Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Attention and Transformers

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 11

1. Motivation for Attention
2. Intuition for Attention
3. Improving RNNs with Attention
4. Transformers: Overview
5. Self-Attention
6. Transformer Encoder & Decoder Architecture
7. Transformers: Putting it All Together

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Motivation for Attention

Frank Hutter Abhinav Valada

University of Freiburg

Introduction

- Often, the inputs to a neural network are very high-dimensional
  - Some parts of the input are more important than others
  - Which parts are relevant depends on the task at hand

- We can learn to attend to the parts of the input data that are most relevant for a task

Motivation: Attention in Humans

- Given a lot of input information, humans only attend to parts of it
- Visual attention: what would you attend to in this driving scene?
  - The fovea of the eye gives us high-acuity vision in only a tiny region of our field of view

[Image source: https://www.flickr.com/photos/29487672@N07/9708030604/, under cc license]

Motivation: Attention in Humans

- Given a lot of input information, humans only attend to parts of it
- Auditory attention: focusing on a single conversation at a cocktail party

[Image source: https://www.pinterest.com/pin/97882991876338392/]

Motivation: Attention in Humans

- Given a lot of input information, humans only attend to parts of it
- Humans can even attend to parts of their memory
  - Example: we still remember some events of our childhood
  - Example: what did we cover in the first week of this course?

Attention is a Very Powerful Core Concept in Deep Learning

- Attention is the key element of Transformers
- Attention is used in most state-of-the-art networks
- Attention facilitates the interpretability of neural networks (to some degree)

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Which three examples of attention in humans did we discuss?

- Discussion:
  Can you think of applications that intuitively require attention to obtain good performance?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Intuition for Attention

Frank Hutter Abhinav Valada

University of Freiburg

Implicit Attention in Artificial Neural Networks

- A trained neural network reacts more strongly to some parts of the input than to others
- We can quantify this by the gradient of a class probability output w.r.t. the input pixels
  - E.g., $\frac{\partial \hat{p}(dog)}{\partial \text{pixel}_i}$: how much does the probability of predicting "dog" change with pixel i?
  - This allows us to compute a saliency map with a single step of backpropagation:

Figure: Visualizing implicit attention with a single backprop through a trained CNN

[Image source: Simonyan et al., 2014]

Explicit Attention

- Explicit attention mechanisms actively focus on parts of the input data and downweight other inputs

Figure: Not all parts of the input image are equally important for classification.

[Image source: Yu et al., 2019]

Explicit Attention

- In explicit attention, the network computes input-dependent attention weights
  - These weights are the output of a computation, not to be confused with trainable parameters
  - The attention mechanism to compute these weights is learned jointly with the classifier
- We have actually seen explicit attention mechanisms before
  - E.g., the input gate in LSTMs depends on the input and determines how much the input affects the LSTM's state
- Explicit attention can have various benefits:
  - Computational efficiency (faster training)
  - Scalability (e.g., high resolutions)
  - Interpretability (what parts of the input does the network focus on)
- In this lecture, we will focus on such explicit attention mechanisms

Hard Attention vs. Soft Attention

- Hard attention: binary decision what to pay attention to
  - Advantage: computational savings
  - Disadvantage: hard to train since not differentiable
- Soft attention: continuous weights for all inputs
  - Advantage: directly trainable via gradient descent
  - Disadvantage: can be computationally costly; but lots of research for making this cheaper
  - This is the type of attention we'll discuss throughout the lecture

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What are the differences between implicit and explicit attention?
- Repetition:
  What are the differences between hard and soft attention?
- Discussion:
  Why don't we just have a static mask over the input that serves as "attention"?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

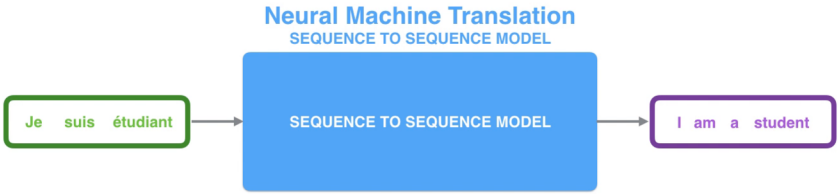
Improving RNNs with Attention

Frank Hutter Abhinav Valada

University of Freiburg

Attention for Improving Gated RNNs

- Before the current Transformer architectures, the best methods for many language tasks were gated RNNs (such as the LSTM)
- An exemplary language task is neural machine translation (NTM)
  - This translates a sequence of words into another sequence of words



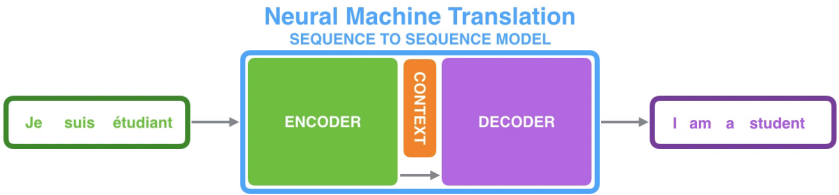
- Classical sequence to sequence models sequentially read inputs and produce outputs

