Maschinelles Lernen
- Informatik -

Rapp, DHBW Lörrach

02.02.2024

Inhaltsübersicht

Prüfungsaufgabe

Prüfungsaufgabe 4

Entwicklung eines Kunst-Chatbots

1. Erstellen Sie repräsentative Anfragen *(engl. Prompts)* an einen Kunst-Chatbot während einer Kunstmesse mit den entsprechenden User Stories in tabellarischer Form aus Sicht von
   - Kunst-Sammler (z.B. Privatier, Museum) und
   - Kunst-Aussteller (z.B. Gallerie, Künstler).

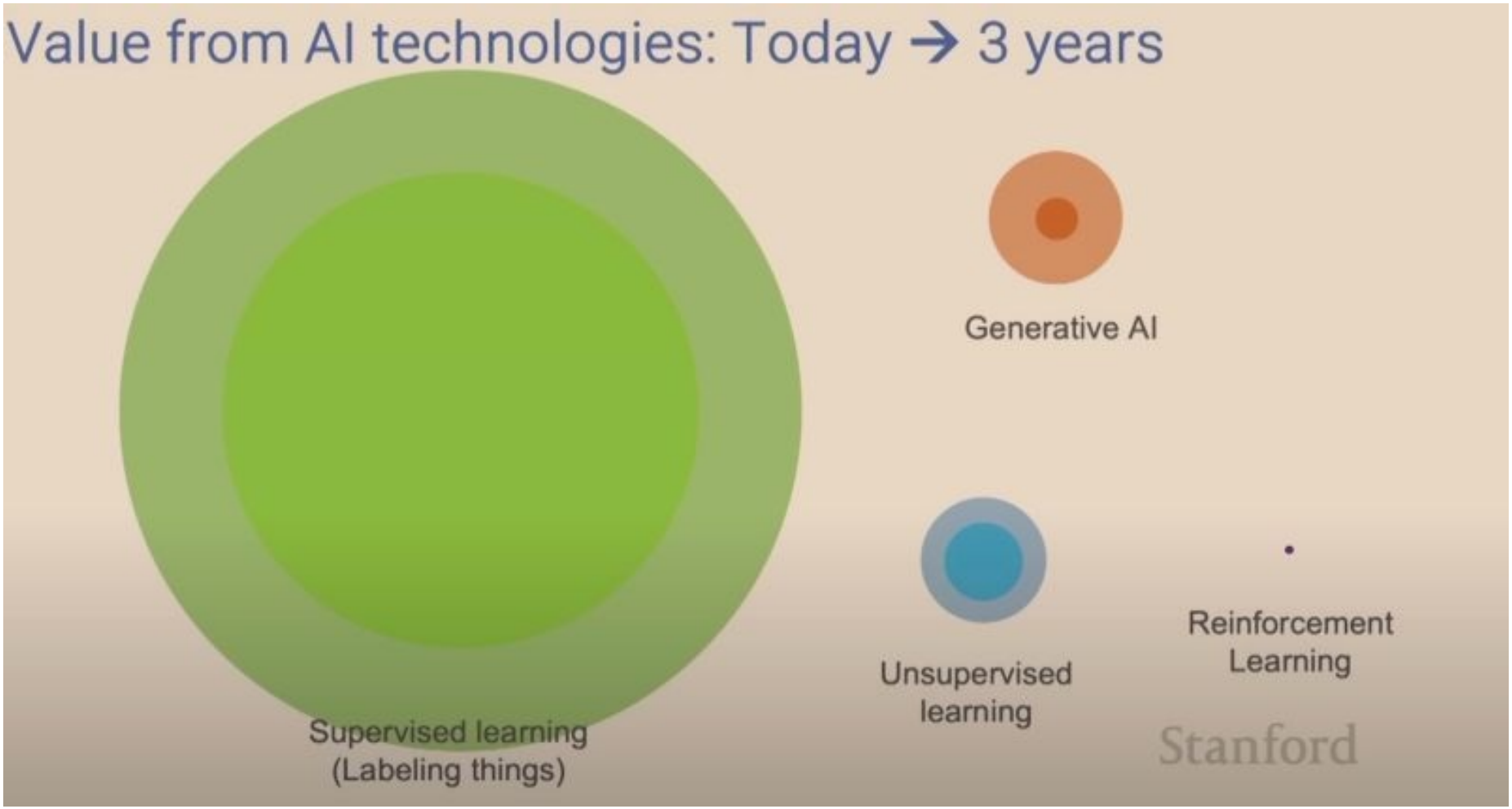
Prompt	As a <role>	I want <goal>	so that <benefit>	Acceptance criteria (Conditions of satisfaction)
...	...	...	...	...

2. Optimieren Sie den Kunst-Chatbot im Programmcode Chatbot_LangChain_RAG.py auf die beiden Metriken
   - Sensibleness und Specifity Average (SSA) und
   - Perplexität.
3. Identifizieren Sie zusätzliche Optimierungspotentiale des Kunst-Chatbots und beurteilen diese im Hinblick auf den Einsatz bei einer realen Kunstmesse.

Beispiele für Bewertungskriterien

- Die Antworten des Kunst-Chatbots auf repräsentative Anfragen sind sinnvoll und spezifisch.
- Aus dem Programmcode ist ersichtlich, dass die wesentlichen Konzepte aus der Vorlesung verstanden und auf die konkrete Problemstellung angewendet wurden.
- Die Argumentationsstruktur bei der Beantwortung der Fragen ist stringent und nachvollziehbar.
- Der Fremdanteil am Programmcode ist mit entsprechenden Quellenangaben gekennzeichnet.
- ...

Motivation


Value from AI technologies: Today → 3 years
Generative AI
Supervised learning
(Labeling things)
Unsupervised
learning
Reinforcement
Learning
Stanford

Cross Entropy Loss Function

Loss Funktionen

Loss Funktion *(=Verlustfunktion)*

- **Allgemeine Definition:** Schätzt den Schaden eines statistischen Entscheidungsproblems, der dadurch entsteht, dass die Entscheidung vom wahren Parameter abweicht.
- **Beispiel Regression:** Quadratische Error Loss Funktion $\lambda(x) = (t - x)^2$

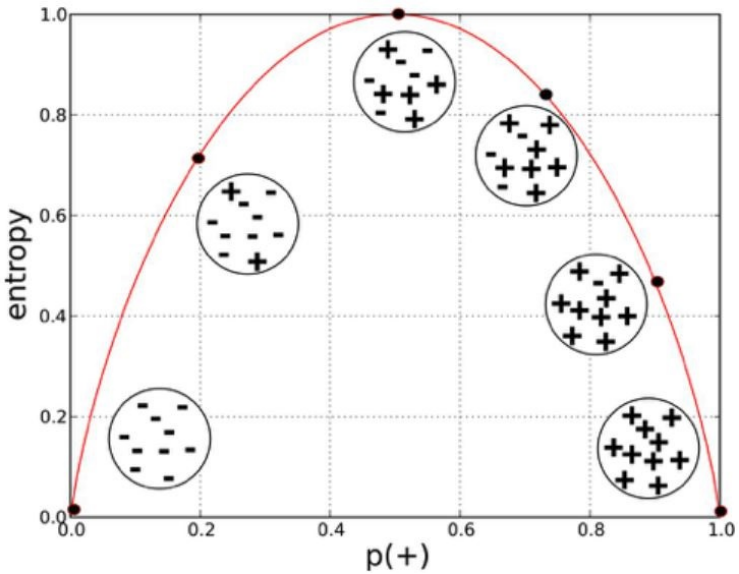
Cross-Entropy Loss Funktion

- Misst die Performanz eines Klassifikationsmodells, dessen Vorhersage einem Wahrscheinlichkeitswert zwischen 0 und 1 entspricht.
- Standard-Loss-Funktion für Klassifikationsprobleme
- Wird umso kleiner, je näher die vorhergesagte Klasse der tatsächlichen entspricht.
- **Minimierung** der Cross-Entropy in Bezug auf Modellparameter entspricht **Maximierung der Log-Likelihood Funktion.**

