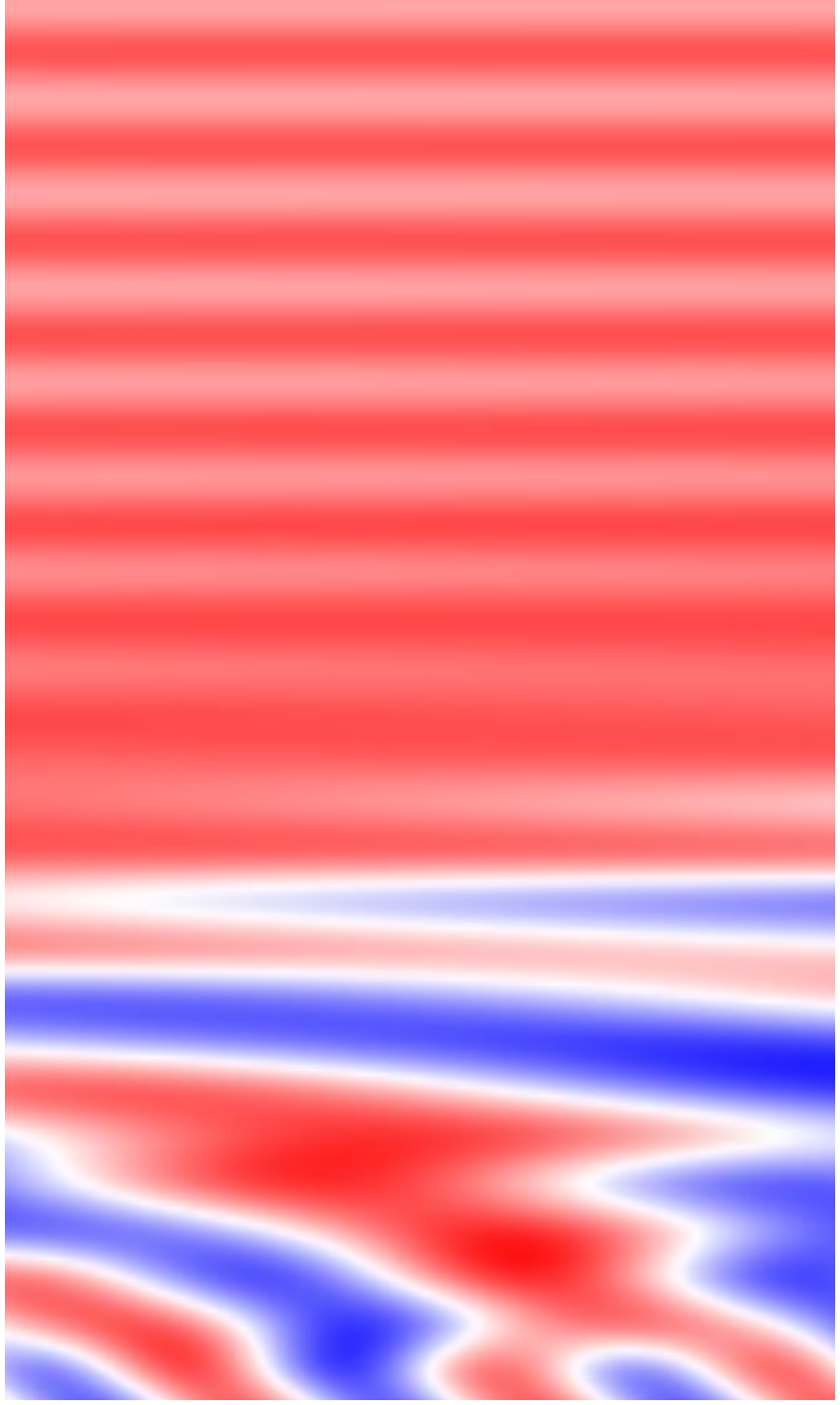


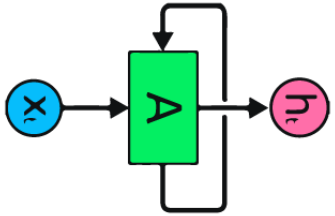
Transformers for time-series

Andrey Ustyuzhanin

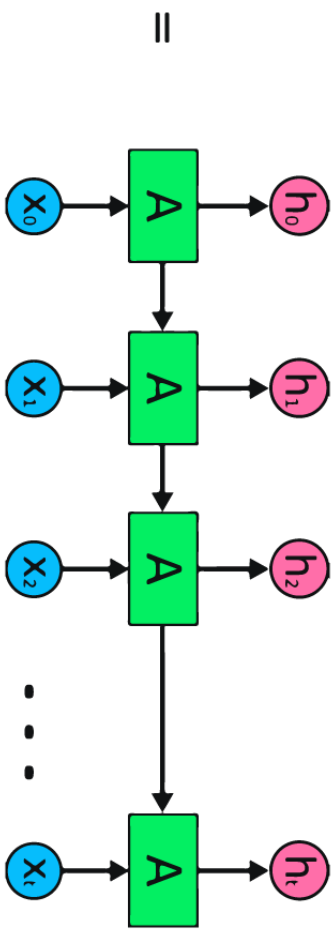
Outline

- Introduction
- Autencoders recap
- Positional encoding
- Attention mechanism
- Transformer architecture
- Transformer’s applications

Why transformers?

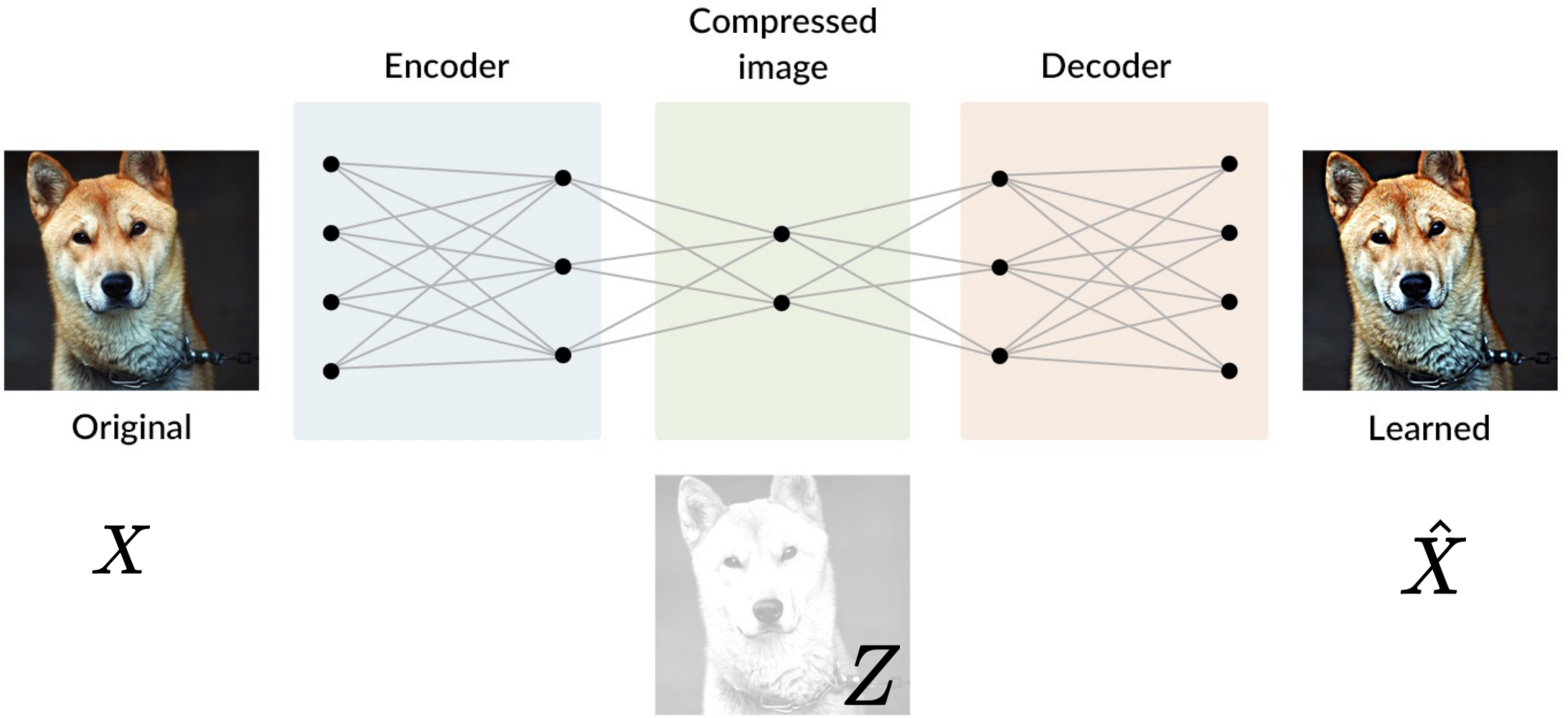


- Sequences in science: spectrums, chemical chains, hadron jet structures, measurement readings, …
  - Either: symbolic representation like protein structures
  - Or: continuous values like experimental measurements
- State of the art: RNNs, LSTMs
  - Have been around since 90s
  - Intuitive to understand once you wrap your head around time-notation
  - Are good for reasonably short sequences,
  - However, not easily to parallel for long sequence training, and building bigger and bigger models is cumbersome (intractable) due to gradient vanishing.