[Animation source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Encoder-Decoder Network

Sequence to sequence models are typically composed of an encoder and a decoder

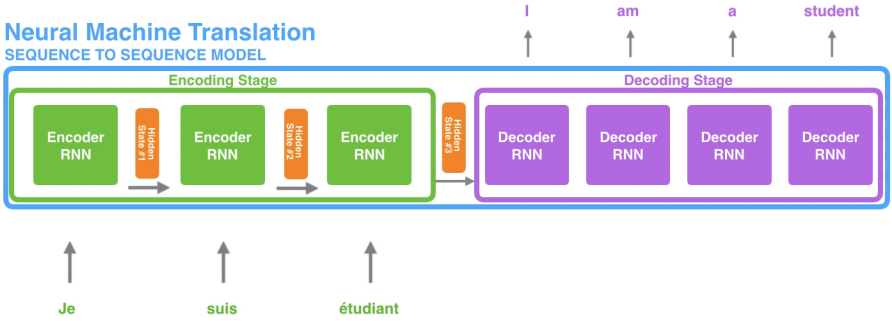
- The encoder (often an RNN) processes the input sequence into a context vector
- The decoder (often an RNN) takes this context vector and produces the output sequence



[Animation source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Encoder-Decoder Seq2Seq Model

- The last hidden state of the encoding stage is passed to the decoder RNN
- Disadvantage: one vector has to encode information for all the inputs

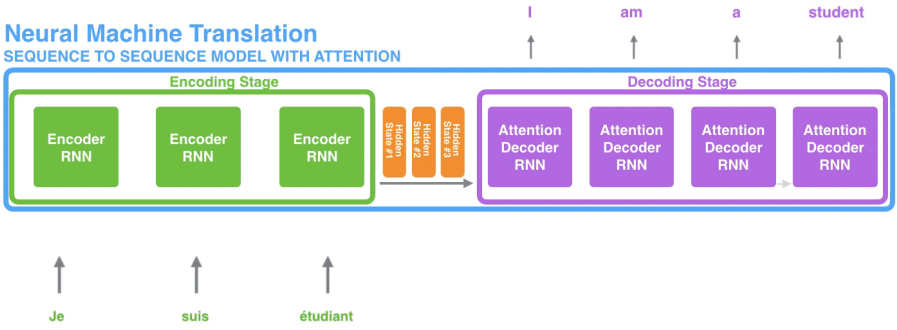


[Original Paper: Sutskever et al., 2014]

[Animation source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Seq2Seq Model with Attention

- Idea: All intermediate hidden states of the encoding stage are passed to the decoder RNN
- Decoder then learns to focus on the relevant input vectors from the encoder



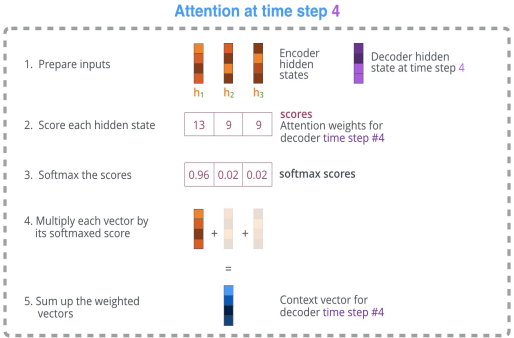
[Original Paper: Bahdanau et al., 2015]

[Animation source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Decoder Attention

The decoder "attends" to parts of the input that are relevant to this decoding step

- It computes a score for each of the hidden states it received
  - This depends on their similarity to the decoder's hidden state
- It multiplies each hidden state by its soft-maxed score
  - This amplifies hidden states with high scores and drowns out those with low scores



[Animation source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/]

Worked Example from the Literature (with a Bi-Directional RNN)

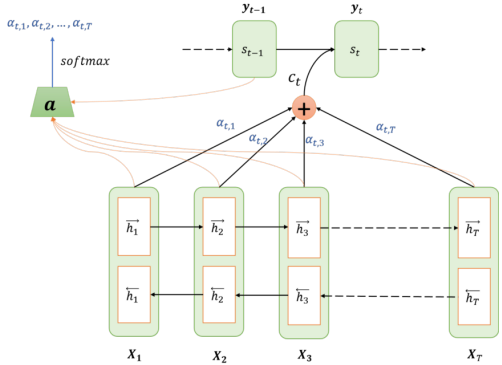


Figure: Seq2Seq Bi-directional Recurrent Network with Attention

Decoder has run for $t-1$ steps and is now going to produce output for time step t

Encoder computes hidden states h_j corresponding to each input X_j

[Original Paper: Bahdanau et al., 2015]
[Image source: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e]

