TDT4171 Artificial Intelligence Methods

Lecture 11 – Natural Language Processing

Norwegian University of Science and Technology

Helge Langseth
Gamle Fysikk 255
helge.langseth@ntnu.no

NTNU

Outline

1. Classical Natural Language Processing
   - Introduction
   - Probabilistic language models
   - Example: Learning over text data
   - Information Retrieval
2. Deep learning and NLP
   - Word embeddings
   - Transformers
   - RLHF: Reinforcement Learning with Human Feedback
3. Summary

Learning goals for the this part

Some knowledge of:

- **probabilistic models** for language.
- A “feel” for **classic NLP tasks** – Problem definition, simple solution, evaluation.
- **Deep Learning and NLP** – Word embeddings, attention, transformer model.

Note! The curriculum has been slightly updated: Two new subsections on Deep NLP. DL book is out. Check BB.

Language

- Language is a tool to **pass on information**
- Formal languages, like first-order logic, has a clear syntax and semantics. Natural languages do not, but are still of interest!
- Problems w/ analyzing natural language:
  - Ambiguity: "Bank" as financial institution vs. "river bank"
  - Subjectivity: "Football" vs. "soccer"
  - Incorrectness/Inconsistency: "I like to sleeping"
- Natural Language Processing tasks can be, e.g.:
  - Information extraction: Document to nuggets
  - Information retrieval: Information need to document
  - Machine translation: Language to language
  - Question-Answering: Query to nugget
  - Conversational systems: Learning, entertainment
  - Speech recognition: Sound to text
  - Speech synthesis: Text to sound

Probabilistic language model

- A **probabilistic language model** defines a probability distribution over (possibly infinite) list of strings
- This is an alternative to **logical** language models, and there are **several advantages**:
  - Can be **trained from data**; learning is simply counting occurrences in a **text corpora** (e.g., www)
  - More **robust**: recognizes that not all speakers agree upon when a sentence is part of a language.
  - Allows **disambiguation**: Use probabilities to say which interpretation is more likely
- Note that spoken languages (in contrast to e.g. programming languages) are "vague" both wrt. syntax and semantics $\Rightarrow$ Logic-based language model may break.

Tokenization

What constitutes a "token"?

- Token: the smallest meaningful building-block of a language.
- The ultimate goal is to create a basic unit that supports downstream tasks.
- **Possible definitions include:**
  - Word tokenization: "smarter"
  - Word stem: "smart"
  - Characters: "s", "m", "a", ...
- Basic unit is up to **system designer**.

In the slides I simplify and use "word" and "token" interchangeably.

Alternative language models

Problem: How to calculate the probability of a string of words $\langle w_1, w_2, \ldots, w_n \rangle$?

Unigram model: Assign a probability to each word w_i in the text-string,

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i).$$

$\Rightarrow$ Looks at each word in total isolation; disregards location of word in string. Often called **"bag of words"** approach.

As we don't have $P(w_i)$, we must approximate it. We can, e.g., use frequency of each word in the corpora.

Sampling from this distribution:

logical are as are confusion a may right tries agent goal that was diesel more object then information-gathering search

Alternative language models

Problem: How to calculate the probability of a string of words $\langle w_1, w_2, \ldots, w_n \rangle$?

Bigram model: Assigns probabilities to each word pair in the text-string, i.e., $P(w_i|w_{i-1})$.

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1}); \quad (P(w_1|w_0) := P(w_1))$$

planning purely diagnostic expert systems are very similar computational approach would be represented compactly using tic tac toe a predicate

Alternative language models

Problem: How to calculate the probability of a string of words $\langle w_1, w_2, \ldots, w_n \rangle$?

Trigram model: Assigns probabilities to each word given previous two words in the string, i.e., $P(w_i | w_{i-1}, w_{i-2})$.

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1}, w_{i-2})$$

planning and scheduling are integrated the success of naive Bayes model is just a possible prior source by that time

Alternative language models – what to choose?

- **Unigram** models often used when the corpora is **not too large**; Good enough to **categorize** text; too shallow to **generate** text.
- **Trigram** models require **huge** corpora. As an example, the AI book contains 15.000 different words $\Rightarrow$ there are $15.000^3 = 3.375.000.000.000$ possible word triplets.
- **Bigrams** define the "middle ground". Models still require **large** corpora ($15.000^2 = 225.000.000$ possible word pairs).
- **Problem:** If we have not seen a word pair, is it then impossible?

Alternative language models – what to choose?

- **Unigram** models often used when the corpora is **not too large**; Good enough to **categorize** text; too shallow to **generate** text.
- **Trigram** models require **huge** corpora. As an example, the AI book contains 15.000 different words $\Rightarrow$ there are $15.000^3 = 3.375.000.000.000$ possible word triplets.
- **Bigrams** define the "middle ground". Models still require **large** corpora ($15.000^2 = 225.000.000$ possible word pairs).
- **Problem:** If we have not seen a word pair, is it then impossible? — **Of course not!** $\Rightarrow$ Use **smoothing**:

$$P(w) := (c+1)/(N+B)$$

where

- c is no. times we see the word w
- N is no. word-observations in total
- B is no. different words

Example: Learning to Classify Text

Why consider how to classify text?

