FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

NI-NLM – Lecture 1

Introduction to NLP with Neural Networks

Zdeněk Kasner

17 Feb 2026

Course introduction

The teacher

Zdeněk Kasner

Postdoctoral researcher at **ÚFAL, MFF UK**

- Research on LLMs: controllable text generation, evaluation, reasoning.

My academic path

- **FIT CTU**: Computer Science (Bc.)
- **FEL CTU**: Artificial Intelligence (Ing.)
- **MFF UK**: Computational Linguistics (Ph.D.)

What to expect from me?

- I am also teaching **NPFL140 – Large Language Models** **at MFF UK** together with my colleagues (3rd year in a row).
  - Our course will be different (CZ, less people → more interactivity during lectures, tutorials with paper presentations).
- The **first year** I am teaching this course here → there may be some hiccups.
- My experience is in academic research → I am curious to learn what you know better!

> ⚠️ **Thesis supervision**
>
> Sorry, I cannot supervise your thesis – I am at full capacity now.

What to expect from this course?

Me: talking on LLMs

I will **present the topic**. The slides (in English) and other materials will be available on Course Pages.

Us: discussing LLMs

I want **you to discuss and reason** about the things we learn (test-time scaling! 😉)

You: working on the project

Semestral project is the major part of the workload in this course → consultations during the tutorials.

You: presenting

You can get extra points for the final test by **presenting a paper or your project** (→ tutorials).

About the course

Learning objectives

1. Help you understand **how LLMs work** on the architectural level.
2. Teach you **how to handle LLMs** in practice.
3. Teach you **how to work with AI/NLP scientific literature**.
4. Teach you **how to think about the broader context** of using LLMs.

What is expected from you?

- You should be familiar with **Python + basics of deep learning**.
- You should be **active during the lectures**.
- You should be **enthusiastic about LLMs**! (at least somewhat 😎)

Course requirements

How to get the assessment

- You need to **work on the semestral project** and **submit the report** till 31 May 2026.
- You also need achieve **50/100 points in the final test**.
  - In-person during the last lecture (12 May 2026) + one extra term during the exam period.

Course logistics

- **1 lecture per week + 1 tutorial every two weeks (starting today)**
  - Attendance not mandatory, but highly recommended.

Lectures – T9:347

Outline:

1. Intro to NLP with neural networks
2. Transformer architecture
3. Training language models
4. Text generation, decoding algorithms
5. Data and evaluation
6. Chain-of-thought, reasoning models
7. RAG, tool calling, agents
8. Efficiency: quantization, LoRA
9. Interpretability
10. Multimodal models
11. Applications, ethics, legal aspects
12. Final test

~ **60 minutes** of me talking, **30 minutes** of discussions and other activities

Tutorials – T9:350

Today: Semestral project, how to compute.

Next five tutorials: Paper presentations + consultations

Paper presentations

- List of recommended papers (or propose yourselves)
- **15 min presentation** → 5 points for the final test (only once per semester).
- Sign up in the spreadsheet.

Consultations

- Help with the semestral project, **voluntary**.
- After the paper presentations are over.
- If there are too many papers, we can consult after the tutorial.

Questions for you

Question

- What is your LLM expertise (if any)?
- Why have you signed up for the course?
- What do you expect to learn from this course?

Questions for you

Question

Could you explain the following terms to a friend?

- vector
- machine learning
- neural networks
- embedding
- language modeling
- Transformer
- LLM
- GPT
- finetuning
- RLHF
- RAG
- AI agent

How to represent language?

What is language?

Are these languages?

```
The quick brown fox jumps over the lazy dog.
```

```
ᑐᓴᕐᓂᐊᑭᓐᓇᖏᑦᑐᐊᔪᓂᖓ
```

```
print("Hello, world!")
```

```
+++++++[>++++++++++<-]>+.
```

```
(())()(())
```

```
efhiew nwjfksahfiahf asdkfa sfhasiofh
```

What is language?

Language is a **sequence of symbols** that conveys meaning.

It is discrete

- Made of distinct units (characters, words).

→ Sequence $w_1, w_2, ..., w_T$ from vocabulary V.

It is compositional

→ Infinite sentences from finite vocabulary.

- Meaning comes both from the units (words) and their structure.

How to represent language?

Question

Suggest 2-3 ways **how to represent a language sequence** in a computer.

- How do you create the representation?
- What are its advantages and disadvantages?
- Is it suitable as an input for a neural network? Why?

Text as byte sequences

Text strings are represented as **byte sequences**.

Various encodings: **ASCII** (8 bits), **UTF-8**: variable-length, ...

Char	Dec	Bin
A	65	1000001
B	66	1000010
C	67	1000011
...	...	...
a	97	1100001
b	98	1100010
c	99	1100011

Disadvantages:

- Misleading dependencies:
  - Character ids are arbitrary constants
  - # of characters in the word
  - ...
- No meaningful relationship between related words.

One-hot encoding

Idea

Let's get rid of any dependencies!

One-hot encoding: represent each word as a **binary vector** with a single 1.

a	1	0	0	0	0	...	0
aardvark	0	1	0	0	0	...	0
abandon	0	0	1	0	0	...	0
...	...	...	...	...	...	...	...
zyzzyva	0	0	0	0	0	...	1

Each word w_i gets a vector $\boldsymbol{v} \in \{0, 1\}^{|V|}$ where $|V|$ is the vocabulary size.