Encoding

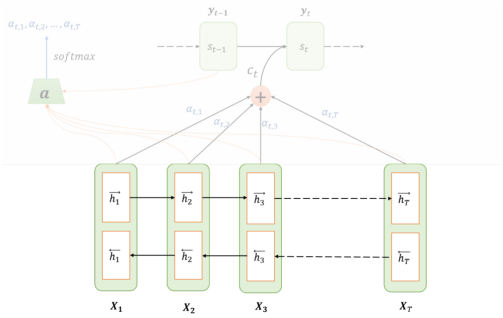


Figure: Focusing on bi-directional encoder

Notation:

$(X_1, X_2, \ldots X_T)$: Input sequence, where T is the length of the sequence

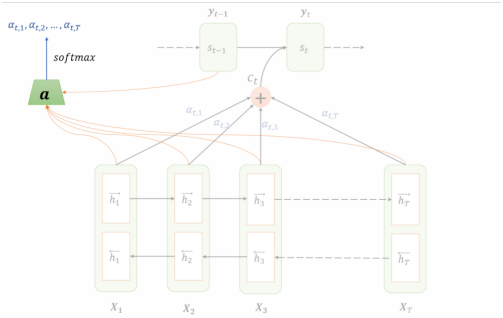
$(\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots \overrightarrow{h_T})$: Hidden states of the forward RNN

$(\overleftarrow{h_1}, \overleftarrow{h_2}, \ldots \overleftarrow{h_T})$: Hidden states of the backward RNN

$h_j = [\overrightarrow{h_j}; \overleftarrow{h_j}], \forall j \in [1, T]$

[Original Paper: Bahdanau et al., 2015]
[Image source: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e]

Calculating the Attention Weights



We compute a score for each encoder hidden state h_j

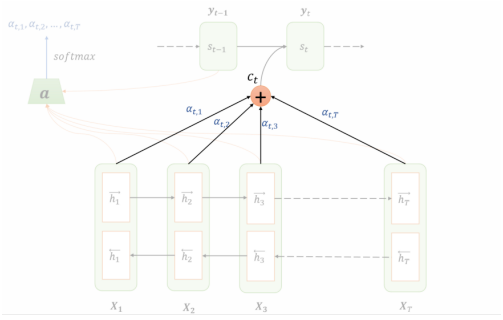
- This is dependent on our current decoder hidden state s_t
- Here, we use a feedforward neural network α for this

$$e_{tj} = \alpha(h_j, s_{t-1}), \forall j \in [1, T]$$

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}$$

[Original Paper: Bahdanau et al., 2015]
[Image source: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e]

Calculating the Context Vector

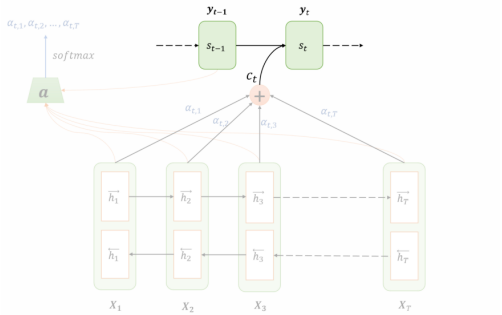


α_{tj} is the hidden input-dependent weight for hidden encoder state h_j

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$

[Original Paper: Bahdanau et al., 2015]
[Image source: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e]

Decoding Stage



The next decoder state s_t then depends on:

- the previous decoder state s_{t-1}
- the previous output y_{t-1}
- the context vector c_t

$$s_t = f(s_{t-1}, y_{t-1}, c_t)$$

In the original paper by Bahdanau et al. f was based on a GRU

[Original Paper: Bahdanau et al., 2015]
[Image source: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What is the role of the encoder and what is the role of the decoder?

- Repetition:
  Does the decoder calculate attention scores at every time step or only when it initially receives the encoder hidden states? Why?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Transformers: Overview

Frank Hutter Abhinav Valada

University of Freiburg

Introduction

- Transformers were introduced in the paper "Attention is All You Need" [Vaswani et al, 2017]
- They strongly improved the state of the art in machine translation
  - Before Transformers, recurrent networks with attention had performed best

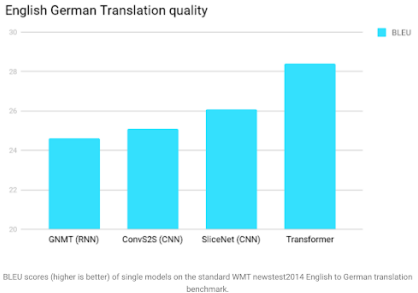


Figure: Results comparing transformers on the prominent WMT dataset (machine translation task)

[Image source: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html]

The Rise of Transformers in Deep Learning

- Transformers have become one of the most popular types of neural networks
- Transformer-based language models (e.g. BERT, GPT-3) have revolutionized NLP
- Transformer-based architectures also achieve strong performance for other modalities
  - E.g., Computer Vision (ViT)
  - E.g., Speech Recognition (Conformer)