- Learn which news articles are of interest
- Learn to classify text documents (web pages, newsgroup entries) by topic

Setup:

- **Dataset:** A number of $\langle \texttt{Document}, \texttt{Class} \rangle$ pairs.
- **Text-documents:** A sequence of items/words from a vocabulary.

20 Newsgroups

- Given 1000 training documents from each group
- Learn to classify new documents according to which newsgroup it came from

comp.graphics	sci.space
comp.os.ms-windows.misc	sci.crypt
comp.sys.ibm.pc.hardware	sci.electronics
comp.sys.mac.hardware	sci.med
comp.windows.x	
alt.atheism	misc.forsale
soc.religion.christian	
talk.religion.misc	rec.autos
talk.politics.mideast	rec.motorcycles
talk.politics.misc	rec.sport.baseball
talk.politics.guns	rec.sport.hockey

Article from rec.sport.hockey

From: xxx@yyy.zzz.edu (John Doe)
Subject: Re: This year's biggest and worst
Date: 5 Apr 93 09:53:39 GMT

I can only comment on the Kings, but the most obvious candidate for pleasant surprise is Alex Zhitnik. He came highly touted as a defensive defenseman, but he's clearly much more than that. Great skater and hard shot (though wish he were more accurate). In fact, he pretty much allowed the Kings to trade away that huge defensive liability Paul Coffey. Kelly Hrudey is only the biggest disappointment if you thought he was any good to begin with. But, at best, he's only a mediocre goaltender. A better choice would be Tomas Sandstrom, though not through any fault of his own, but because some thugs in Toronto decided [. . .]

Example: Learning to Classify Text – Summary of Setup

- We have documents from 20 groups/classes, and 1000 training documents from each class.
- The main categories of classes are easily separated, but we want to recognize **each** of the 20 classes.
- The **vocabulary is unrestricted** — All documents written in plain but "uncleaned" English.
- Some parts of a document (e.g. `Alex Zhitnik`) tell a lot, but can we **generalize** from that?
- Since a word-sequence (i.e., a document) should be the basis of a probabilistic language model, we need to consider what **language model** to use.

Naïve Bayes Classifier

General classifier setup: Assume target function $f : \mathcal{X} \rightarrow V$, where each instance x is described by attributes $\langle a_1, a_2 \ldots a_n \rangle$. What is the most probable value of $f(x)$?

$$\begin{aligned} v^* &= \operatorname*{argmax}_{v_j \in V} P(v_j | a_1, a_2 \ldots a_n) \\ &= \operatorname*{argmax}_{v_j \in V} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)} \\ &= \operatorname*{argmax}_{v_j \in V} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \end{aligned}$$

Problem:

A_i is word at location i, and has no. states equal to the size of the vocabulary. n is the number of words in the document.

$\rightarrow$ The CPT $P(a_1, a_2 \ldots a_n | v_j)$ is **huge**!

Naïve Bayes Classifier

General classifier setup: Assume target function $f : \mathcal{X} \to V$, where each instance x is described by attributes $\langle a_1, a_2 \ldots a_n \rangle$. What is the most probable value of $f(x)$?

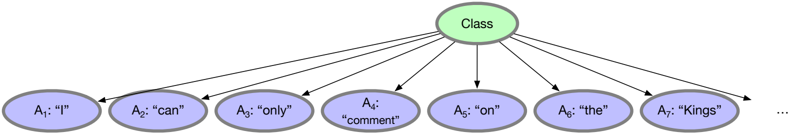
$$
\begin{aligned}
v^* &= \operatorname*{argmax}_{v_j \in V} P(v_j | a_1, a_2 \ldots a_n) \\
&= \operatorname*{argmax}_{v_j \in V} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)} \\
&= \operatorname*{argmax}_{v_j \in V} P(a_1, a_2 \ldots a_n | v_j) P(v_j)
\end{aligned}
$$

Use "**Naïve Bayes** assumption": $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$. This corresponds to a **class-specific unigram model**.

Naïve Bayes classifier: $v_{\text{NB}} = \operatorname*{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$

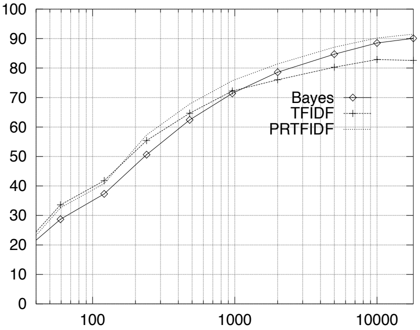
Approximate $P(a_i | v_j)$ as observed frequency of word at position i when class is v_j. Use smoothing!

The model as a Bayesian Network



- **Conditional Probability Tables:**
  - **Class variable** V: Observed fraction of each class;
    $P(V = v_j) \leftarrow \text{no obs. of } V = v_j / \text{no obs. in total}$.
  - **Attribute** A_i **given** V: Smoothed word fraction for class:
    $$P(A_i = \omega | V = v_j) \leftarrow \frac{1 + \text{no times word } \omega \text{ used in docs from } v_j}{B + \text{no words in docs from class } v_j}.$$
  - Note that the the CPTs for all A_i-nodes are **identical**!
- **Inference**: Insert observations $\langle a_1, \ldots, a_n \rangle$, where a_i is word at position i and n is the document length, then calculate $\text{argmax}_v\, P(v) \prod_{i=1}^{n} P(a_i|v)$.