Definition *(Wiederholung)*

Wir bewerten Daten hinsichtlich ihrer Unreinheit (bzw. Unordnung) mit der **Entropie** (ugs. „*Maß für Unordnung*“) im Sinne der Informatik:

$$H(X) = -\sum_x p(x)\ log_2\ p(x)$$

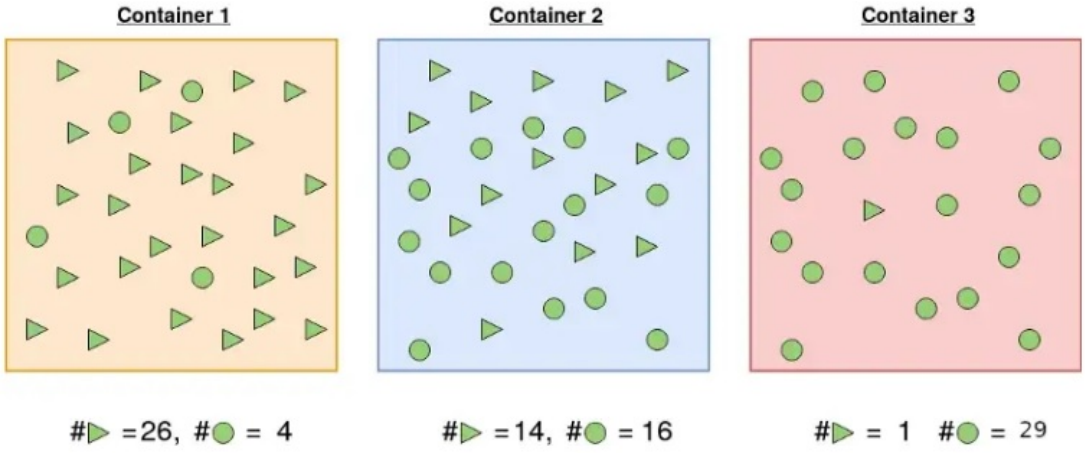


Beispiele

- $H(X) = 0$, falls $p(x) = 1$ für ein x für $k = 2$ Klassen
- $H(X) = 1$ mit $p(x) = \frac{1}{2}$

Beispiel

Berechnung der Entropie in Containern mit Dreiecken und Kreisen



$$
\begin{aligned}
H_1(X) &= -\sum_x p(x)\log(p(x)) \\
&= -[p(x_1)\log_2(p(x_1)) + p(x_2)\log_2(p(x_2))] \\
&= -\left[\frac{26}{30}\log_2\left(\frac{26}{30}\right) + \frac{4}{30}\log_2\left(\frac{4}{30}\right)\right] \\
&= 0.5665 \\
H_2(X) &= 0.9968 \\
H_3(X) &= 0.2108
\end{aligned}
$$

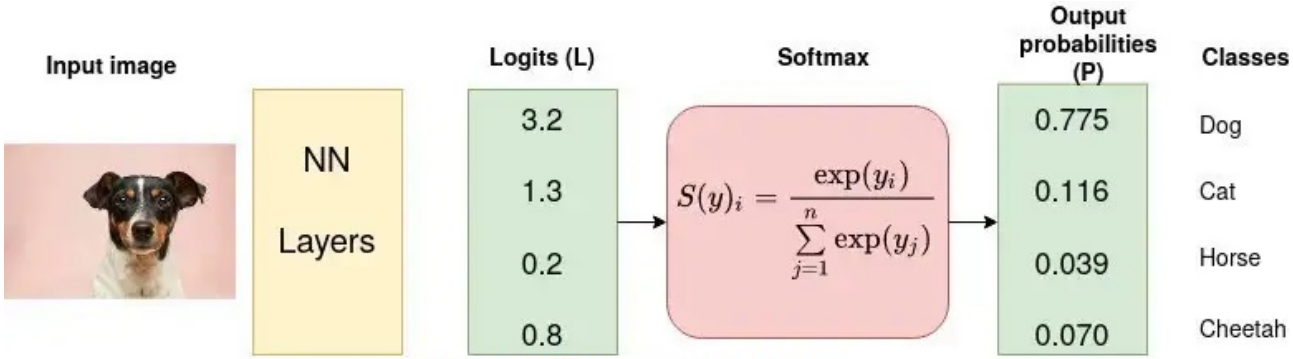
Die Entropien des ersten und dritten Containers sind kleiner als die im zweiten Container. Das liegt daran, dass das Ereignis, eine gegebene Form zu entnehmen, im ersten und dritten Container sicherer ist.

Definition

Cross-Entropy Loss *(n Klassen)*

$$L_{CE} = -\sum_{i=1}^{n} t_i \ log(p_i)$$

- t_i: Label
- p_i: Softmax Wahrscheinlichkeit für die Klasse i
- wird verwendet, um die Modellgewichte während des Trainings anzupassen



From: towardsdatascience.com, Koech

$n = 2$: Binary Cross-Entropy Loss *(Binäre Klassifikation)*

$$\begin{aligned} L_{BCE} &= -\sum_{i=1}^{2} t_i \ log(p_i) \\ &= -[t_1 \ log(p_1) + t_2 \ log(p_2)] \\ &= -[t \ log(p) + (1-t) \ log(1-p)] \\ &\overset{t=1, p=0.9}{=} -log(0.9) = 0.11 \end{aligned}$$

Sequence to Sequence Learning

Übersicht

Sequence to Sequence Learning

- Verfahren, das eine Sequenz in eine andere umwandelt
- verwendet ein RNN bzw. LSTM oder GRU, um das Problem verschwindender Gradienten zu reduzieren
- Trainiert wird üblicherweise auf der Cross-Entropy Loss Funktion
- Wesentliche Elemente sind ein Encoder- und Decoder-Netzwerk

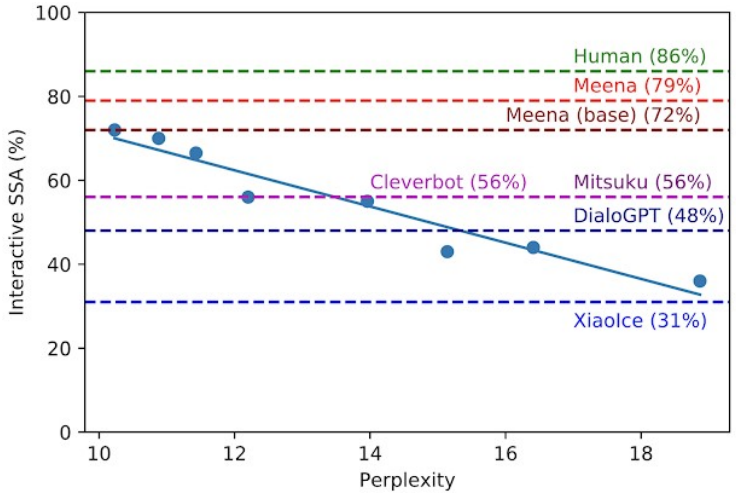
Anwendungsbereiche überwiegend im Bereich **Natürliche Sprachverarbeitung (NLP)**