Problems with one-hot encoding

- **Large vectors:** natural languages may contain 100,000+ words → each vector needs to have 100,000+ dimensions.
- **Extremely sparse:** most entries are 0.
- **No similarity** between individual words:

$$\boldsymbol{v}_{\text{Czechia}} \cdot \boldsymbol{v}_{\text{Slovakia}} = 0$$

$$\boldsymbol{v}_{\text{Czechia}} \cdot \boldsymbol{v}_{\text{computer}} = 0$$

→ Despite all this, it was the standard **input representation for the early NLP** (Bayes classifiers, logistic regression, early feedforward neural networks, …)

Representing meaning

Question

What is meaning? Can you think of a definition that we can formalize, extract and represent?

- Referential semantics, prototype theories, use-based theories (Wittgenstein), …
- **Distributional semantics**: Words that appear in similar contexts have similar meanings
  - “You shall know a word by the company it keeps.” (J. R. Firth, 1957)

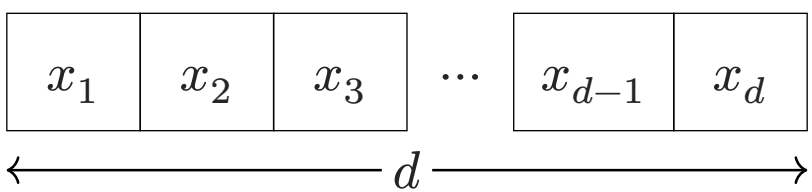
Word embeddings

Word embeddings

Word embedding: dense, low-dimensional vector representation of a word:

$$\mathrm{emb} : V \to \mathbb{R}^d$$

where $d \ll |V|$ (think of d between 300 to 10k, V between 50k to 200k).



Our goal: vectors of words with similar meaning are "similar" → close in the d-dimensional vector space.

How to compute similarity of vectors?

Cosine similarity $\approx$ angle between the vectors

$$\mathrm{sim}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{\|\boldsymbol{v}_1\| \, \|\boldsymbol{v}_2\|} \in \langle -1, 1 \rangle$$

Toy example with $d = 3$:

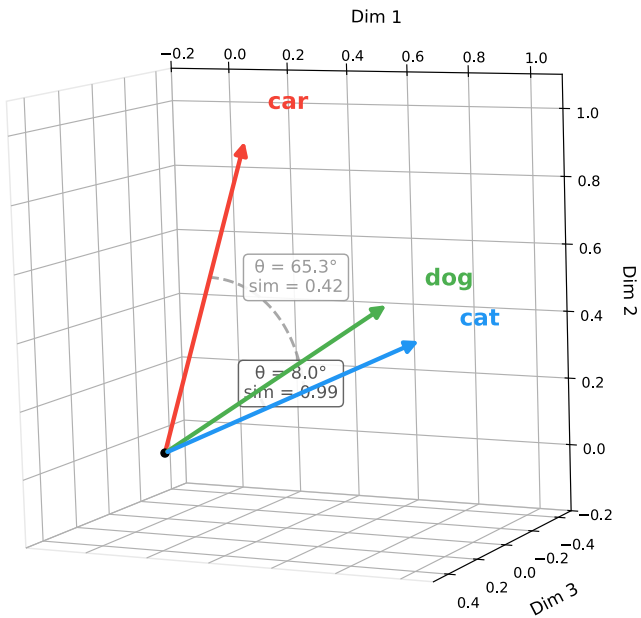
$$\boldsymbol{v}_{\text{cat}} = [0.9, 0.4, 0.1]$$

$$\boldsymbol{v}_{\text{dog}} = [0.8, 0.5, 0.1]$$

$$\boldsymbol{v}_{\text{car}} = [0.1, 0.9, -0.4],$$

$$\mathrm{sim}(\boldsymbol{v}_{\text{cat}}, \boldsymbol{v}_{\text{dog}}) \approx 0.99$$

$$\mathrm{sim}(\boldsymbol{v}_{\text{cat}}, \boldsymbol{v}_{\text{car}}) \approx 0.42$$



Cosine similarity – detail

Detailed computation:

For $\boldsymbol{v}_{\text{cat}}$ and $\boldsymbol{v}_{\text{dog}}$:

$$\boldsymbol{v}_{\text{cat}} \cdot \boldsymbol{v}_{\text{dog}} = 0.9 \times 0.8 + 0.4 \times 0.5 + 0.1 \times 0.1$$