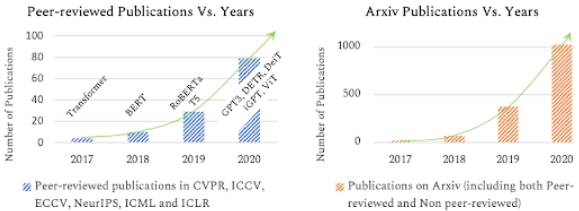


Figure: Steep increase in the number of transformer-based papers over the years

[Image source: Khan et al., 2021]

Transformers Remove the Need for Recurrent Networks (RNNs)

- RNNs sequentially handle inputs
- Transformers handle all inputs in one step

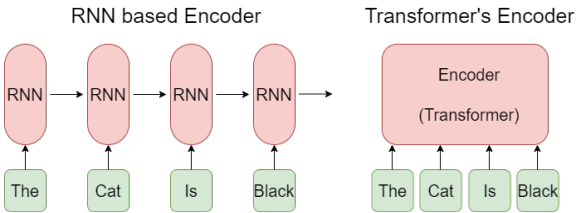


Figure: The transformer gets rid of recurrence

High-level Transformer Architecture

- A stack of encoder blocks (x6), followed by a stack of decoder blocks (x6)

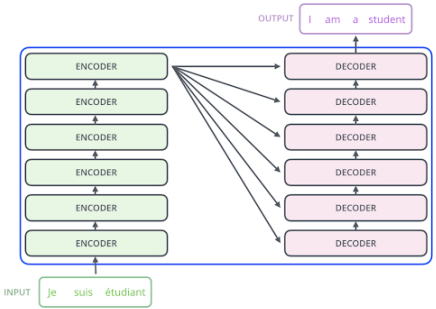


Figure: High-level overview of the transformer encoder-decoder architecture, for a neural machine translation task (French → English).

[Image source: https://jalammar.github.io/illustrated-transformer/]

Main Components of the Encoder and Decoder Blocks

- Each encoder block consists of 2 main components:
  - Self-Attention Layer
  - Fully Connected Layer
- Each decoder block consists of 3 main components:
  - Self-Attention Layer
  - Encoder-Decoder Attention Layer
  - Fully Connected Layer

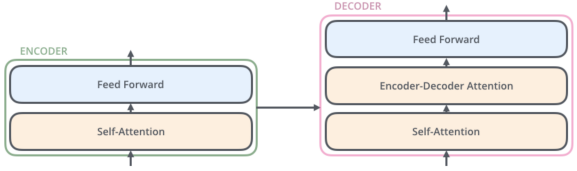


Figure: Elements of the encoder and decoder blocks

[Image source: https://jalammar.github.io/illustrated-transformer/]

Flow of Vectors Through the Encoder

- For each input word: encode into vector of size d (e.g., $d = 512$)
- Each encoder block takes an input vector of size d and outputs a vector of size d

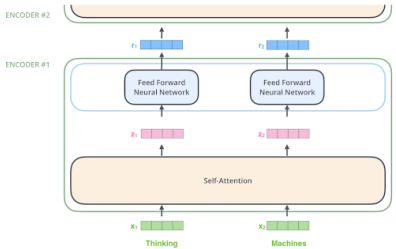


Figure: Flow of vectors through the transformer's encoder

- The self-attention block operates on all inputs jointly
- The feedforward block operates on each word separately
- The representation of each word gets transformed to take into account the entire sentence

[Image source: https://jalammar.github.io/illustrated-transformer/]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Which encoder layers is each of the Transformers' decoder layers connected to?
- Repetition:
  Draw a transformer's high-level encoder-decoder architecture, including the size of the vectors at each stage.
- Discussion:
  What are the recent breakthroughs you have heard about that are based on the transformer architecture?
- Discussion:
  We have seen that RNNs can take a sequence of variable lengths as input. How do transformers deal with this? Can you think of any limitations?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Self-Attention

Frank Hutter Abhinav Valada

University of Freiburg

Attention as Soft Retrieval from a Database

Let's say we have a list of (key, value) pairs and a query q

key_1	value_1
key_2	value_2
key_3	value_3
.	.
.	.
.	.
key_N	value_N

- Database retrieval: compare q to keys and return value of exact match

$$db\_retrieval(q, \boldsymbol{k}, \boldsymbol{v}) = \sum_{i=1,\ldots,N} \mathbb{I}(q, \boldsymbol{k}_i) \times \boldsymbol{v}_i$$

- Attention: compare q to keys and return weighted average of values

$$\begin{aligned} attention(q, \boldsymbol{k}, \boldsymbol{v}) &= \sum_{i=1,\ldots,N} a_i \times \boldsymbol{v}_i \\ &= \sum_{i=1,\ldots,N} Softmax(Similarity(q, \boldsymbol{k}_i)) \times \boldsymbol{v}_i \\ &= \sum_{i=1,\ldots,N} \frac{\exp[Similarity(q, \boldsymbol{k}_i)]}{\sum_{j=1,\ldots,N} \exp[Similarity(q, \boldsymbol{k}_j)]} \times \boldsymbol{v}_i \end{aligned}$$