- Maschinelle Übersetzung
- Textzusammenfassung
- Konversationsmodelle (z.B. Chatbots)
- Bild Captioning

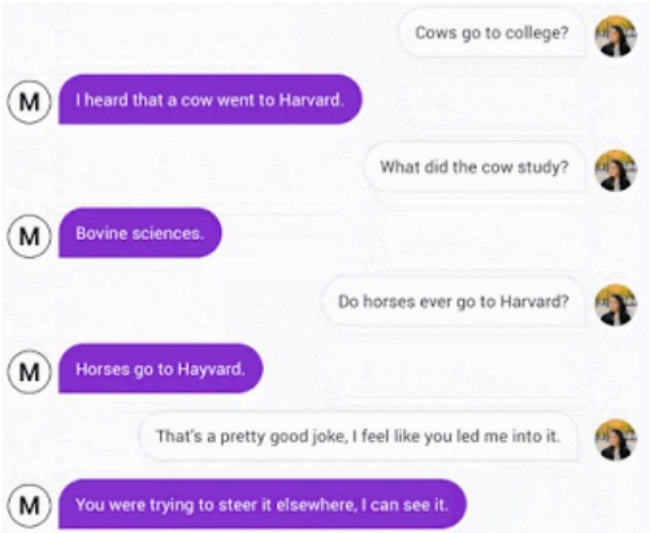
Korrelation von menschlicher mit Chatbot-Metrik

Sensibleness and Specificity Average (SSA)

- Messung grundlegender Eigenschaften menschlicher Unterhaltungen.
- Beispiel: **„Ich mag Tennis“**
  - **„Das ist toll!“:** vernünftige, aber keine spezifische Antwort
  - **„Ich auch, ich kann von Roger Federer gar nicht genug bekommen“:** vernünftig und spezifisch



Interactive SSA vs. Perplexity. Each blue dot is a different version of the Meena model. A regression line is plotted demonstrating the strong correlation between SSA and perplexity. From: Google AI Blog



Beispielunterhaltung mit Meena (2020)

Perplexität (Verwirrung)

- Messung Unsicherheit eines Sprachmodells (LLM).
- Je niedriger die Perplexität, desto höher die Vorhersagegenauigkeit des nächsten Tokens.
- Weitere Beispiele von Evaluationsmetriken: Cross-Entropy, Präzision, Recall, F1-Score

Transformer

Transformer vs. RNN

Transformer sind eine **Neuronale Netzwerk Architektur,** die auf einem Deep Learning Modell mit in Serie geschalteten **Encodern und Decodern** basiert. Dafür kommen nicht, wie bisher, Recurrent Neural Networks oder Convolutional Neural Networks zum Einsatz.

- Anwendungen hauptsächlich im Natural Language Processing *(NLP)*
- Eingeführt in 2017 im Paper *„Attention Is All You Need“*

Transformer	**RNN Herausforderungen**
• Ermöglichen Long-Range Abhängigkeiten • Kein Gradient Vanishing und Explosion • Weniger Trainingsschritte • Keine Rekurrenz, weshalb parallele Berechnungen möglich sind	• Long-Range Abhängigkeiten • Gradient Vanishing und Explosion • Große Zahl an Trainingsschritten • Rekurrenz verhindert parallele Berechnungen (z.B. GPUs)

Grundarchitekturen vortrainierter Machine-Learning Modelle

- BERT: Bidirectional Encoder Representations from Transformers
- GPT: Generative Pretrained Transformer

Beispiel Übersetzung

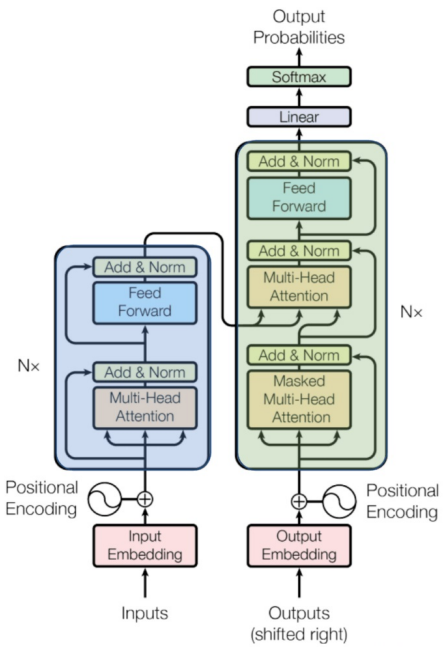
Übersetzung Englisch-Deutsch

- Input: Englischer Satz
  - z.B. „The quick brown fox“
  - Aufteilung in z.B. ein Token pro Wort d.h. 4 Tokens
- Output: Deutscher Satz

Die Vorgehensweise für **Chatbots** ist analog:

- Input: User-Utterance (User-Äußerung/Anfrage)
- Output: Antwort

Transformer Architektur



Attention is all you need, Vaswani et al. 2017

Transfomer bestehen aus in Serie geschalteten **Encodern** *(links in blau)* und **Decodern** *(rechts in grün)*. Nx: mehrfach aufeinandergestapelt.

Ein **Encoder** besteht aus einem

1. Self-Attention Layer
2. Feedforward Layer

→ wandelt jeden Token (=Item der Eingabesequenz) in einen Hidden Vektor *(=interne Repräsentation)* um, der den Token selbst sowie seinen Kontext *(=Bedeutung)* abstrakt abbildet.

Der **Decoder** besteht aus

1. Self-Attention Layer
2. Encoder-Decoder-Attention Layer
3. Feedforward Layer

→ kehrt den Prozess um und verwendet die interne Repräsentation als Input und übersetzt sie in eine Ausgabesequenz.

Attention

- Optimierungsansatz im Seq2Seq Learning
- Berechnung der Korrelation eines Eingabe Items zu den anderen Eingabe Items (z.B. Zuordnung des Pronomens 'ihm' → 'Sean')
- Ermöglicht dem Decoder die selektive Betrachtung der Input Sequenz, die den gesamten Kontext speichert

