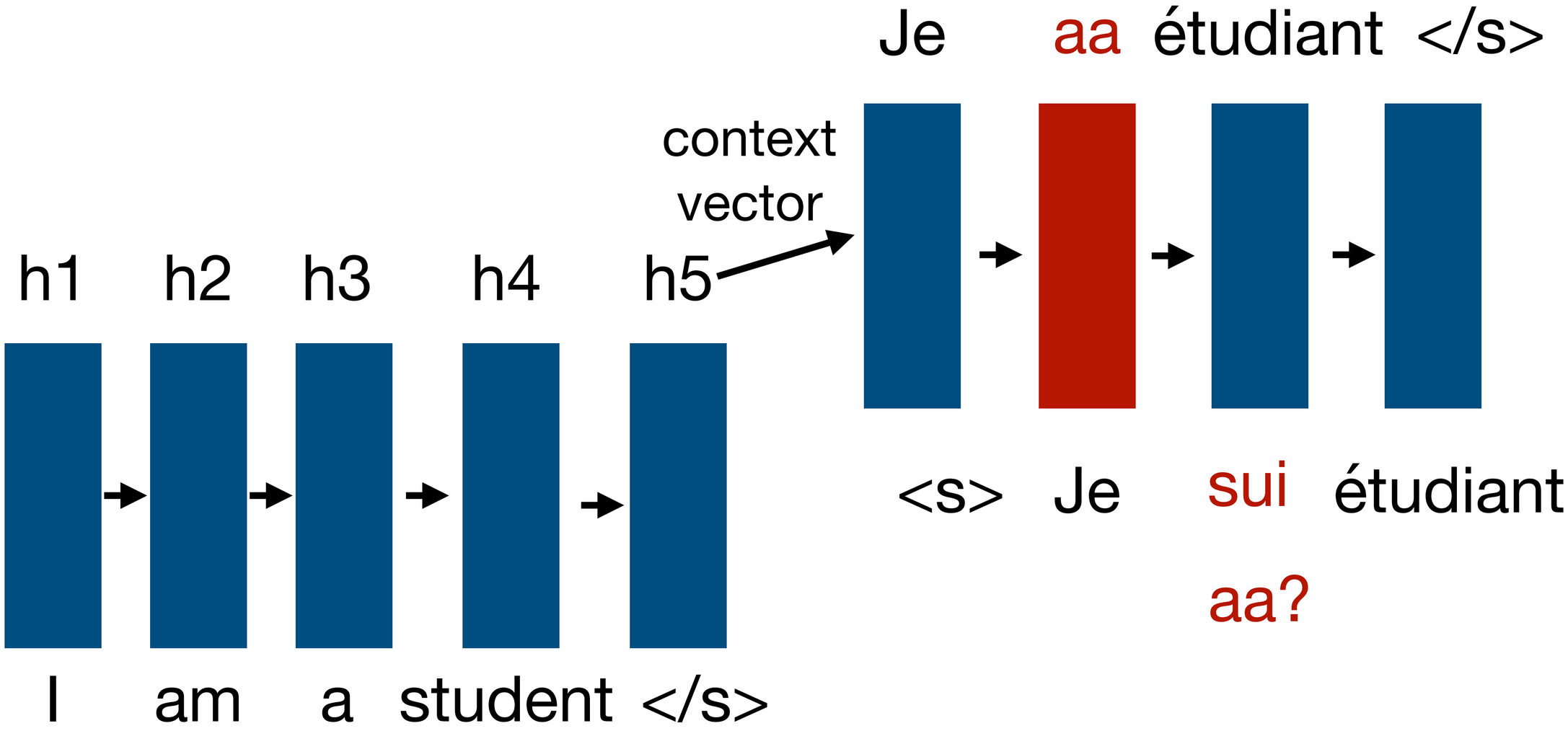
Output distribution:

$$P(y_t \,|\, y_1, \ldots, y_{t-1}, \text{source}) = \text{softmax}(W_o \cdot s_t + b_o)$$

Sequence-to-sequence model

- Separate parameters: Encoder (W^{enc}) and Decoder (W^{dec}) have different weights
- Decoder input: Previous target word y_{t-1} (during training: ground truth; during inference: predicted word)
- Context c flows into decoder initialization or at each step

Training Seq2seq model



Training Seq2seq model

- Training (Teacher Forcing): At each decoder step, use the true previous word as input.