Learning Curve for 20 Newsgroups



Accuracy vs. Training set size

- The **Naïve Bayes classifier**: Unigram/"Bag of Words" model w/smoothing is very useful in many (data-sparse) domains.
- Comparable to dedicated text-based approaches

Another classic task: Information retrieval

Information retrieval is the task of finding documents that are relevant to a user's need for information.

An IR system is characterized by a **document collection** $\mathcal{D}$, a **query** Q posed in some query language, and **result set** presented in some way.

Typical model designs

Boolean keyword model: Document is relevant if and only if the query Q evaluates true for that document.

Probabilistic model: Calculate the *probability* that a document $D \in \mathcal{D}$ is relevant to a query Q: $P(R = \texttt{true}|Q, D)$.

Evaluating IR systems

What is a good IR system?

- Consider an IR system looking at a dataset with 50 relevant and 50 irrelevant documents
  - The system should produce a result-set with as many relevant and as few irrelevant documents as possible

	In result set	Not in result set
Relevant	30	20
Not relevant	10	40

- **Precision** is the fraction of the result set we actually found relevant. Here, precision is $30/(30+10) = .75$.
- **Recall** is the fraction of the relevant documents we got to see. Here, recall is $30/(30+20) = .60$.

Can optimize each separately by **cleverly choosing the size of the result set**. Therefore, one should focus on *both* together.

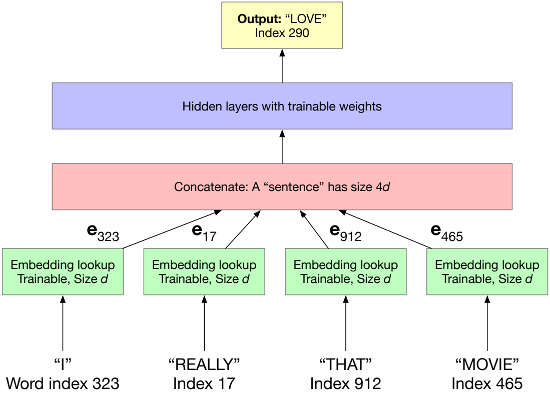
Deep learning in NLP

- Starting from the early 2010s, **deep learning has taken over** in almost all aspects of natural language processing.
- DL models are **routinely used** for, e.g., language understanding, generation, Q&A, chatbots, ...
- Currently there is an increasing focus on **multi-modality**, mixing text, sound, images, video, ...
- At the heart of this: A clever idea for language **representation** and an extremely efficient model for **inference**.

The first big question for deep learning: Representation!

- A sentence is a **sequence** of **words** (or “tokens”)
  - We are not afraid of the sequence-part: We have **RNNs**
- ... but **what about each word**?
  - If we have B words in the vocabulary, why not give them unique index-values?
  - ... or use one-hot encoding (binary vector of length B)? And maybe extend to something encoding n-grams, too?
- We will do better using **word-embeddings**:
  - For each word we define a high-dim. **vector-representation** for the word. This is a d-dimensional vector of real values, that hopefully somehow encodes the **semantics** of the word.
  - The representation is used **instead of** one-hot or whatever.
  - We **learn** the representation from massive data-sets.

Simple idea for learning Word-Embeddings

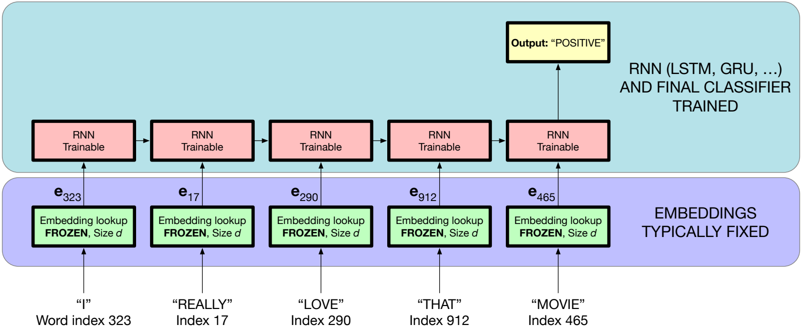


- **Simple idea:** Learn to predict the word in the middle of (sub-)string. Here, "`love`" in "`I really <?> that movie`".
  - Can also do **other tasks** like Part-of-Sentence tagging
- **Embeddings** and **classifier-head** learned simultaneously.

Learning Word-Embeddings (cont'd)

- We have a huge trainable matrix of **word embeddings**.
  - One vector of size d (e.g., $d = 12288$ in ChatGPT, 3072 in GPT4) per word in the vocabulary.
- We learn the embeddings **as we would weights**
  - End-result: We can **do a simple lookup**: $\texttt{word} \rightarrow e_{\text{word}}$.
- The embeddings are **dense** descriptors using real numbers, and typically work well for **downstream tasks**.
- The embeddings tend to be **similar** for words that are similar:
  - `sushi` $\approx$ `[sashimi, ramen, nigiri, teriyaki]`
  - `pasta` $\approx$ `[spaghetti, ravioli, carbonara, gnocchi]`
  - `burger` $\approx$ `[hamburger, cheeseburger, hotdog]`
  - `tokyo` $\approx$ `[japan, osaka, seoul, beijing, kyoto]`
  - `paris` $\approx$ `[hilton, france, florence, berlin]`
  - `liverpool` $\approx$ `[newcastle, southampton, swansea]`

Full-blown models are easy



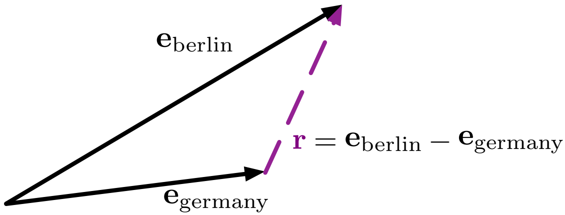
Example: Sentiment analysis

- Use pre-trained word embeddings for representation
- Use, e.g., an LSTM or other RNN to take care of the sequence
- Train a classifier on the last cell's internal representation.