Verarbeitung

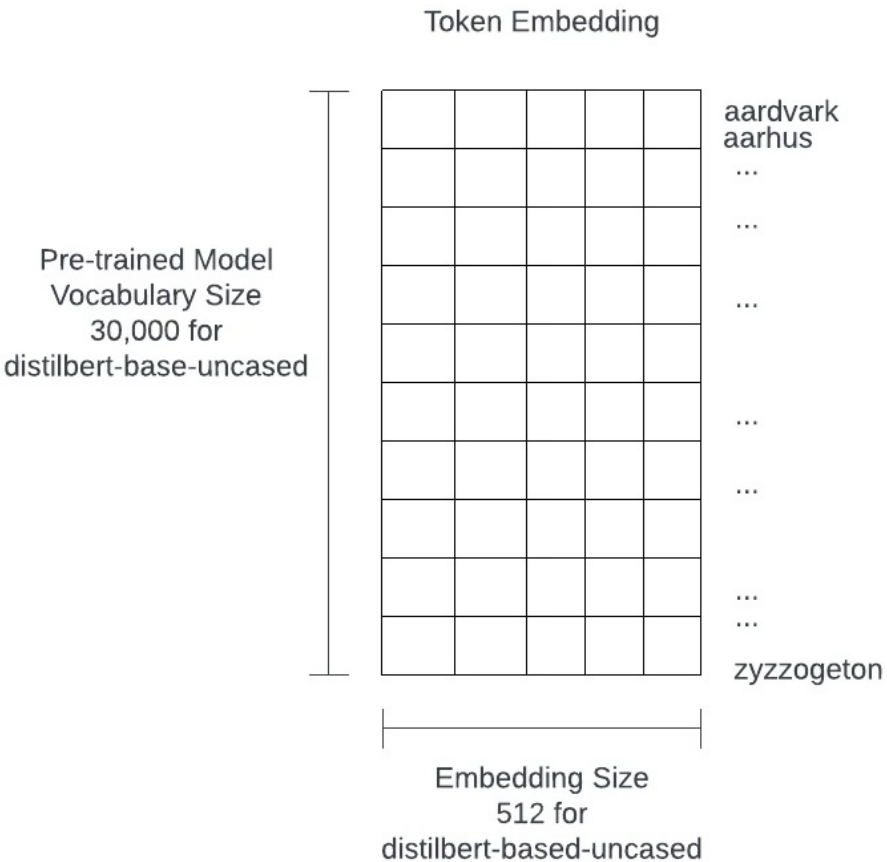
- Die Eingabesequenz wird in **Batches** verarbeitet, wobei die Länge der Encoder-Decoder-Pipeline die maximale Länge der Eingabesequenz (z.B. 512 Tokens) beschränkt.
- Je nach Größe des Netzwerks können beispielsweise einzelne Sätze oder auch ganze Absätze verarbeitet werden.
- Bei Eingabesequenzen, welche kürzer sind als die Länge der Kodierer-Dekodierer-Pipeline, wird Padding verwendet, um die Eingabesequenz aufzufüllen.

Encoder

Encoder-Schritt 1: Input Embedding

Tensor n-ter Stufe

- **Tensor:** Anordnung von Zahlen entlang n Achsen.
- **Beispiele:** $n = 0$ Skalar, $n = 1$ Vektor, $n = 2$ Matrix,...



Tensor 2. Stufe im Beispiel BERT-Embedding
(30.000 x 512 Matrix)

- **Erste Tensor-Dimension** bildet die Anzahl Vokabeln (ca. 30.000 Stück) ab.
- **Zweite Tensor-Dimension:** bildet die Embedding- (bzw. *Hidden*-) Größe ab und entpricht der Anzahl trainierbarer Gewichte für jeden Token im Vokabular (standardmäßig 512).

Beispieleinträge der Matrix
Das Wort 'dog' steht im BERT Vokabular der Python Bibliothek *transformers* an Stelle 3899 ($\Rightarrow$ Token ID=3899)

```
>>> print(embedding_layer.weight[3899])
tf.Tensor([ -0.01493477 0.01244431 ...
-0.04449853 -0.01931076 0.02335184 ],
shape=(512,), dtype=float32)
```

Jeder Input Token erhält eine Vektor-Darstellung, bevor er vom Transformer weiter verarbeitet werden kann.

Encoder-Schritt 2: Positionskodierung

Durch **Positionskodierung** wird die sequentielle Abfolge der Wörter berücksichtigt. Ein Wort erhält somit zu Beginn eines Satzes eine andere Repräsentation als am Ende.

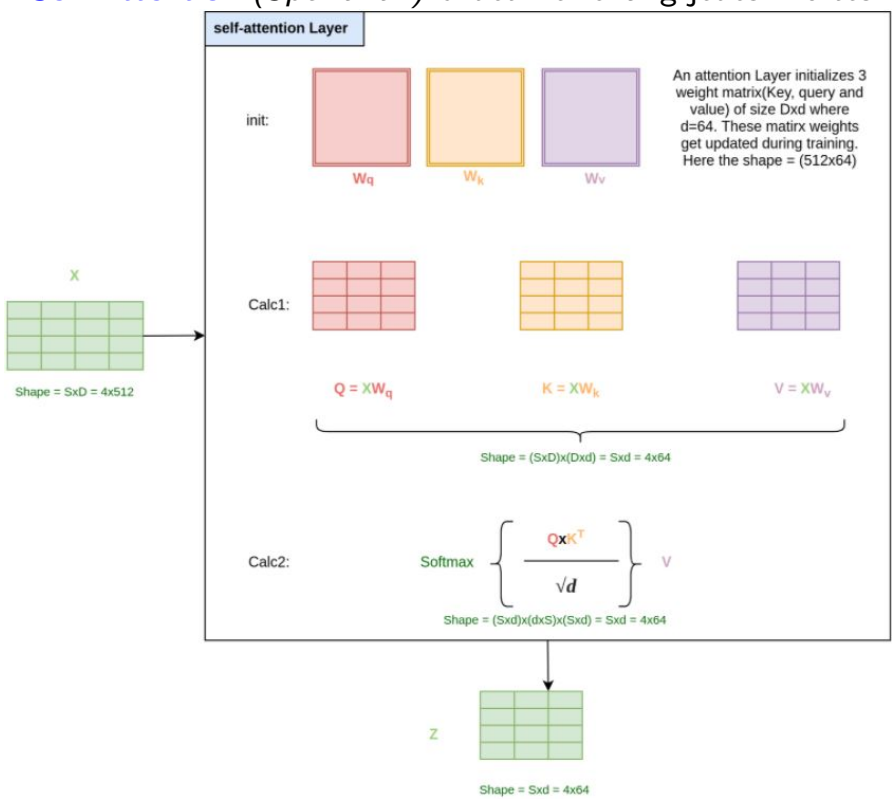
Encoder-Schritt 2: Die Gleitkommazahlen aus dem vorherigen Input Embedding werden mit denen des Position Embeddings aufaddiert, um den Encoder Input zu erzeugen.

Encoder-Schritt 3: Attention

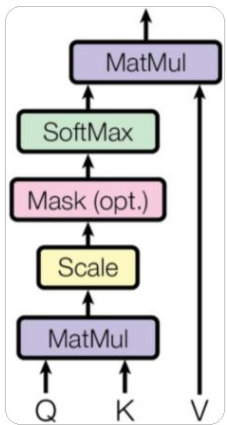
Attention gibt an, wie stark jedes Wort der durch die Query *(Q)* dargestellten Sequenz von allen anderen Wörtern der durch Key *(K)* dargestellten Sequenz beeinflusst wird:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \ softmax \in (0, 1)$$

Self-Attention *(Spezialfall)* bildet Beziehung jedes Wortes innerhalb eines Satzes zu den anderen Wörtern im selben Satz ab



Schematischer Ablauf der mathematischen Operationen im Self-Attention Mechanismus mit den Matrixprodukten Q, K und V aus Rechenschritt 1 (Calc1) als Input.