Different Similarity Functions Used To Compute Attention

Ways to compute $Similarity(q, \boldsymbol{k}_i)$

- $q^\top \boldsymbol{k}_i$ dot product
- $q^\top \boldsymbol{k}_i / \sqrt{d}$ scaled dot product; d is the dimensionality of each key
- $q^\top \boldsymbol{W} \boldsymbol{k}_i$ generalized dot product
- $\boldsymbol{w}_q{}^\top \boldsymbol{q} + \boldsymbol{w}_k{}^\top \boldsymbol{k}_i$ additive similarity

Putting the 'Self' Into Self-Attention

- We'll attend to the other words in the same sentence
  - I.e., we construct queries, keys and values from the same sequence of inputs

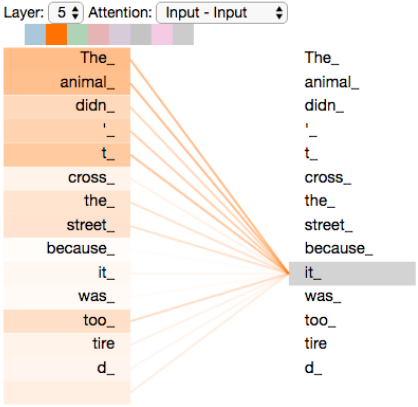


Figure: For each word, we attend to the other words of the sentence

Obtaining Keys, Values and Queries from a Sentence

- For each input $\boldsymbol{x}_i$, we compute query, key and value vectors by linear mappings
  - There are thus three trainable weight matrices $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V$
  - E.g., multiplying $\boldsymbol{x}_i$ by $\boldsymbol{W}^Q$ produces $\boldsymbol{q}_i$, the query vector associated with that word

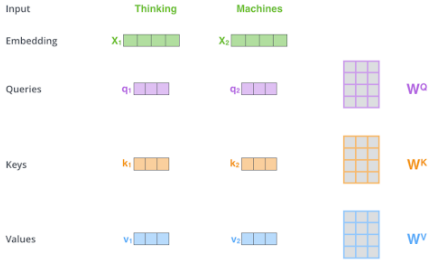


Figure: A self-attention block includes 3 trainable weight matrices ($\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V$), that linearly transform inputs $\boldsymbol{x}_i$ to yield $\boldsymbol{q}_i$, $\boldsymbol{k}_i$ and $\boldsymbol{v}_i$. The attention scores are then calculated from these vectors

[Image source: https://jalammar.github.io/illustrated-transformer/]

Computing Self-Attention Weights

- Transformers use scaled dot-product attention: $Similarity(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^\top \boldsymbol{k}_i / \sqrt{d}$
- So if we're processing self-attention for query input $\boldsymbol{x}_1$:
  - Score for first input: $\boldsymbol{q}_1^\top \boldsymbol{k}_1 / \sqrt{d}$
  - Score for second input: $\boldsymbol{q}_1^\top \boldsymbol{k}_2 / \sqrt{d}$
- After we have the scores, compute softmax to get values in [0,1]

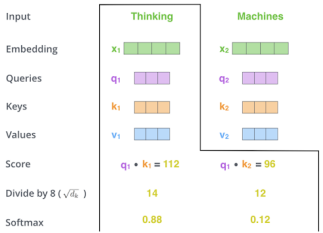


Figure: Computing the attention weights for the query "Thinking"

[Image source: https://jalammar.github.io/illustrated-transformer/]

Final Self-Attention Output

- Multiply the softmaxed scores by the inputs' respective values
- This determines how much each word will be expressed at each position
- E.g., output of the self-attention layer for the first position:

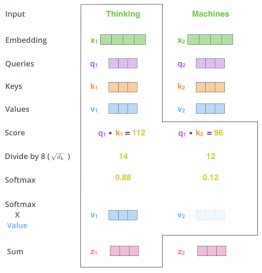


Figure: Computing the transformed representation of the input "Thinking"

[Image source: https://jalammar.github.io/illustrated-transformer/]

Self-Attention in Matrix Form

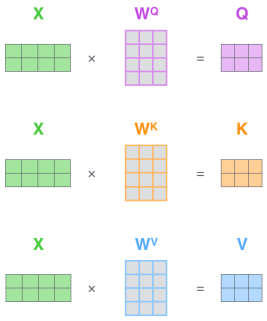


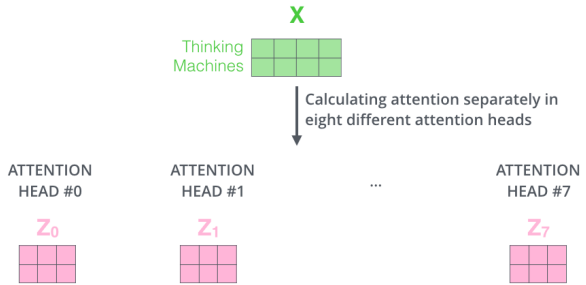
Figure: Computing query, key and value matrices