Embedding "arithmetic"

Embeddings typically **high-dimensional** ($d \gg 1000$).

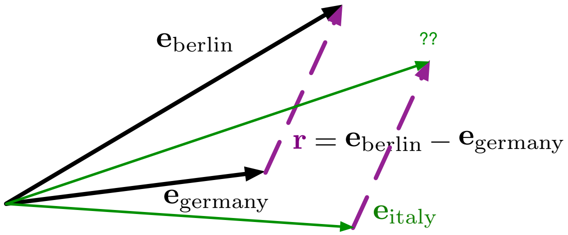
- Helps providing "enough space" to ensure that similar words have similar embeddings, while dissimilar words are separated.
- Helps provide language for **relations**:
  - Define $\boldsymbol{r} = \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$. Can we interpret $\boldsymbol{r}$ as a `capital_of`-relation now that $\boldsymbol{e}_{\text{berlin}} = \boldsymbol{r} + \boldsymbol{e}_{\text{germany}}$?



Embedding "arithmetic"

Embeddings typically **high-dimensional** ($d \gg 1000$).

- Helps providing "enough space" to ensure that similar words have similar embeddings, while dissimilar words are separated.
- Helps provide language for **relations**:
  - Define $\boldsymbol{r} = \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$. Can we interpret $\boldsymbol{r}$ as a `capital_of`-relation now that $\boldsymbol{e}_{\text{berlin}} = \boldsymbol{r} + \boldsymbol{e}_{\text{germany}}$?
  - **Yes**, because it holds that $\boldsymbol{e}_{\text{rome}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{italy}}$, $\boldsymbol{e}_{\text{tokyo}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{japan}}$, etc. for **the same** $\boldsymbol{r} := \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$.



Embedding "arithmetic"

Embeddings typically **high-dimensional** ($d \gg 1000$).

- Helps providing "enough space" to ensure that similar words have similar embeddings, while dissimilar words are separated.
- Helps provide language for **relations**:
  - Define $\boldsymbol{r} = \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$. Can we interpret $\boldsymbol{r}$ as a `capital_of`-relation now that $\boldsymbol{e}_{\text{berlin}} = \boldsymbol{r} + \boldsymbol{e}_{\text{germany}}$?
  - **Yes**, because it holds that $\boldsymbol{e}_{\text{rome}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{italy}}$, $\boldsymbol{e}_{\text{tokyo}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{japan}}$, etc. for **the same** $\boldsymbol{r} := \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$.
- This makes LLMs excellent at **association-games**: Find the word whose embedding is closest to $\boldsymbol{r} + \boldsymbol{e}_{\langle\texttt{start}\rangle}$.
  - "`king`" is to "`queen`" as "`uncle`" is to "`aunt`" because $\boldsymbol{e}_{\text{king}}$ is the embedding closest to $(\boldsymbol{e}_{\text{uncle}} - \boldsymbol{e}_{\text{aunt}}) + \boldsymbol{e}_{\text{queen}}$.

Embedding "arithmetic"

Embeddings typically **high-dimensional** ($d \gg 1000$).

- Helps providing "enough space" to ensure that similar words have similar embeddings, while dissimilar words are separated.
- Helps provide language for **relations**:
  - Define $\boldsymbol{r} = \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$. Can we interpret $\boldsymbol{r}$ as a `capital_of`-relation now that $\boldsymbol{e}_{\text{berlin}} = \boldsymbol{r} + \boldsymbol{e}_{\text{germany}}$?
  - **Yes**, because it holds that $\boldsymbol{e}_{\text{rome}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{italy}}$, $\boldsymbol{e}_{\text{tokyo}} \approx \boldsymbol{r} + \boldsymbol{e}_{\text{japan}}$, etc. for **the same** $\boldsymbol{r} := \boldsymbol{e}_{\text{berlin}} - \boldsymbol{e}_{\text{germany}}$.
- This makes LLMs excellent at **association-games**: Find the word whose embedding is closest to $\boldsymbol{r} + \boldsymbol{e}_{\langle \texttt{start} \rangle}$.
  - "`king`" is to "`queen`" as "`uncle`" is to "`aunt`" because $\boldsymbol{e}_{\text{king}}$ is the embedding closest to $(\boldsymbol{e}_{\text{uncle}} - \boldsymbol{e}_{\text{aunt}}) + \boldsymbol{e}_{\text{queen}}$.
  - "`nurse-midwife`" is to "`doctor`" as "`woman`" is to "`man`"
  - "`rome`" is to "`tokyo`" as "`italy`" is to "`japan`"
  - "`pizza`" is to "`sushi`" as "`italy`" is to "`japan`"