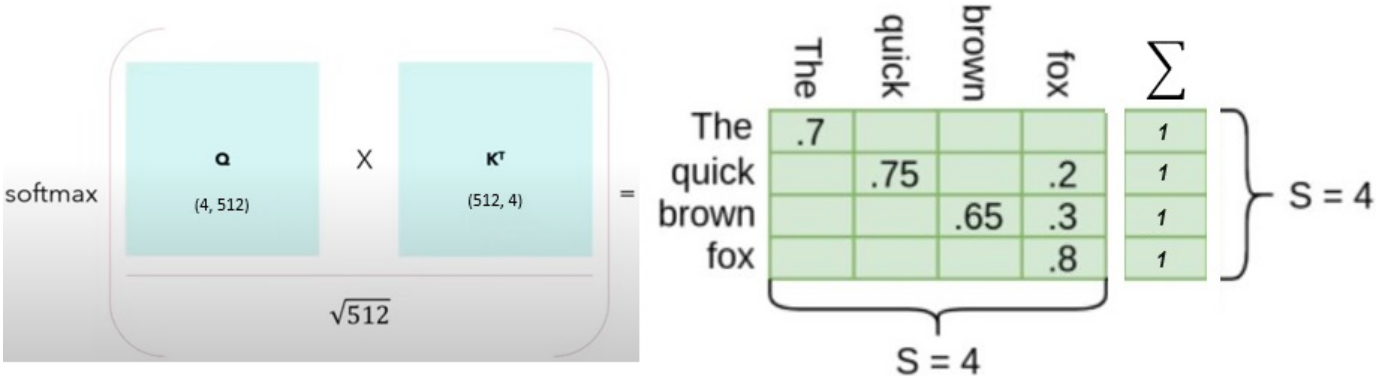


- X: Input
- S: Länge des Input-Satzes, z.B. „The quick brown fox“
- D: Embedding Size *(Default: 512)*, dessen Gewichte mit dem Netzwerk trainiert werden.

- Die erste Matrixmultiplikation *MatMul* in Rechenschritt 2 (Calc2) kann als **multiplikative Ähnlichkeitsmessung** von Q und K^T interpretiert werden.
- Für jede **Q**uery lernt das Modell somit, welchen **K**ey-**V**alue Input es ansteuern soll.

Multiplikative Ähnlichkeitsmessung

Die berechnete QK^T-**Matrix** mit SxS-Form stellt den Beitrag jedes Wortes zu einem anderen Wort innerhalb desselben Satzes dar:



Diagonaleinträge sind groß, d.h. die Wortbeiträge zu sich selbst sind hoch. Das Wort 'quick' trägt zu 'quick' und 'fox' bei. Beide Wörter 'quick' und 'brown' tragen zum Wort 'fox' bei. Zeilensummen sind auf 1 normiert.

Self-Attention Interpretation

Der Embedding Vektor des Wortes 'quick' wird effektiv durch das 0.75-fache des 'quick'-Embeddings und 0.2-fache des 'fox'-Embeddings ersetzt.

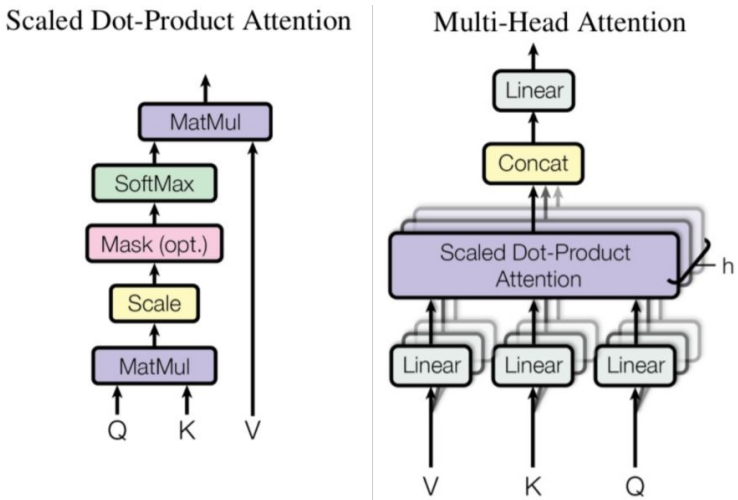
Die Zeileneinträge der resultierenden *Attention*-**Matrix** *(Z)* repräsentieren somit:

- *Encoder Schritt 1:* **Bedeutung** des Wortes (s. Input Embedding)
- *Encoder Schritt 2:* **Position** des Wortes (s. Kodierung)
- *Encoder Schritt 3:* **Beziehung** des Wortes zu allen anderen Wörtern innerhalb desselben Satzes (s. Self-Attention)

Single-Head Self-Attention vs. Multi-Head Attention

Single-Head Self-Attention *(„Scaled Dot-Product Attention")*

- learn one type of relationship



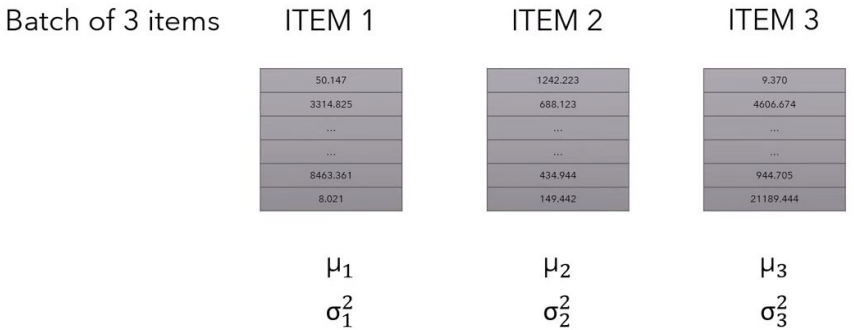
Multi-Head Attention

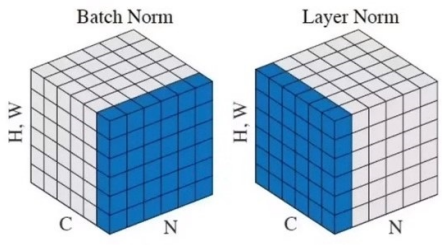
- Multiple different sets of key, query and value matrices
- consists of several attention layers running in parallel
- each attention module can focus on calculating different types of relationships between inputs and create specific contextualized embeddings.
- A single attention block can tell a model to pay attention to something specific such as the tense in a sentence.
- Adding multiple attention blocks allows the model to pay attention to different linguistic elements such as **part of speech, tense, nouns, verbs.**
- Embeddings are concatenated and put through ordinary linear neural network layer, together making the final output of the so-called Multi-Headed Attention Module.

Encoder Schritt 4: Layer Normalisierung

Layer Normalisierung

Stellt die Vergleichbarkeit der Feature-Verteilungen für alle Neuronen eines Layers her.