Step 1: Input <s> → s_1 → predict "Je" $Loss_1 = -\log P(\text{"Je"}|s_1)$

Step 2: Input "Je" → s_2 → predict "suis" $Loss_2 = -\log P(\text{"suis"}|s_2)$

…

Step 4: Input "étudiant" → s_4 → predict <EOS> $Loss_4 = -\log P(\text{<EOS>}|\ s_4)$

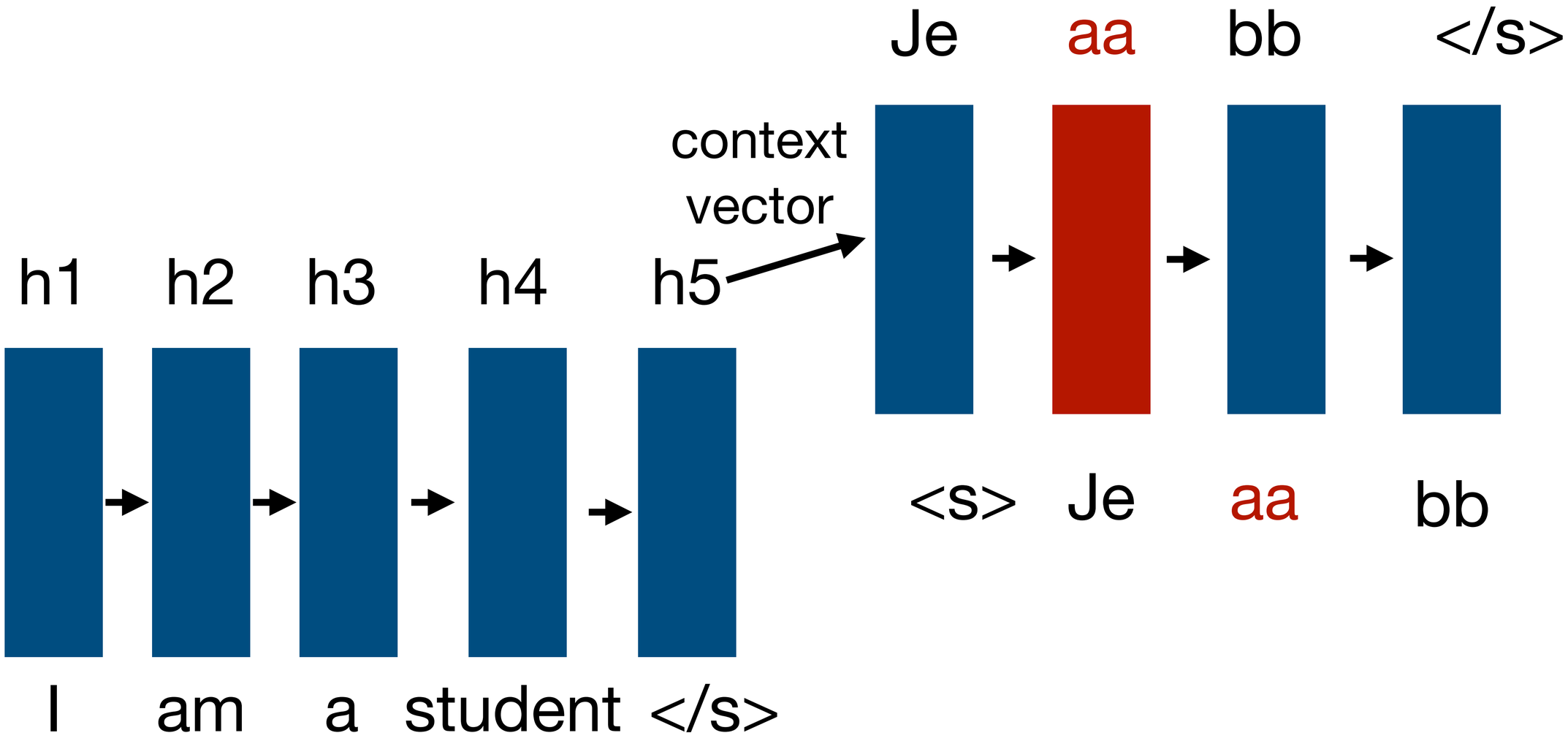
Training Seq2seq model

- Training (Teacher Forcing): At each decoder step, use the true previous word as input.

$$\text{Loss} = -\frac{1}{T}\sum_{t=1}^{T} \log P(y_t^* \mid y_1^*, \ldots, y_{t-1}^*, \text{source})$$

where y_t^* is the true target word

Inference Seq2seq model



Model learns when to stop

- There is no predefined sequence length.
- Model generates a special <EOS> / stop token.
- Generation ends when stop token has highest probability.

Many NLP tasks are seq2seq

- Text summarisation

Original Text (truncated): lagos, nigeria (cnn) a day after winning nigeria's presidency, ***muhammadu buhari*** told cnn's christiane amanpour that **he plans to aggressively fight corruption that has long plagued nigeria** and go after the root of the nation's unrest. *buhari* said he'll "rapidly give attention" to curbing violence in the northeast part of nigeria, where the terrorist group boko haram operates. by cooperating with neighboring nations chad, cameroon and niger, **he said his administration is confident it will be able to thwart criminals** and others contributing to nigeria's instability. for the first time in nigeria's history, the opposition defeated the ruling party in democratic elections. *buhari* defeated incumbent goodluck jonathan by about 2 million votes, according to nigeria's independent national electoral commission. **the win comes after a long history of military rule, coups and botched attempts at democracy in africa's most populous nation.**