$$= 0.72 + 0.20 + 0.01 = 0.93$$

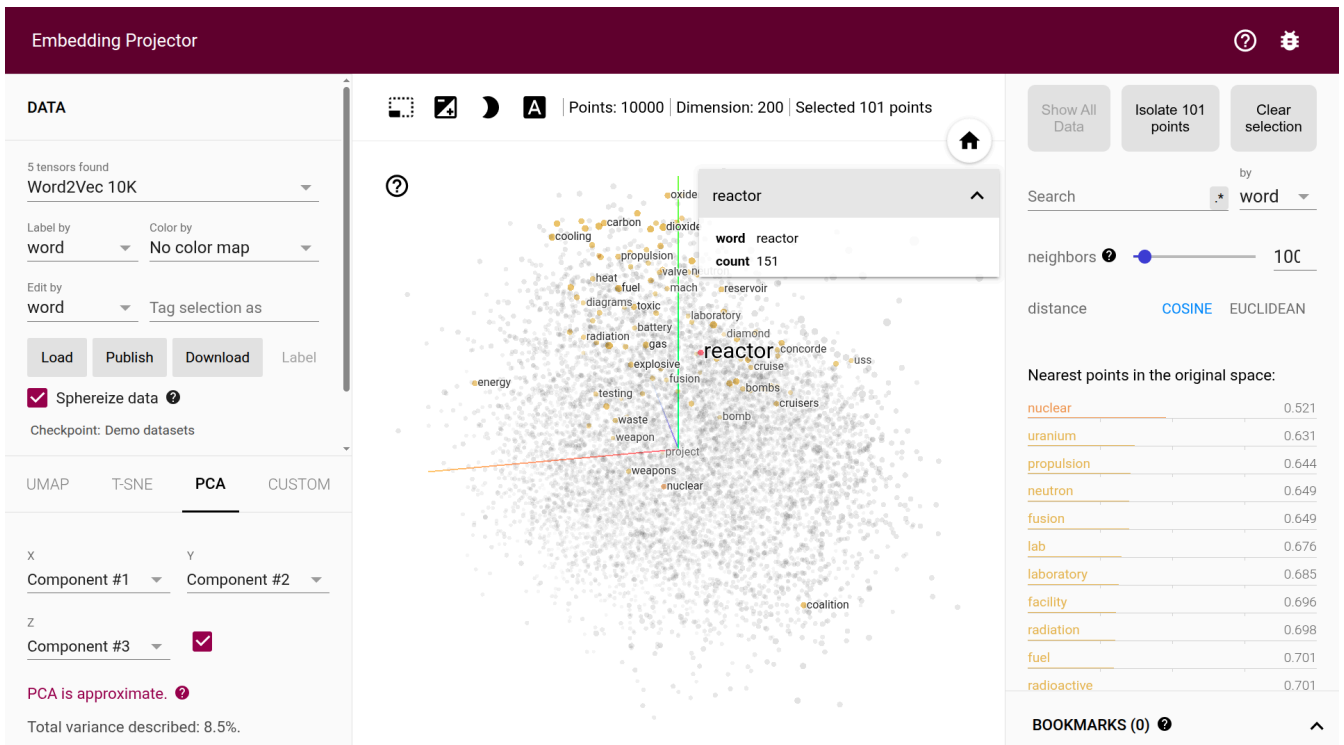
$$\|\boldsymbol{v}_{\text{cat}}\| = \sqrt{0.9^2 + 0.4^2 + 0.1^2} = \sqrt{0.81 + 0.16 + 0.01} = \sqrt{0.98} \approx 0.99$$

$$\|\boldsymbol{v}_{\text{dog}}\| = \sqrt{0.8^2 + 0.5^2 + 0.1^2} = \sqrt{0.64 + 0.25 + 0.01} = \sqrt{0.90} \approx 0.95$$

$$\text{sim}\big(\boldsymbol{v}_{\text{cat}}, \boldsymbol{v}_{\text{dog}}\big) = \frac{0.93}{0.99 \times 0.95} \approx \frac{0.93}{0.94} \approx 0.99$$

Trained word embeddings

👉 https://projector.tensorflow.org

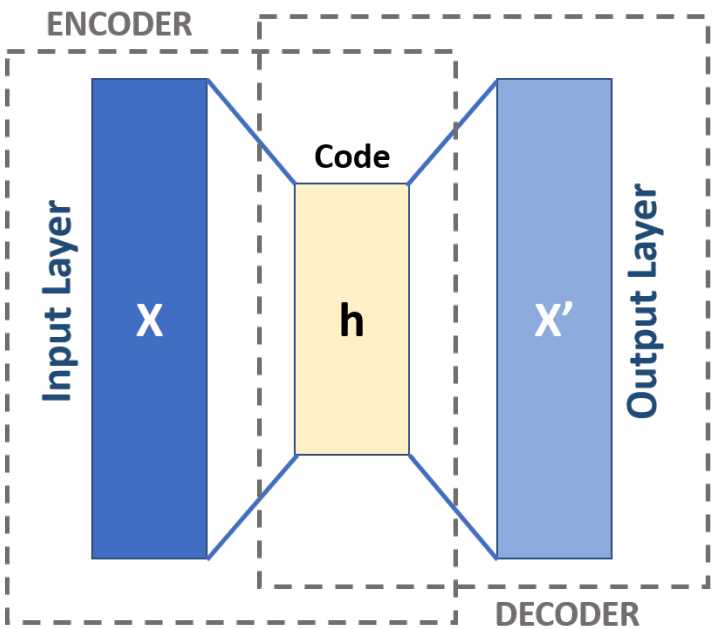


How to get word embeddings?

Question

How to train word embeddings?