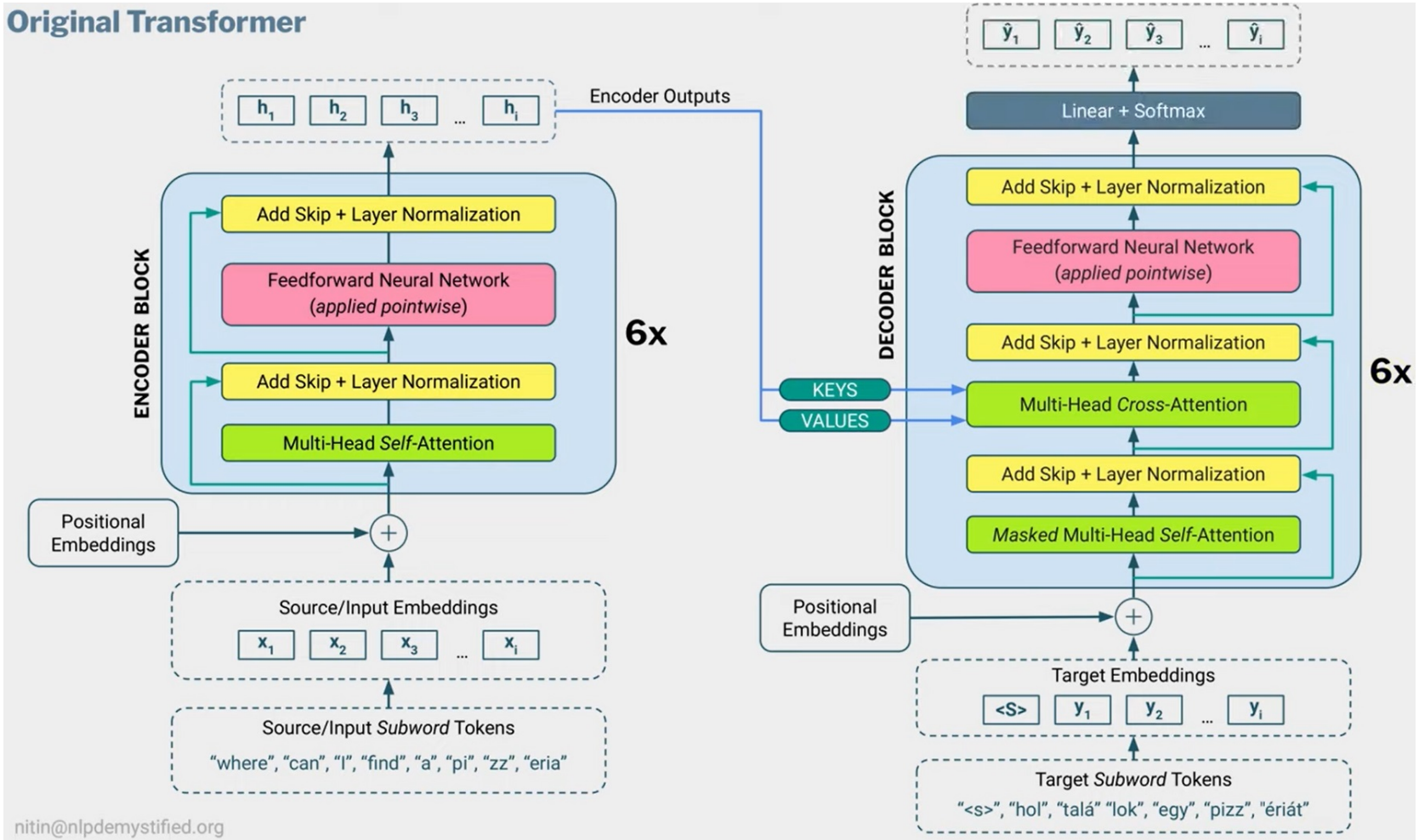
Normalisierungsgleichung

$$y = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \otimes \gamma + \beta$$

- μ Mittelwert der Dimensionen D
- σ^2 Varianz der Dimensionen D
- ϵ kleiner Wert (Gleitkommazahl), der kleine σ kompensieren kann
- γ (multiplikativ) und β (additiv) sind Parameter in Tensorform, die für Fluktuationen in den Daten notwendig sind und während des Trainings gelernt werden.

Decoder

Übersicht Transformer Architektur



Decoder: Maskierung von Multi-Head Attention

Kausales Modell

Der Output von einem Wort an einer bestimmten Position hängt nur von vorherigen Wörtern ab.

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	~~0.119~~	~~0.134~~	~~0.148~~	~~0.179~~	~~0.152~~
CAT	0.124	0.278	~~0.201~~	~~0.128~~	~~0.154~~	~~0.115~~
IS	0.147	0.132	0.262	~~0.097~~	~~0.218~~	~~0.145~~
A	0.210	0.128	0.206	0.212	~~0.119~~	~~0.125~~
LOVELY	0.146	0.158	0.152	0.143	0.227	~~0.174~~
CAT	0.195	0.114	0.203	0.103	0.157	0.229

Die Softmax Funktion wandelt minus unendlich in null um, damit das Modell zukünftige Wörter nicht sehen kann.

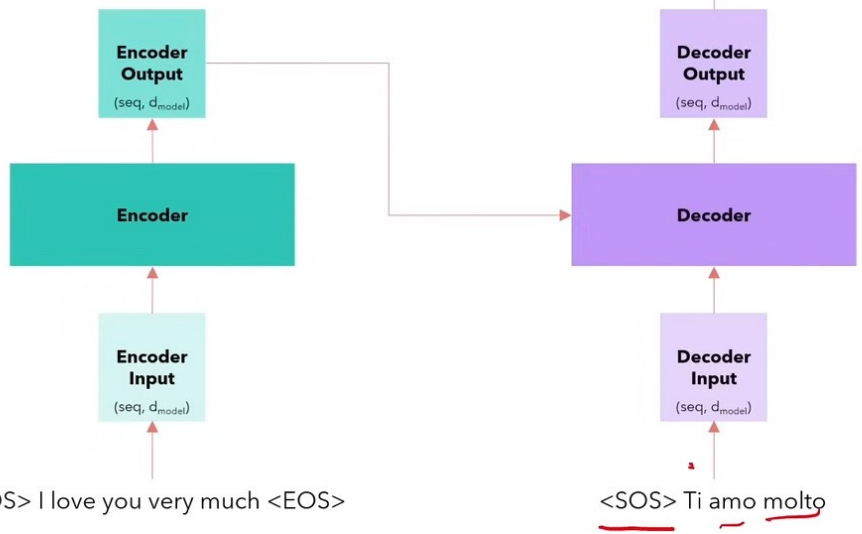
Transformer training: One sequence in one time step!

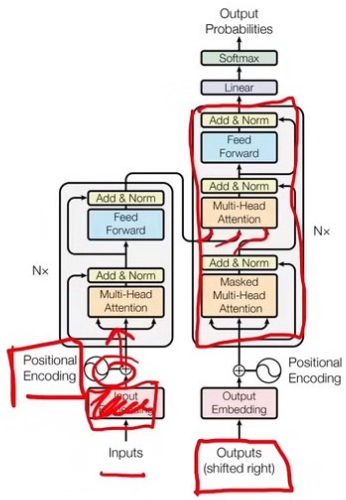
Training

Time Step = 1

It all happens in one time step!

The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.