Figure: Computing the final self-attention

[Image source: https://jalammar.github.io/illustrated-transformer/]

Multi-head Self-Attention: Attending to More than One Concept

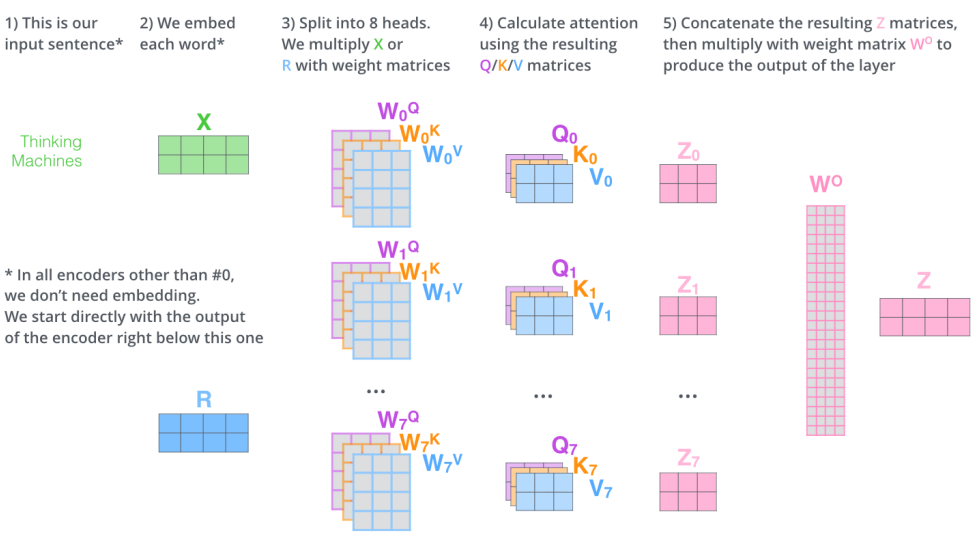
- We might want to attend to more than one concept at a time
  - E.g., different meanings of 'it' in "The chicken didn't cross the road because it was too wet."
  - Using a single attention would force us to average over concepts



- In the original paper: embedding dimension $d = 512$; for each attention head: $d/8 = 64$

[Image source: https://jalammar.github.io/illustrated-transformer/]

Multi-head Self-Attention: Attending to More than One Concept



[Image source: https://jalammar.github.io/illustrated-transformer/]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What does the "self" in self-attention refer to?

- Repetition:
  Write down the formula for $attention(q, \boldsymbol{k}, \boldsymbol{v})$ with scaled dot-product attention.

- Discussion:
  Draw a diagram visualizing the matrices involved in multi-head self-attention and discuss the individual computation steps and matrix dimensions.

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Transformer Encoder & Decoder Architecture

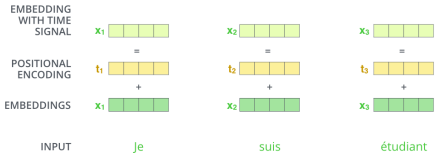
Frank Hutter Abhinav Valada

University of Freiburg

Taking Into Account the Position of Inputs in the Sequence

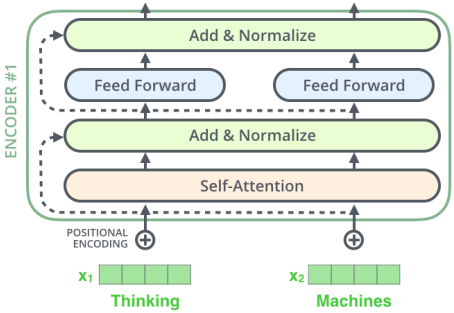
- Unlike RNNs, self-attention does not consider the order of the inputs
  - For some applications with set-valued inputs, that is actually desirable
  - However, for many tasks the order of the sequence is important
    - E.g., machine translation
    - E.g., language modelling (predicting the next word)
    - E.g., protein folding
    - E.g., RNA folding

- Positional encodings allow Transformers to take into account order information
  - For each input position: fixed encoding of the position
  - This fixed encoding is simply added to the usual learnable input embedding



[Image source: https://jalammar.github.io/illustrated-transformer/]