Word relations

Start with the word `king` and consider what relations we could benefit from using and what they should do:

- $\texttt{king} + \textbf{feminine} \rightarrow \texttt{queen}$
- $\texttt{king} + \textbf{young} \rightarrow \texttt{crown\_prince}$
- $\texttt{king} + \textbf{powerless} \rightarrow \texttt{prince\_consort}$
- $\texttt{king} + [\textbf{feminine}, \textbf{young}] \rightarrow \texttt{crown\_princess}$
- $\texttt{king} + [\textbf{young}, \textbf{feminine}] \rightarrow \texttt{crown\_princess}$
- $\texttt{king} + [\textbf{feminine}, \textbf{young}, \textbf{powerless}] \rightarrow \texttt{princess}$

Word relations

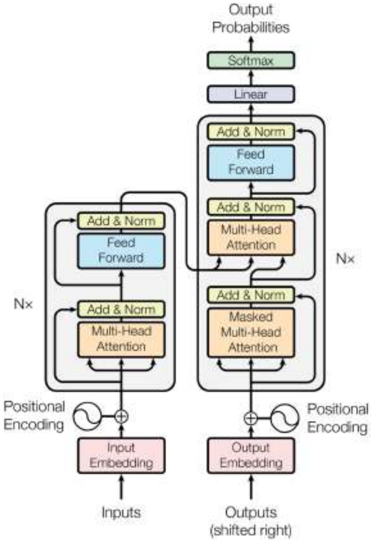
Start with the word `king` and consider what relations we could benefit from using and what they should do:

- `king` + **feminine** → `queen`
- `king` + **young** → `crown_prince`
- `king` + **powerless** → `prince_consort`
- `king` + [**feminine**, **young**] → `crown_princess`
- `king` + [**young**, **feminine**] → `crown_princess`
- `king` + [**feminine**, **young**, **powerless**] → `princess`

Worth noticing:

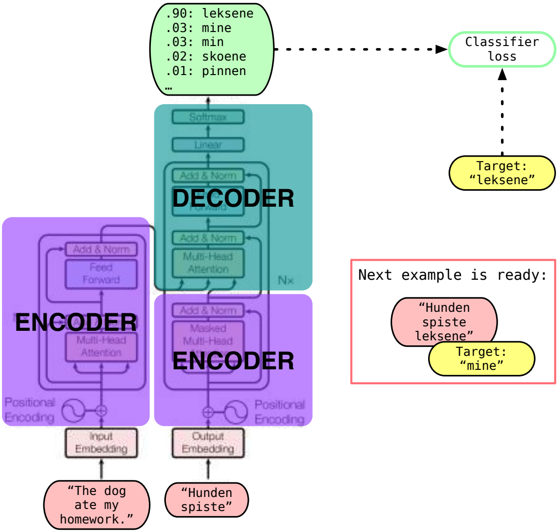
- We need many relations; must be found during learning.
- Relations (almost) additive ∼ vectors (almost) perpendicular.
- High-dim space have lots of room for "almost perpendicular" vectors, number grows exponential in d.
- Relations are used in **different contexts**: `japan` → `italy`, `tokyo` → `rome` and `sushi` → `pizza` identical vector-moves.

The Transformer architecture



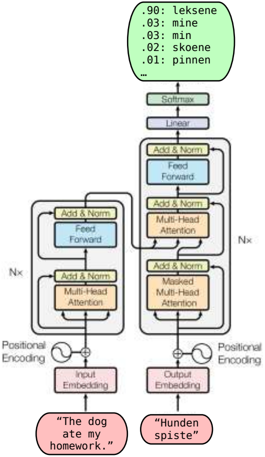
- Google (2017): *"Attention Is All You Need"* describes an encoder-decoder model that feeds off these relations.
- The **attention**-module extremely important in NLP (and other DL applications).
- Use **positional encoding** to help encode sequential structure of data.
- Modelling **heavily optimized** for parallell computation on GPUs.

The Transformer architecture

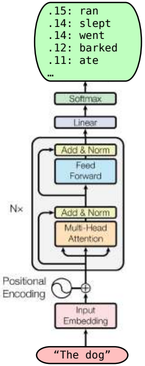


Different usecase result in different model designs

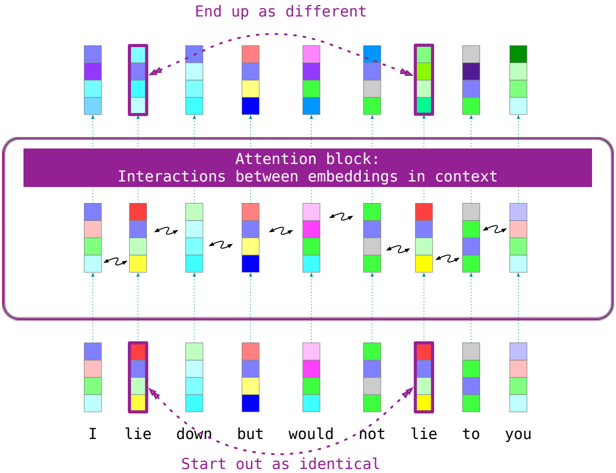
Sequence-to-sequence
(e.g. translation)



Sequence-to-single
(e.g. next word)
Only uses ENCODER



The attention-block: Adding contextual information



- A word-embedding is **local**, and does not capture **context**.
- e_{lie} same when used in "I lie down" and "I wouldn't lie to you" .
- Attentions provide a mechanism to fix this.