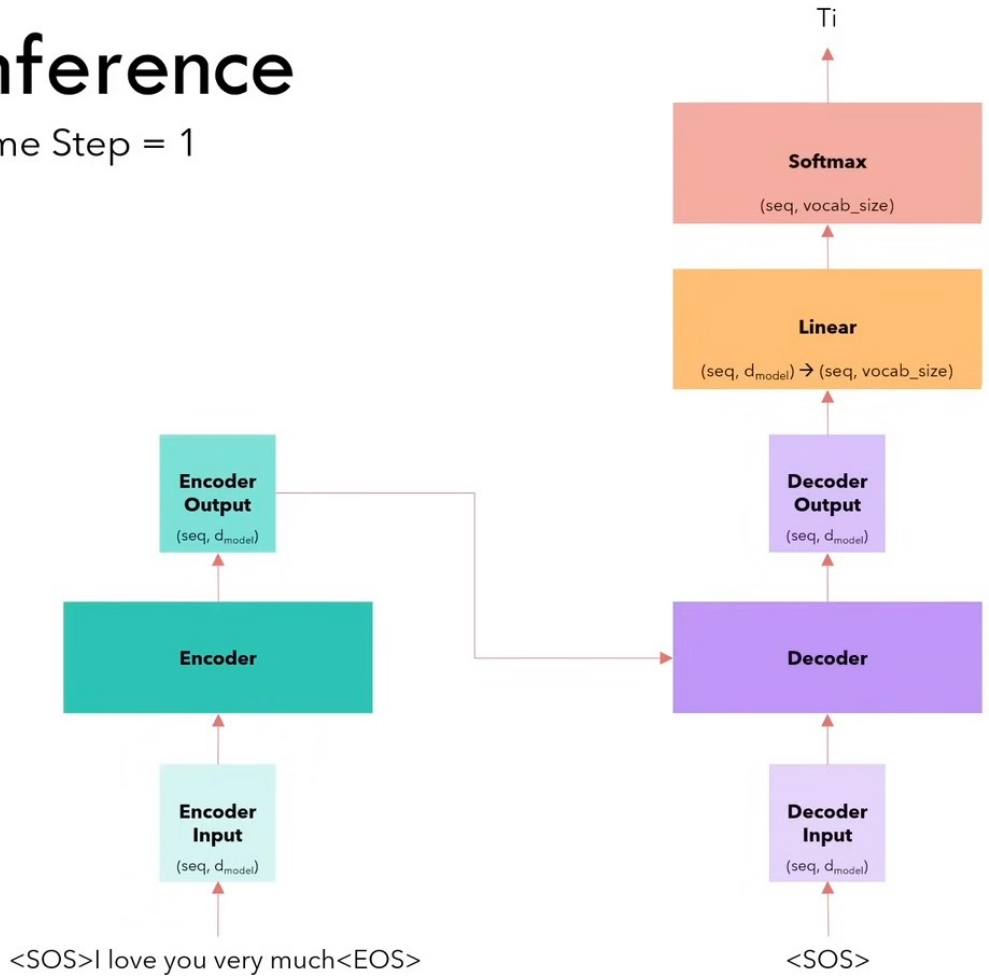


We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

Transformer inference: One token in one time step

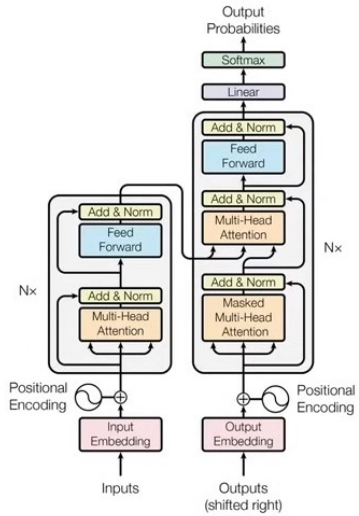
Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum value.

The output of the last layer is commonly known as **logits**



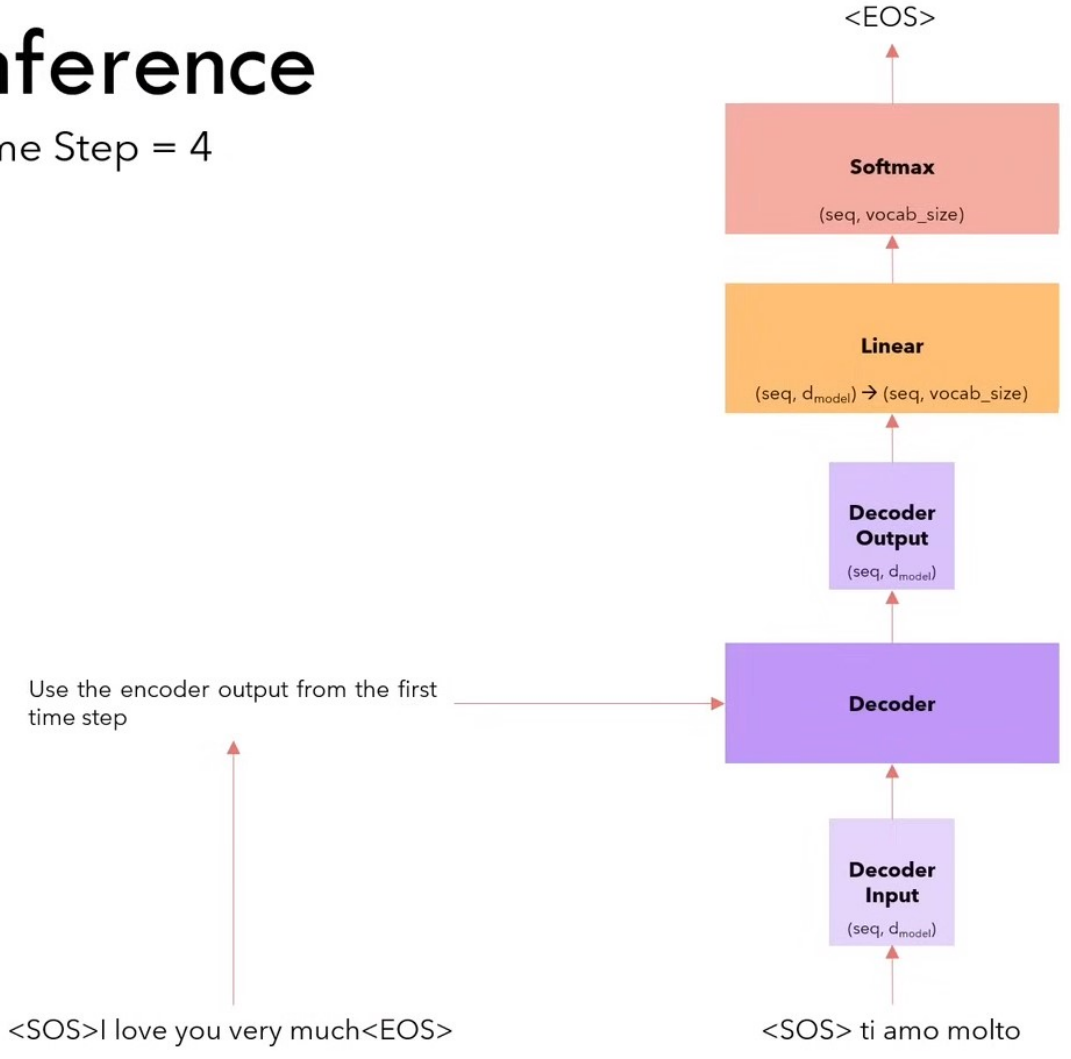
* Both sequences will have same length thanks to padding

3 more inference timesteps...