Hint: can you recognize the image on the right?



source: Wikipedia

Autoencoder:

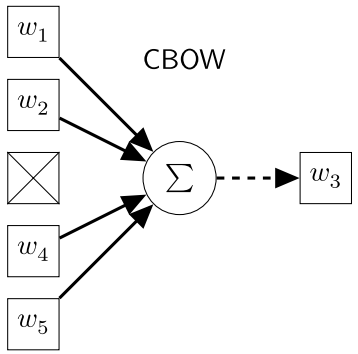
- Neural network acts as a dimensional bottleneck.
- During training, it needs to learn an efficient representation of the object it encodes.

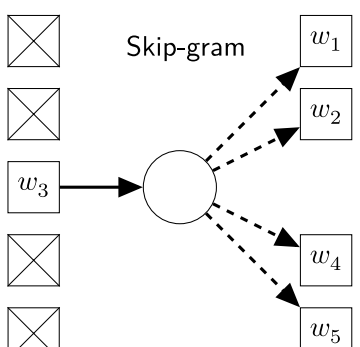
source: Mikolov et al. 2013

Word embeddings – training objectives

Two variants proposed for **Word2Vec** (Mikolov et al., 2013)

- Predict a *target word* from its *context words* (**CBOW**).
- Predict *context words* from a *target word* (**skip-gram**).





On the **input**, the words are encoded using one-hot encoding.

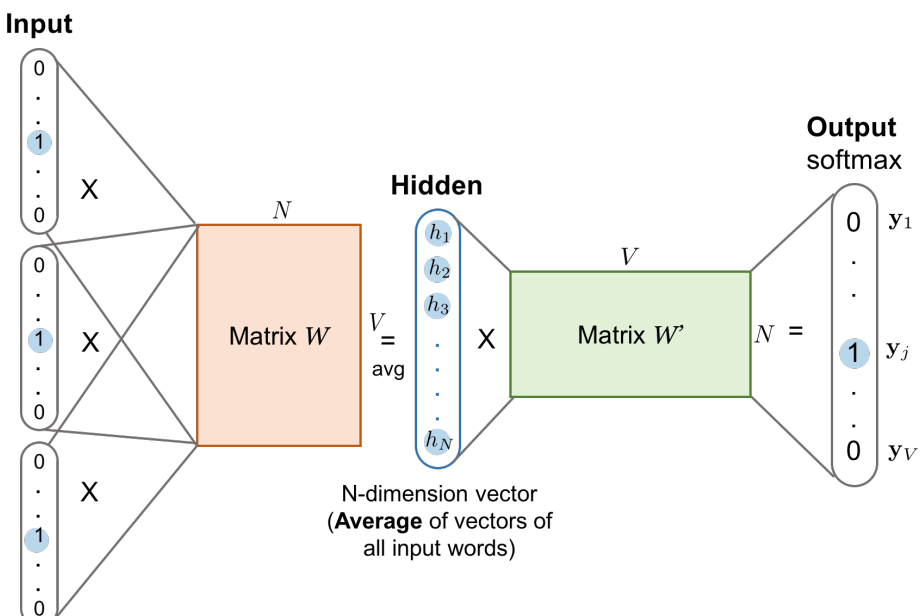
On the **output**, we have a V-dimensional vector: a probability score for each word in vocabulary.

source: NPFL129

Word2Vec – CBOW

source: Lil'Log

Given context words within a window of size c, predict the target word w_t:

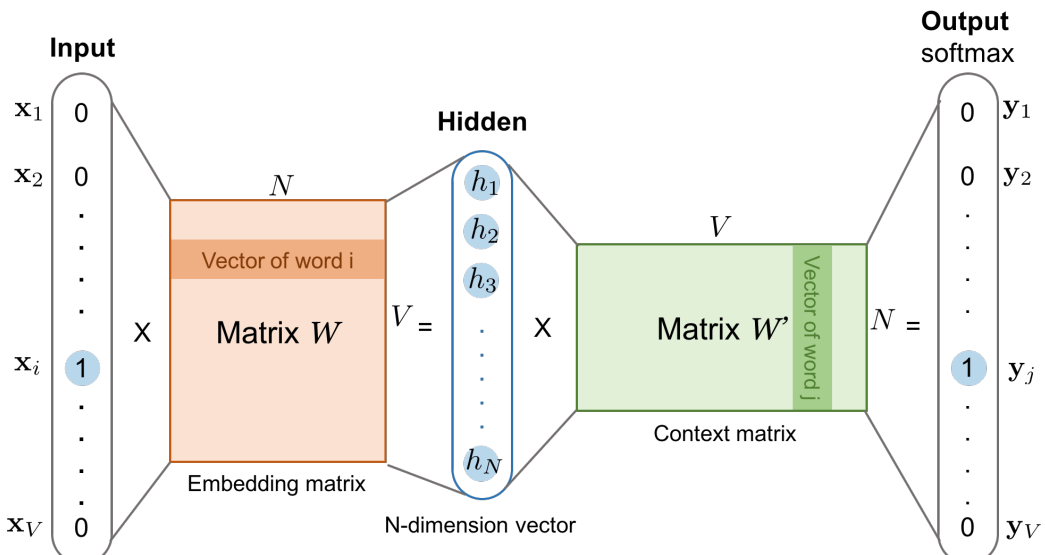


More stable (averaging the context smooths out the noise), faster.

Word2Vec – skip-gram

source: Lil'Log

Given a target word w_t, predict context words within a window of size c (one by one):



More efficient use of training data: each word generates c examples.