Baseline Seq2Seq + Attention: **UNK UNK** says his administration is confident it will be able to **destabilize nigeria's economy**. **UNK** says his administration is confident it will be able to thwart criminals and other **nigerians**. **he says the country has long nigeria and nigeria's economy.**

Seq2Seq Philosophy: Learning What's Important

Task: Text Summarization - Capturing Key Information

- Source document: Very long, rich information (hundreds of words)
- Summary: Short, only the most important points (tens of words)
- Question: How do we decide what's important?

Seq2Seq Philosophy: Learning What's Important

Classical method (Feature Engineering):

- TF-IDF: Extract sentences with high-scoring keywords
- Position-based: First/last sentences are important
- Keyword extraction: Look for specific indicator words

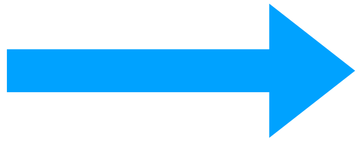
Learning from data

Encoder: Compress entire document into context vector c, context vector contains ALL information from source

Decoder: Learn to extract key information from c

- Trained on (document, summary) pairs
- Model learns what's important from the data
- No manual feature engineering needed!

Seq2seq to Seq-concat-seq

I am a student  Je sui étudiant

Modern solution:

<s> English: I am a student. French: Je sui étudiant </s>

Learning from data

FineWeb Pretrain Data: Translation Pairs

Old English : bacan = ”baker”
Proto-Germanic : bakanan = ”bake”
Old Norse : baka = ”bake”
Middle Dutch : backen = ”bake”
Old High German : bahhan = ”bake”
German : backen = ”bake”
PIE : bheg- ”to warm, roast, bake”
Greek : phogein ”to roast”
Root : bhe- ”to warm”

From seq2seq to ChatGPT

- If we frame seq2seq as a language modeling task:
- Input + Output = one long sequence
- Train to predict next token (same as RNN-LM)
- No need for separate encoder and decoder
- This insight led to models like ChatGPT.

Thank you!