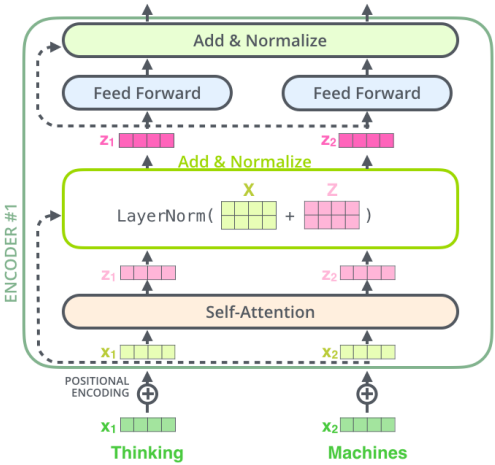
The Transformer's Encoder Block



- There are skip connections around both self-attention & feed forward layers
  - As in ResNets, these are followed by normalization (here: layer norm)
  - Each layer only needs to learn residual changes of a word's representation
- Self-attention is followed by position-wise feed-forward layers
  - The weights of the feed forward layers in the same encoder are shared

[Image source: https://jalammar.github.io/illustrated-transformer/]

The Transformer's Encoder Block: Details

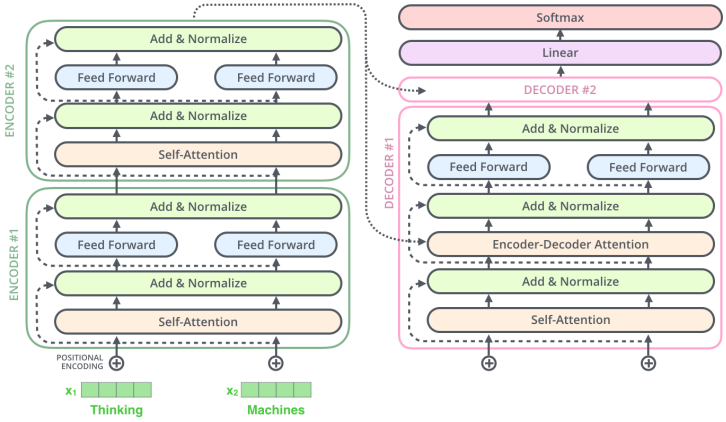


- Detail: the layer norm happens after the residual connection

[Image source: https://jalammar.github.io/illustrated-transformer/]

The Transformer's Encoder-Decoder Architecture

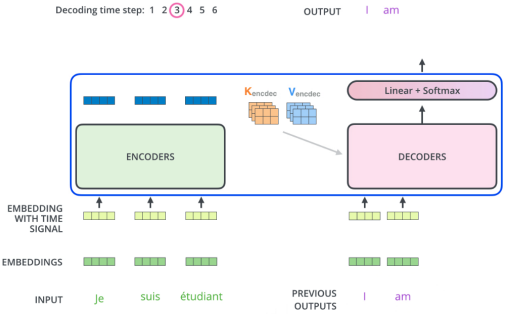
- Encoders and decoders are very similar
- Decoders also have encoder-decoder attention layers



[Image source: https://jalammar.github.io/illustrated-transformer/]

The Decoder is a Conditional Model

- It takes a prefix of an output sentence as input and predicts the next word



  - It begins with a start token
  - It stops decoding when it generates an end token as an output

- Its self-attention layer is only allowed to attend to earlier positions in the output sequence
  - This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation

[Image source: https://jalammar.github.io/illustrated-transformer/]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Where are skip connections located in Transformers? For a Transformer with 5 encoder blocks and 7 decoder blocks, how many skip connections are there?

- Repetition:
  What types of attention are used in Transformers and where?

- Repetition:
  In which ways does a Transformer's decoder architecture differ from its encoder architecture?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 11: Attention and Transformers

Transformers: Putting it All Together

Frank Hutter Abhinav Valada

University of Freiburg

Recap of Scaled Dot Product Attention and Multi-head Attention

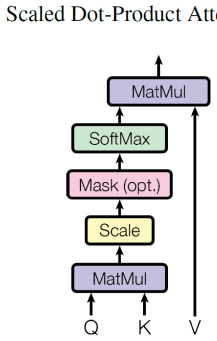


Figure: Scaled dot product attention as visualized in the original paper introducing Transformers

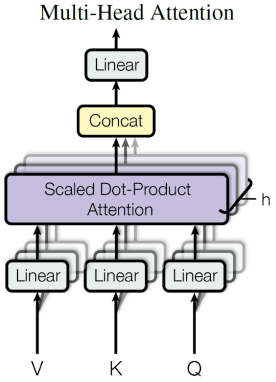


Figure: Multi-head attention as visualized in the original paper introducing Transformers

[Image source: Vaswani et al., 2017]

The Complete Transformer Architecture

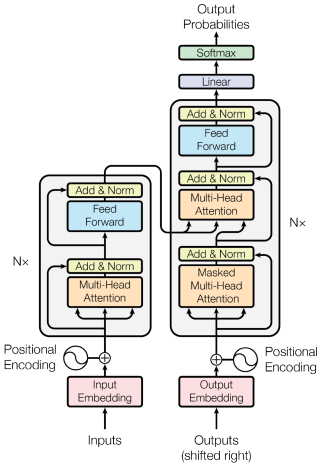


Figure: The entire Transformer architecture as visualized in the original paper introducing Transformers