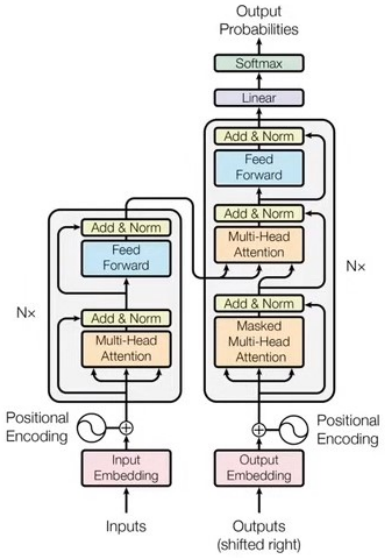
Transformer inference: Last time step

Inference

Time Step = 4



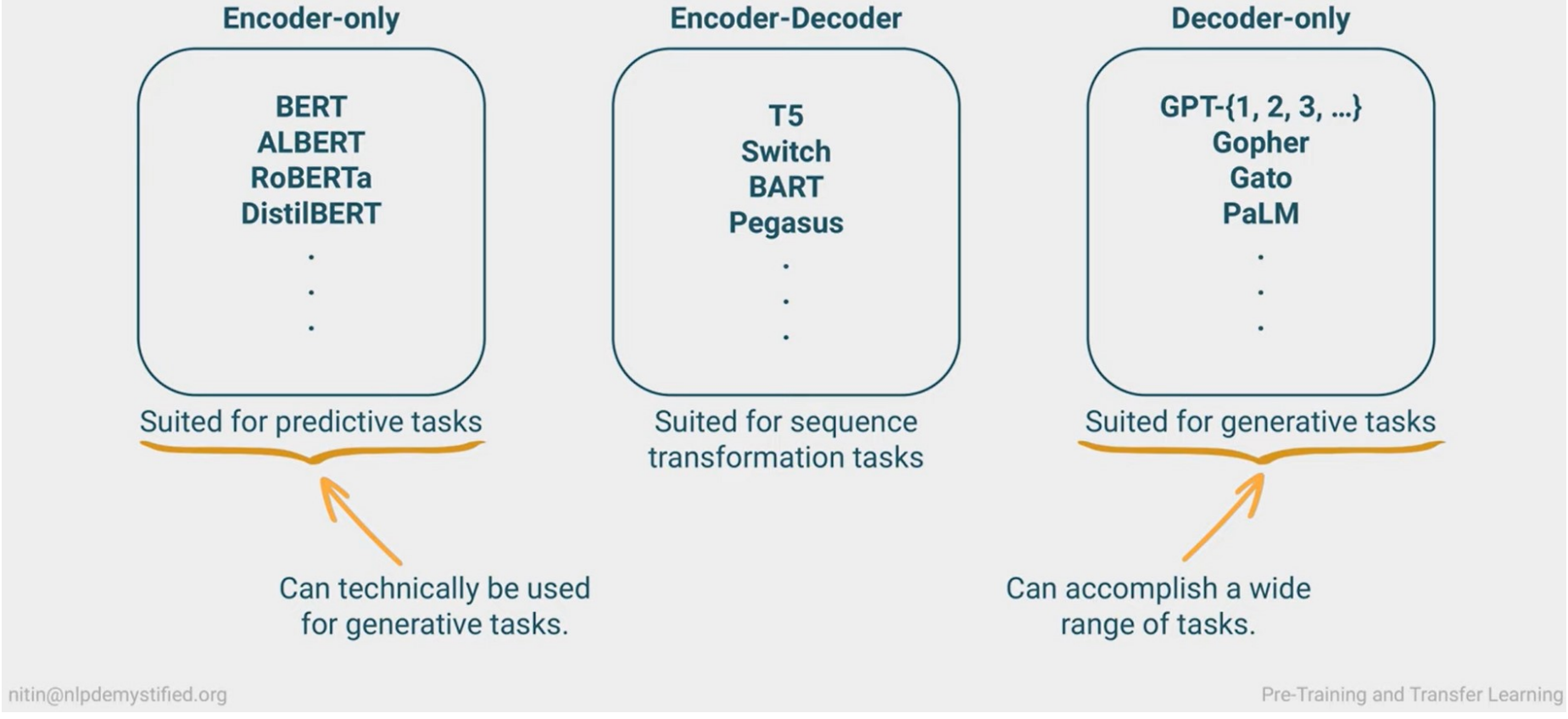
Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.



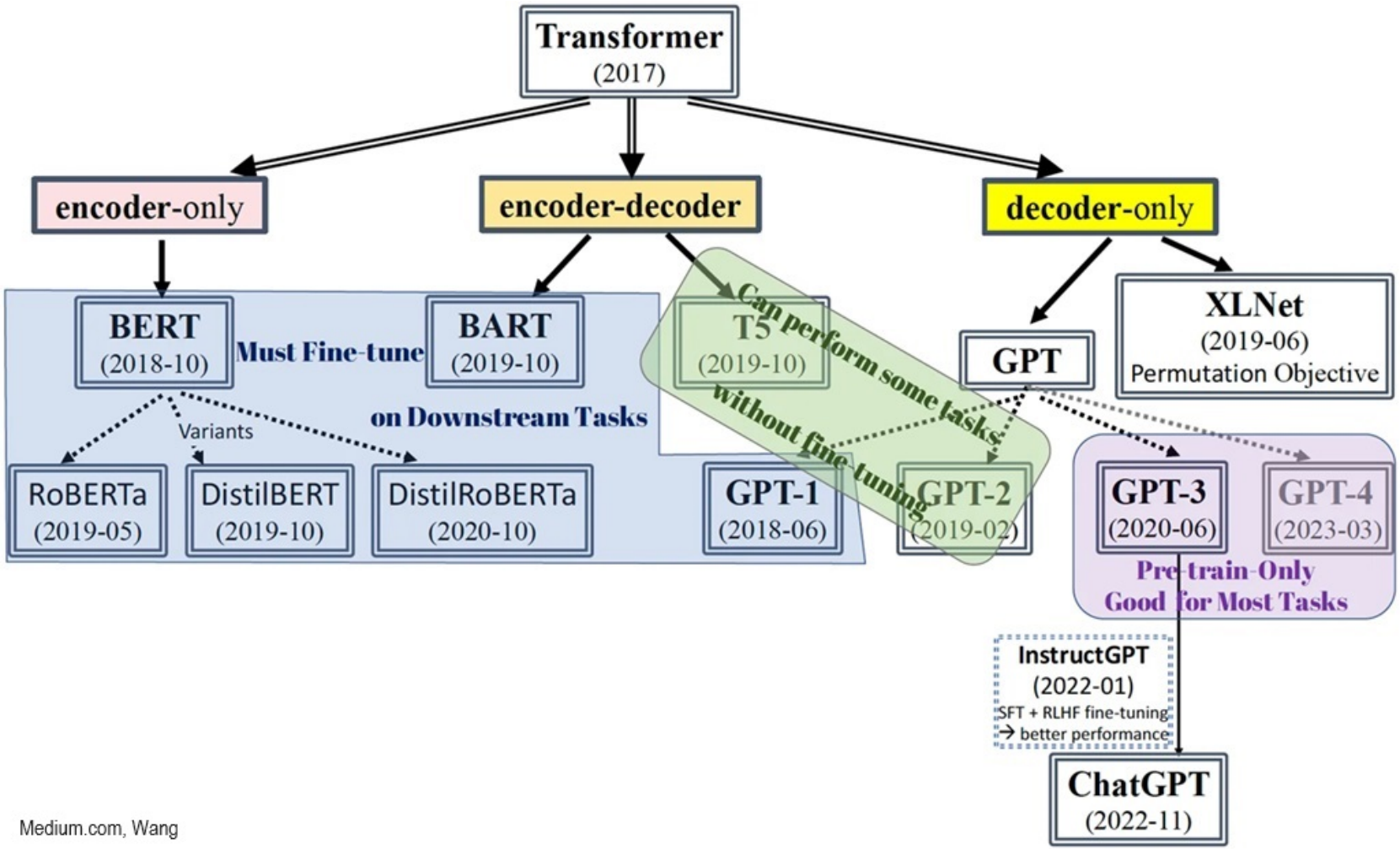
Append the previously output word to the decoder input

Überblick Transformer Modelle

Transfomer based models overview



Transfomer based models overview



Transformer-based models graph
The graph illustrates models of different architectures — encoder-only (autoencoding AE), decoder-only (autoregressive AR), and encoder-decoder models, and whether they require fine-tuning on downstream task-specific datasets.

GPT Exam Results