Word2Vec – how to generate skip-gram examples

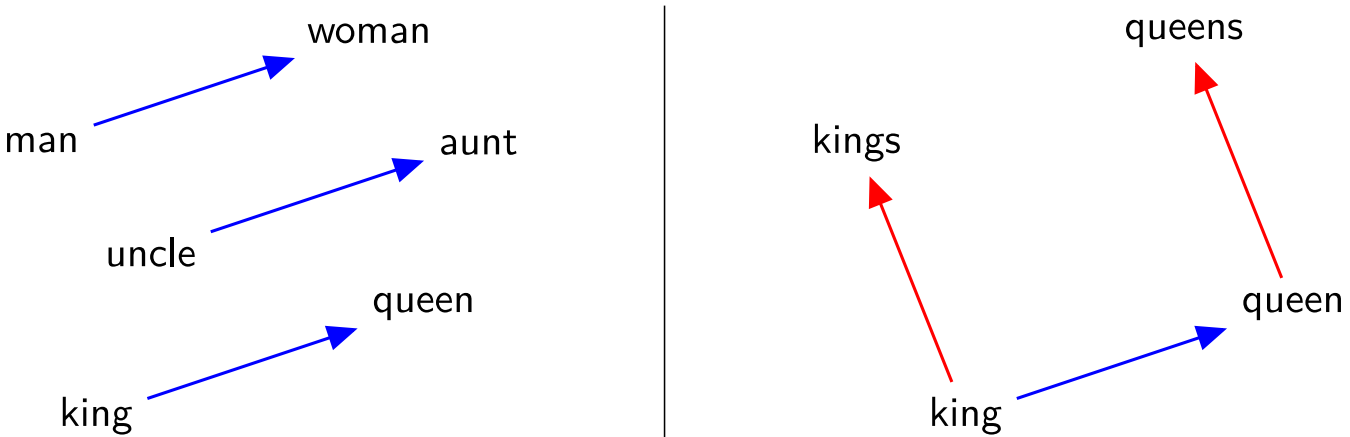
source: NPFL129

1.	[All human beings] are born free and equal in dignity ...	→	(All, humans) (All, beings)
2.	[All human beings are] born free and equal in dignity ...	→	(human, All) (human, beings) (human, are)
3.	[All human beings are born] free and equal in dignity ...	→	(beings, All) (beings, human) (beings, are) (beings, born)
4.	All [human beings are born free] and equal in dignity ...	→	(are, human) (are, beings) (are, born) (are, free)

Word2Vec – interesting properties

source: NPFL129

Learned embeddings seem to capture **linear relationships** between words.*



* With a few caveats: (1) the quality of relationships depends on the training data, (2) the arithmetics does not always work as nicely as expected, (3) closest word is often the word itself.

Other static word embeddings

- GloVe (Global Vectors): Constructs a word co-occurence matrix → factorizes it to get the embeddings.
- FastText: Word embedding is a sum of substring embeddings → can generate embeddings of unseen words.
- Backpack Language Models (Hewitt et al., 2023): combining multiple sense vectors for each word.

Question

What are the limitations of static word embeddings?

Limitations of static embeddings

Static embeddings = **a single vector** per word, regardless of context.

- The word **“bank”** has the same vector whether it means a *financial institution* or the *side of a river*.
- (No embedding for unseen words → solved by FastText)

Idea

Let’s compute the embedding of each word on-the-fly!

+ The embedding will depend on the current context.

- Cannot pre-compute and pre-download.

Intermezzo: Language modeling

Language modeling

A **language model** assigns a probability to a sequence of words: $P(w_1, w_2, ..., w_t)$

> **Question**
>
> Why can it be useful to know a probability of a sequence of words?

Tells us which sequences of words are more likely than others:

- First used for **automatic speech recognition (IBM, 1980s)** for disambiguating acoustically similar sequences.
- Other uses in **machine translation** (→ which translation to choose), spelling correction (→ what is likely grammatically correct), ...

Language modeling for word prediction

What if we want to use $P(w_1, w_2, ..., w_t)$ to predict the probability of the next word?

Using the **chain rule of probability**, we can decompose this as:

$$P(w_1, ..., w_t) = \prod_{k=1}^{t} P(w_t \mid w_1, ..., w_{k-1})$$
$$= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)...P(w_t \mid w_1, ..., w_{k-1})$$

To predict the **probability of the next word** v in timestep t:

$$P(w_t = v \mid w_1, ..., w_{t-1}) = \frac{P(w_1, ..., w_{t-1}, v)}{P(w_1, ..., w_{t-1})}$$

Estimating the n-gram probabilities

How to estimate $P(w_1, ..., w_t)$?

Idea

- Count the occurences of each n-gram $(w_1, ..., w_t)$ for $n \in \langle 1, 2, ..., \infty)$ in our dataset to build a table: $(w_1, ..., w_t)$ → n_occurences.
- Divide n_occurences by the total number of occurences to get a probability distribution.

Question

Do you see an issue with this approach?

N-gram language models

We can relax the problem: approximate $P(w_1, ..., w_t)$ with the last $n-1$ words:

$$P(w_t \mid w_1, ..., w_{t-1}) \approx P(w_t \mid w_{t-n+1}, ..., w_{t-1})$$

→ we get an **n-gram language model** (used e.g. in early autocomplete systems).