The attention-block: Querying the context

- **Word embeddings support "relations":**
  - `berlin = capital_of(germany)`
  - Relations are vector additions: $e_{\text{berlin}} = r_{\text{capital\_of}} + e_{\text{germany}}$.
  - The relations are implicit (there is no $r_{\text{capital\_of}}$), but found by comparing existing vectors (like e_{rome} and e_{italy}).
  - The existence of these relations is a bi-product of the learning.

- **Attention will capture this from the context:**
  - Consider sentence "`This city is the capital of Germany`". After attention layers, we want the attended embedding for "`city`" to become **close to** $e_{\texttt{berlin}}$.
  - Similarly, $e_{\texttt{lie}}$ must be colored by context in the sentence "`I lie down.`" to signify lie as "stretch out" not "deceive".
  - This must happen by each word **querying** the other words in the context in ways that generate meaningful **responses**.
  - One idea would be to train a neural network that for each word **produces a relation-vector** based on the words in the context.

Attention ~ Get info through "soft" database lookup

The parts involved:

- $\boldsymbol{q}_\omega$: The **query** raised by a word ω. Dim d not necessarily the same as embedding dim.
- $\boldsymbol{k}_{\mathtt{w}_j}$: The **key** to the info encoded by each $\mathtt{w}_j$ in context.
- $\boldsymbol{v}_{\mathtt{w}_j}$: The **value** of the information encoded by $\mathtt{w}_j$.
- $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j})$: The **similarity** between query and key, hence between the words ω and $\mathtt{w}_j$. High similarity means $\boldsymbol{v}_{\mathtt{w}_j}$ is relevant for ω.

The plan:

1. Define similarity between ω and each $\mathtt{w}_j$: $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j})$.
2. Pick out words in the context most similar to ω.
3. Enrich the embedding of ω with the values of those words.

Attention $\sim$ Get info through "soft" database lookup

The plan:

1. Define similarity between ω and each $\mathtt{w}_j$: $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j})$.
2. Pick out words in the context most similar to ω.
3. Enrich the embedding of ω with the values of those words.

How it is done:

1. Sim. is normalized **inner product**: $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j}) = \boldsymbol{q}_\omega \boldsymbol{k}_{\mathtt{w}_j}^\mathsf{T} / \sqrt{d}$.
   - Normalization $\sqrt{d}$: d is dim of query / key.
   - $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j}) \approx 0$ if $\boldsymbol{q}_\omega$ and $\boldsymbol{k}_{\mathtt{w}_j}$ random. "Non-zero means signal".
2. Words are weighted according to **softmax** of similarities.
3. Add $\sum_j \mathtt{softmax}\left[s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}})\right]_j \boldsymbol{v}_{\mathtt{w}_j}$ to $\boldsymbol{e}_\omega$.

Note! In reality the calculations are tensor (not vector) operations – All attentions in parallell.

Attention ∼ Get info through “soft” database lookup

	Query		Key	Value
		“I”:	[1., −2., 3., 4.]	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
“lie”:	[4., 3., 1., 0.]	“lie”:	[0.1, −0.2, 0.3, 0.4]	[0.6, 0.5, 0.4, 0.3, 0.2, 0.1]
		“down”:	[1., 0., 1., 1.]	[1., −1., 2., −1., 2., 0.]

Attention ∼ Get info through "soft" database lookup

Query

"lie":	[4., 3., 1., 0.]

	Key	Value	q x k/sqrt, Softmax	
"I":	[1., -2., 3., 4.]	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]	.5	.11
"lie":	[0.1, -0.2, 0.3, 0.4]	[0.6, 0.5, 0.4, 0.3, 0.2, 0.1]	.05	.07
"down":	[1., 0., 1., 1.]	[1., -1., 2., -1., 2., 0.]	2.5	.82

$$s\Big(\mathbf{q}_\omega = [4,3,1,0],\, \mathbf{k}_{\mathtt{w}_1} = [1,-2,3,4]\Big) = \frac{\mathbf{q}_\omega\, \mathbf{k}_{\mathtt{w}_1}^\top}{\sqrt{d}} = \frac{4\cdot 1 + 3\cdot(-2) + 1\cdot 3 + 0\cdot 4}{\sqrt{4}} = .5$$

$$\mathrm{softmax}([.5, .05, 2.5]) = [.11, .07, .82]$$

Attention ∼ Get info through "soft" database lookup

	Query
"lie":	[4., 3., 1., 0.]

	Key	Value
"I":	[1., -2., 3., 4.]	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
"lie":	[0.1, -0.2, 0.3, 0.4]	[0.6, 0.5, 0.4, 0.3, 0.2, 0.1]
"down":	[1., 0., 1., 1.]	[1., -1., 2., -1., 2., 0.]

q x k/sqrt	Softmax
.5	.11
.05	.07
2.5	.82

Softmax x Value

.11 · [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

+

.07 · [0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

+

.82 · [1., -1., 2., -1., 2., 0.]