[Image source: Vaswani et al., 2017]

One Issue of Standard Transformers: Complexity of the Attention Matrix

- The (self-)attention matrix specifies how much each input attends to each other input
- For n inputs, the complexity is thus $\mathcal{O}(n^2)$

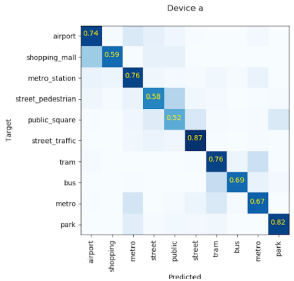


Figure: A typical self-attention matrix; the diagonal usually has high values

- Much recent work aims to reduce this quadratic complexity by approximations

[Image source: Wang et al., 2018]

Limitations of Transformers and Ways to Overcome Them

- Attention cannot deal with text of arbitrary length
  - Longer text has to be split into segments/chunks before being fed into the system as input
- Low inductive bias $\rightarrow$ very data hungry
  - Often combined with self-supervised learning (or pretraining on massive data sets)
- Transformers typically need to be very large to reach good performance
  - But they parallelize nicely on modern specialized hardware (GPUs/TPUs)

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  How is the multi-head self-attention mechanism implemented based on an attention block?

- Discussion:
  For transformers, we have seen that larger is better. Apart from requiring more compute, what could be the additional limitations during training? Can you think of scenarios where these large models could not be used?

References

Bojarski, M., Choromanska, A., Choromanski, K., Firner, B., Ackel, L., Muller, U., Yeres, P., Zieba, K. (2018)
VisualBackProp: Efficient Visualization of CNNs for Autonomous Driving
2018 IEEE International Conference on Robotics and Automation (ICRA), 4701–4708
https://arxiv.org/pdf/1611.05418.pdf

Langlois, T., Zhao, H., Grant, E., Dasgupta, I., Griffiths, T., Jacoby, N. (2021)
Passive attention in artificial neural networks predicts human visual selectivity
Thirty-Fifth Conference on Neural Information Processing Systems
https://arxiv.org/pdf/2107.07013.pdf

Tang, J., Lu, Z., Su, J., Ge, Y., Song, L., Sun, L., Luo, J. (2019)
Progressive Self-Supervised Attention Learning for Aspect-Level Sentiment Analysis
ACL 2019
https://aclanthology.org/P19-1053.pdf

Zhang, A., Lipton, Z., Li, M., Smola, A. (2021)
Dive into Deep Learning
https://arxiv.org/ftp/arxiv/papers/2106/2106.11342.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I. (2017)
Attention is all you need
Advances in neural information processing systems, 5998–6008
https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wang, J., Li, S. (2018)
Self-Attention Mechanism Based System for DCASE2018 Challenge Task1 and Task4
http://dcase.community/documents/challenge2018/technical_reports/DCASE2018_Wangjun_62.pdf

Khan, S., Naseer, M., Hayat, M., Zamir, S., Khan, F., Shah, M. (2021)
Transformers in Vision: A Survey
https://arxiv.org/pdf/2101.01169.pdf

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D. (2020)
Language Models are Few-Shot Learners
https://papers.nips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Yu, F., Liu, C., Wang, Y., Zhao, L., Chen, X. (2019)
Interpreting Adversarial Robustness: A View from Decision Surface in Input Space
https://arxiv.org/abs/1810.00144

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salkhutdinov, R., Zemel, R., Bengio, Y. (2015)
Show, attend and tell: Neural image caption generation with visual attention
2048–2057
http://proceedings.mlr.press/v37/xuc15.pdf

References

Srinivas, S., Fleuret, F. (2019)
Full-gradient representation for neural network visualization
Advances in neural information processing systems,
https://arxiv.org/pdf/1905.00780.pdf

Nguyen, HM. , Kalra, G. , Kim, D. (2019)
Host load prediction in cloud computing using Long Short-Term Memory Encoder–Decoder
The Journal of Supercomputing,
https://link.springer.com/content/pdf/10.1007/s11227-019-02967-7.pdf

Simonyan, K. , Vedaldi, A. , Zisserman, A. (2014)
Deep inside convolutional networks: Visualising image classification models and saliency map
https://arxiv.org/pdf/1312.6034.pdf

Reis, E., Costa, C., Silveira, D., Baravesco, R., Righi, R., Barbosa, J., Antunes, R., Gomes, M., Federizzi, G. (2021)
Transformers Aftermath: Current Research and Rising Trends
Association for Computing Machinery, 154-–163
https://dl.acm.org/doi/pdf/10.1145/3430937

Sutskever, I., Oriol, V., Quoc V., L. (2014)
Sequence to Sequence Learning with Neural Networks
Advances in neural information processing system, 3104–3112
https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

Bahdanau, N., Cho, K., Bengio, Y. (2015)
Neural Machine Translation by Jointly Learning to Align and Translate
3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings
https://arxiv.org/pdf/1409.0473.pdf