Problems:

- Most n-grams never appear.
- No dependencies between the individual n-grams.
- Gets infeasible for $n > 7$.

How large are the n-gram tables?

- approx. tens of MB for 2-grams
- approx. tens to hundreds of GB for 7-grams

(depends on training corpus)

Recurrent neural networks (RNNs)

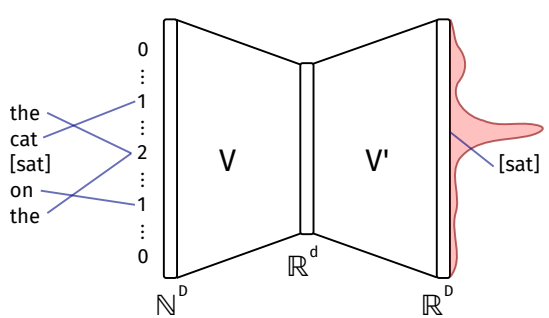
Neural architecture for language processing

source: Wikipedia

Language strings are **variable-length sequences**.

> **Question**
>
> Can we use a feed-forward network for language processing?

Sure! We did that to compute the static word embeddings:



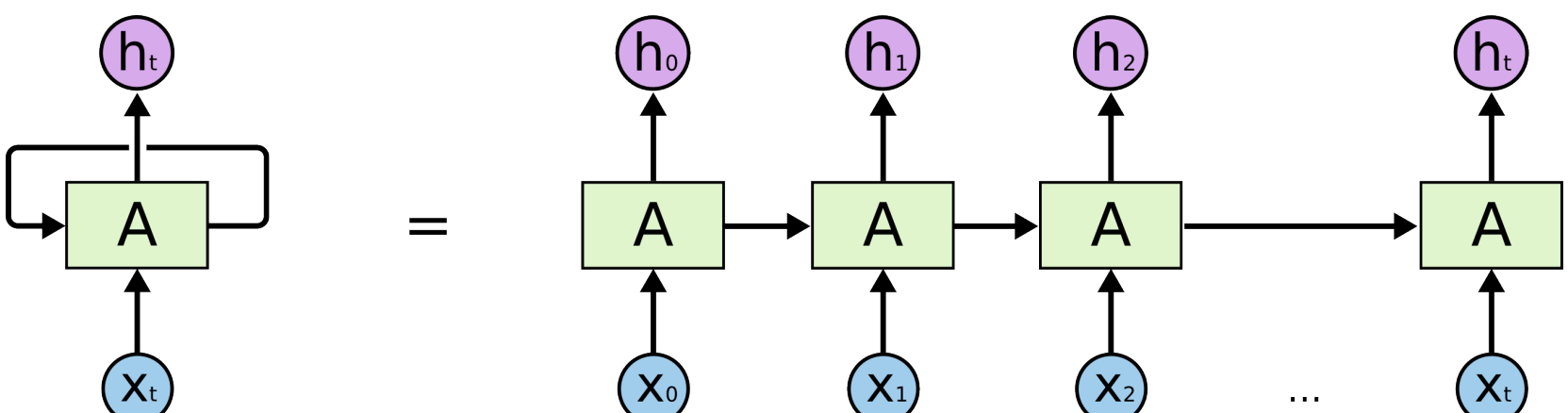
Recurrent neural networks (RNNs)

source: Colah's blog

However, sometimes we really want to **process the full sequence** to capture long-distance dependencies.

Idea

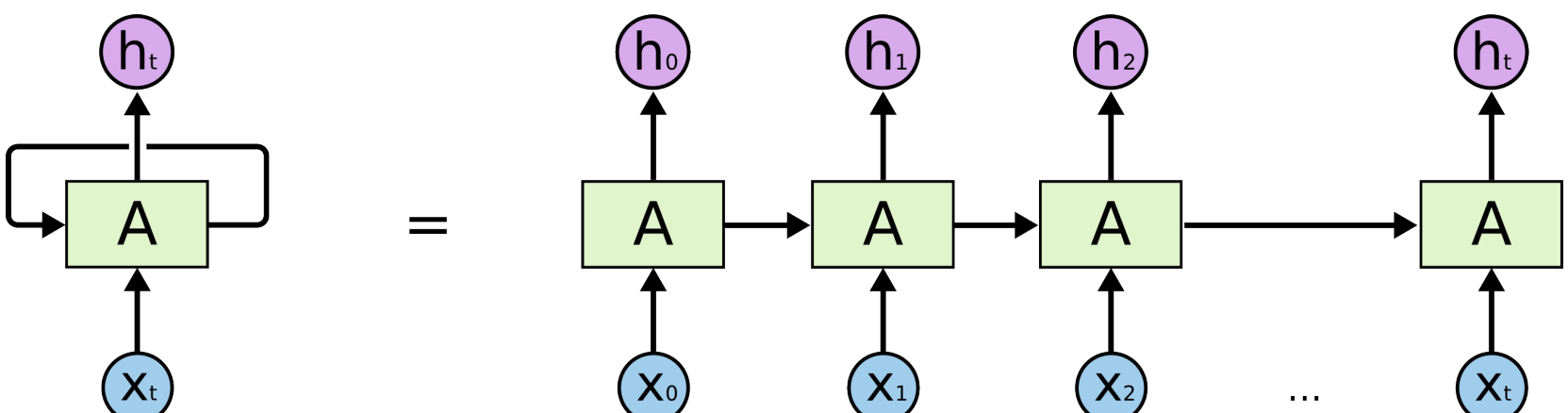
Let's create a `for`-like loop, processing input of length `N` in `N` steps



Recurrent neural networks (RNNs)

source: Colah's blog

- At each time step t, the RNN takes an input x_t and updates its hidden state h_t.
- The hidden state h_t accumulates context from all previous words.
- The **same weights** are shared across all time steps (→ it is essentially a FFNN applied to a single word in a loop.)