= [.9., -.8, 1.7, -.8, 1.7, .1]

Attended addition

Attention $\sim$ Get info through "soft" database lookup

The plan:

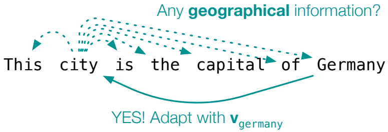
1. Define similarity between ω and each $\mathtt{w}_j$: $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j})$.
2. Pick out words in the context most similar to ω.
3. Enrich the embedding of ω with the values of those words.

How it is done:

1. Sim. is normalized **inner product**: $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j}) = \boldsymbol{q}_\omega \boldsymbol{k}_{\mathtt{w}_j}^\mathsf{T} / \sqrt{d}$.
   - Normalization $\sqrt{d}$: d is dim of query / key.
   - $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_j}) \approx 0$ if $\boldsymbol{q}_\omega$ and $\boldsymbol{k}_{\mathtt{w}_j}$ random. "Non-zero means signal".
2. Words are weighted according to **softmax** of similarities.
3. Add $\sum_j \mathtt{softmax}\left[s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}})\right]_j \boldsymbol{v}_{\mathtt{w}_j}$ to $\boldsymbol{e}_\omega$.

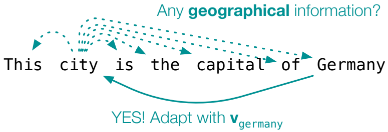
Note! In reality the calculations are tensor (not vector) operations – All attentions in parallell.

Where do the $\langle \texttt{query}, \texttt{key}, \texttt{value} \rangle$-vectors come from?



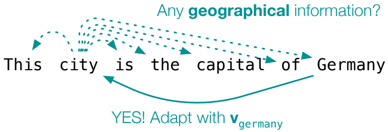
- A word's **query** must relate to the word itself, e.g., "`city`" can be enriched with **geographical information**.
  - The query comes from the embedding, e.g., $\boldsymbol{q}_\omega \leftarrow \boldsymbol{W}_Q \cdot \boldsymbol{e}_\omega$.

Where do the ⟨query, key, value⟩-vectors come from?



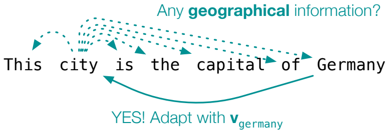
- A word's **query** must relate to the word itself, e.g., "`city`" can be enriched with **geographical information**.
  - The query comes from the embedding, e.g., $\boldsymbol{q}_\omega \leftarrow \boldsymbol{W}_Q \cdot \boldsymbol{e}_\omega$.
- A word's **key** must know what queries a word can answer to, e.g., `germany` is triggered by **geography**-queries.
  - The keys relate to the embedding, e.g., $\boldsymbol{k}_{\texttt{w}_j} \leftarrow \boldsymbol{W}_K \cdot \boldsymbol{e}_{\texttt{w}_j}$.
  - Keys must relate to possible queries $\Rightarrow$ co-training!

Where do the ⟨query, key, value⟩-vectors come from?



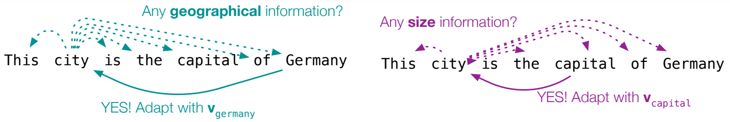
- A word's **query** must relate to the word itself, e.g., "`city`" can be enriched with **geographical information**.
  - The query comes from the embedding, e.g., $\boldsymbol{q}_\omega \leftarrow \boldsymbol{W}_Q \cdot \boldsymbol{e}_\omega$.
- A word's **key** must know what queries a word can answer to, e.g., `germany` is triggered by **geography**-queries.
  - The keys relate to the embedding, e.g., $\boldsymbol{k}_{\mathtt{w}_j} \leftarrow \boldsymbol{W}_K \cdot \boldsymbol{e}_{\mathtt{w}_j}$.
  - Keys must relate to possible queries $\Rightarrow$ co-training!
- A word's **value** must know **how** the word answers the query, e.g., the effect of "`german`-ity".
  - Values relate to the embeddings, e.g., $\boldsymbol{v}_{\mathtt{w}_j} \leftarrow \boldsymbol{W}_V \cdot \boldsymbol{e}_{\mathtt{w}_j}$.
  - Must relate to queries $\Rightarrow$ co-training!

Where do the ⟨query, key, value⟩-vectors come from?



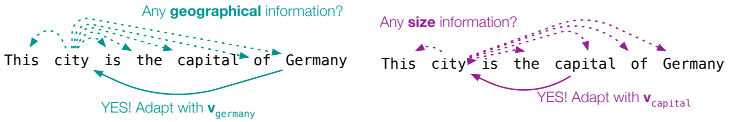
- A word's **query** must relate to the word itself, e.g., "`city`" can be enriched with **geographical information**.
  - The query comes from the embedding, e.g., $\boldsymbol{q}_\omega \leftarrow \boldsymbol{W}_Q \cdot \boldsymbol{e}_\omega$.
- A word's **key** must know what queries a word can answer to, e.g., `germany` is triggered by **geography**-queries.
  - The keys relate to the embedding, e.g., $\boldsymbol{k}_{\texttt{w}_j} \leftarrow \boldsymbol{W}_K \cdot \boldsymbol{e}_{\texttt{w}_j}$.
  - Keys must relate to possible queries $\Rightarrow$ co-training!
- A word's **value** must know **how** the word answers the query, e.g., the effect of "`german`-ity".
  - Values relate to the embeddings, e.g., $\boldsymbol{v}_{\texttt{w}_j} \leftarrow \boldsymbol{W}_V \cdot \boldsymbol{e}_{\texttt{w}_j}$.
  - Must relate to queries $\Rightarrow$ co-training!
- In total, we must learn the matrixes $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$ and $\boldsymbol{W}_V$.

More context: Multi-head attention

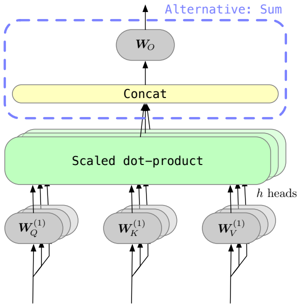


- $e_{\texttt{city}}$ can be enriched by "german-ity" **and** "capital-ness".
- One query-vector cannot ask for both; **specialization**!

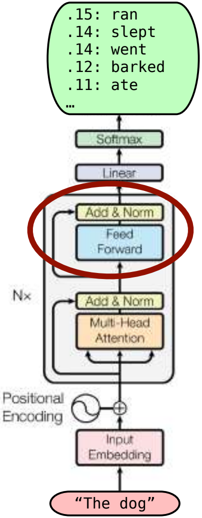
More context: Multi-head attention



- $e_{\texttt{city}}$ can be enriched by "german-ity" **and** "capital-ness".
- One query-vector cannot ask for both; **specialization**!
- Instead, learn h **separate** query-key-value mappings!
- The outputs of each "head" must be **combined**:
  - It is natural to just **add them** together.
  - Concat and linear mix is **more general**.

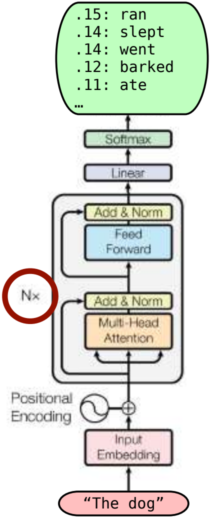


Last pieces of the puzzle



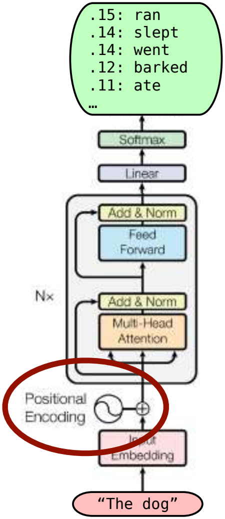
- **Feed forward** neural network in skip-connection, followed by normalization.

Last pieces of the puzzle



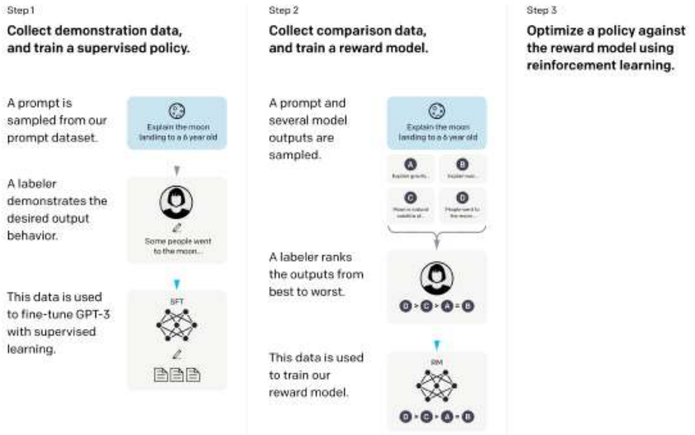
- **Feed forward** neural network in skip-connection, followed by normalization.
- The (Attention + FF)-block is **repeated** N times; typically $N \geq 6$.

Last pieces of the puzzle



- **Feed forward** neural network in skip-connection, followed by normalization.
- The (Attention + FF)-block is **repeated** N times; typically $N \geq 6$.
- Attention is **agnostic** to word-positions:
  - Similarity fails to encode **"sequential closeness"**. If words 1 and 100 are identical, then $s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_1}) = s(\boldsymbol{q}_\omega, \boldsymbol{k}_{\mathtt{w}_{100}})$.
  - **Problematic** for a sentence like "`John arrived after Tim, who had been at work.`". Who was at work?
  - **Positional encoding:** An embedding of position that is **added** to the input embedding. Typically **predefined** using trigonometric functions.

Approaching natural interaction



Open AI (2022): "Training language models to follow instructions with human feedback"

- LLMs trained **supervised** can be "awkward" to interact with.
- Reinforcement Learning with Human Feedback: Mimic the response-preferences of **human labelers**.

Summary

- **"Classic" Natural Language Processing:**
  - Simple language models (unigram, bigram, trigram) a core component of these systems.
  - Useful for limited-data scenarios and for simple tasks like classification, etc.
- **Deep Learning in NLP:**
  - Word embeddings revolutionised the field in early 2010s. High-dim representations that to some extent capture semantics of language.
  - Transformers the go-to model class for NLP (and other) applications: Extremely efficient structure optimized for massive parallell computations
  - Contemporary language models have pleasant interfaces obtained using RLHF. Some think they are sentient/close to AGI, a view I still do not share.