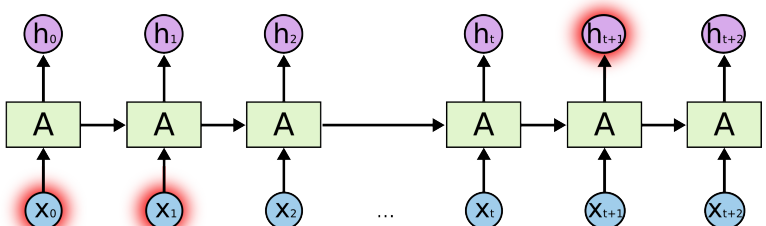
Issues with RNNs

source: Colah's blog

In theory, RNNs can process sequences of any length.

In practice:

- **Hidden state is a bottleneck**: all the information about the sequence must be compressed into a single vector h_t with several hundreds of parameters.
- **Gradients are vanishing**: limited numerical precision make repeated backpropagation over many steps infeasible.



Solutions: LSTM & GRU

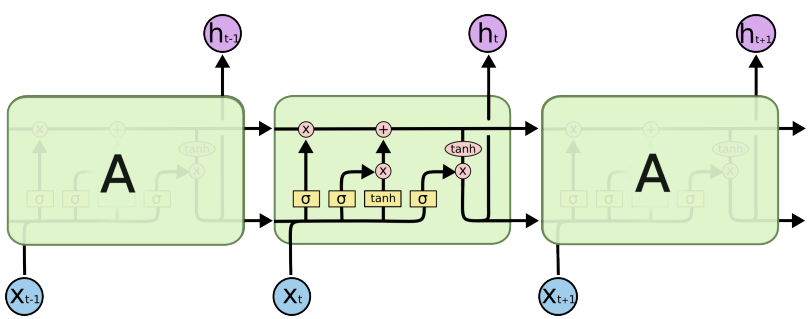
source: Colah's blog

More complex RNN architectures: workaround for these issues until 2018.

Long short-term memory (LSTM)

Hochreiter & Schmidhuber (1997)

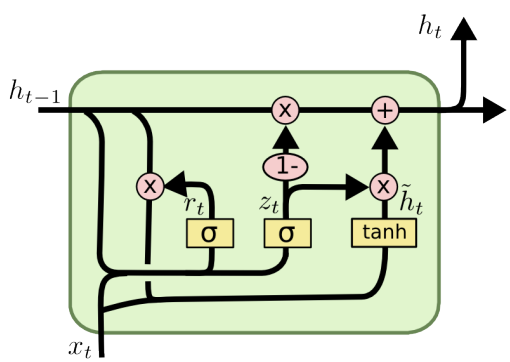
- **Forget gate**: what to discard.
- **Input gate**: what new info to store.
- **Output gate**: what to output.



Gated recurrent unit (GRU)

Cho et al. (2014), simplified LSTM variant.

- **Reset gate**: how much past info to forget.
- **Update gate**: balance between old and new.



What to do with the hidden state(s)?

1. **Classify the final state**: feed h_N to a classifier → prob. distr. over k classes.

$$\boldsymbol{p}(c \mid w_1, ..., w_N) = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_N + \boldsymbol{b})$$

$$\text{where } \boldsymbol{W} \in \mathbb{R}^{k \times d}, \quad \boldsymbol{b} \in \mathbb{R}^k, \quad \mathrm{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Examples:

- *Sentiment analysis*: This movie rocks! → 👍
- *Spam detection*: Buy cheap meds online! → ⛔

Question

Can we use this approach for language modeling?

Neural language modeling as a classification task

Goal: estimate $P(w_t \mid w_1, ..., w_{t-1})$ using an RNN.

> **Idea**
>
> We classify the final state into k classes, where $k = |V|$ is the size of our vocabulary.

Once again, we classify h_N with a **linear layer + softmax**:

$$\boldsymbol{p}(w_{t+1} \mid w_1, ..., w_t) = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_N + \boldsymbol{b})$$

$$\text{where } \boldsymbol{W} \in \mathbb{R}^{|V| \times d}, \quad \boldsymbol{b} \in \mathbb{R}^{|V|}$$

→ We can estimate the probability more efficiently than with n-gram models.

What to do with the hidden state(s)? – contd.

2. **Classify each state**: feed h_t at each time step to a classifier → prob. distr. over k classes per word.

Examples:

- *Part-of-speech tagging*: Time$^{\text{NOUN}}$ flies$^{\text{VERB}}$ like$^{\text{ADP}}$ an$^{\text{DET}}$ arrow$^{\text{NOUN}}$.
- *Named entity recognition*: Alan$^{\text{PER}}$ Turing$^{\text{PER}}$ was$^{\text{O}}$ born$^{\text{O}}$ in$^{\text{O}}$ London$^{\text{LOC}}$.

> **What is different here?**
>
> We classify each state separately instead of classifying just the final state.

What to do with the hidden state(s)? – contd.

3. **Generate a sequence**: use h_T as an initial state for another RNN (=seq2seq).

Examples:

- *Machine translation*: Language models → Jazykové modely
- *Summarization*: [...long detailed article...] → AI will destroy humanity.
- *Question answering*: Why to use LLMs? → To answer all your questions.

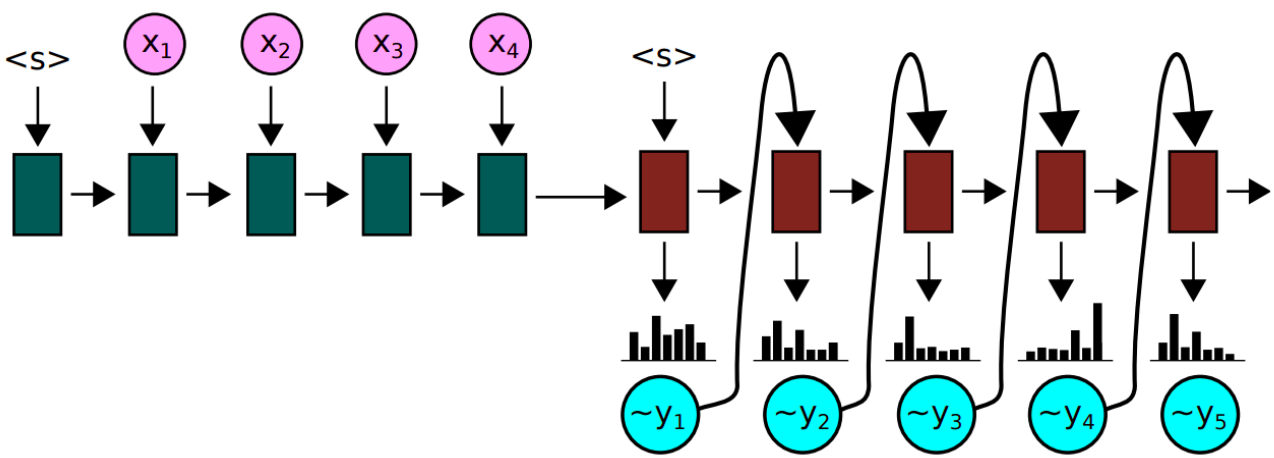
> **Question**
>
> How would you implement that?

Sequence-to-sequence models

Sequence-to-sequence (seq2seq) architecture

source: Sutskever et al. 2014

1. The **encoder** encodes the input $x_1, ..., x_N$ into the **hidden state** h_N.
2. The **decoder** starts with h_N and generates the sequence $y_1, ..., y_M$, at each time step conditioning on the hidden state h_{t-1} and the generated token y_{t-1}.



source: NPFL116

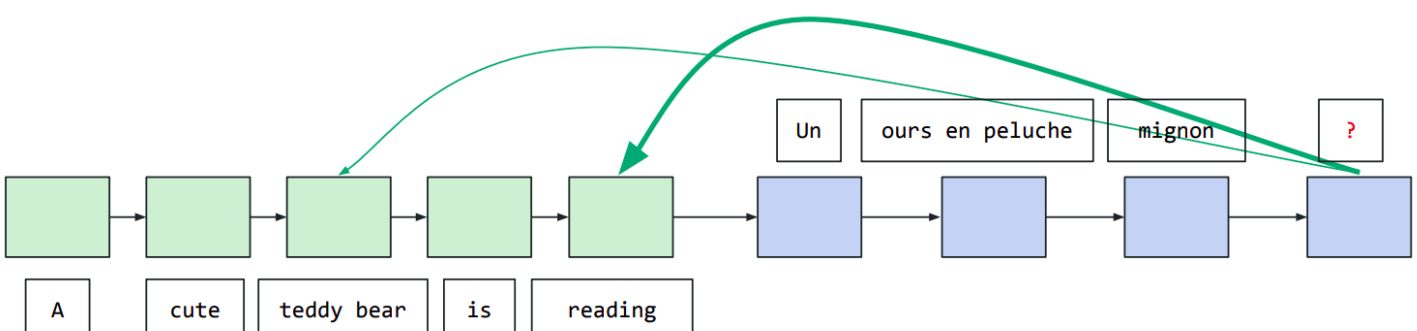
Up next: Can we solve the hidden state bottleneck?

source: Stanford CME295

The entire input is compressed into a **single fixed-size vector** h.

Question for the next time

Can we get around this issue?



Summary

Summary

Takeaways from today

- For processing language:
  - We first need to represent it efficiently → **word embeddings**.
  - We then need a way to process variable-length sequences → **RNNs**.
- The goal of **language modeling** is predicting the probability of a sequence of words
  - We can perform language modeling with RNNs by formulating it as a classification task.

Links

- Understanding LSTM networks – Colah's blog
- Illustrated Word2Vec – Jay Alammar
- Learning word embedding – Lil'Log