x → z → ẋ

Autoencoders. Recap



https://arnoldkokoroko.com/projects/imagecompress/

PCA vs Autoencoder

PCA fits linear transformation, an autoencoder (AE) fits two separate neural networks instead: a forward and an inverse transformation. Any arbitrary neural network can be used inside and AE.

AE can be applied to much more complex data like images or time-series.

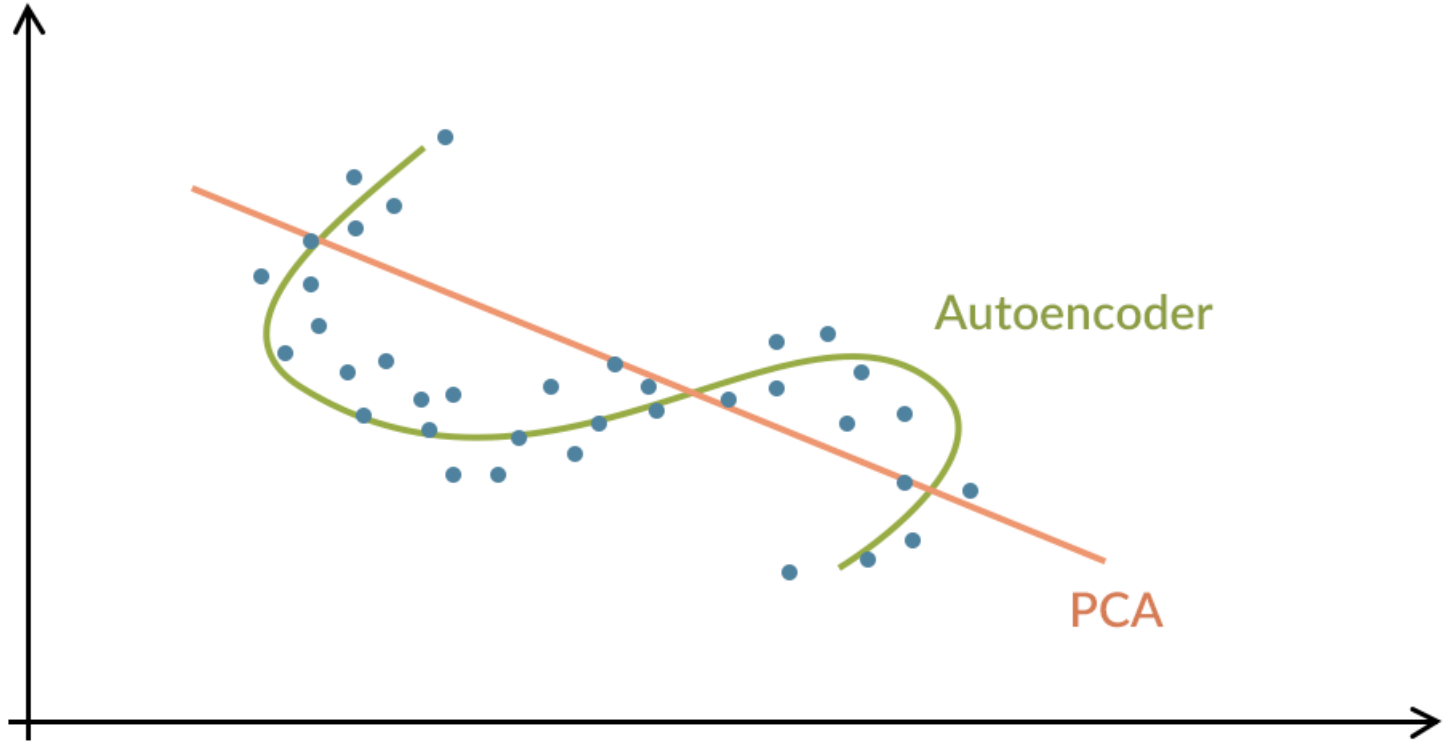
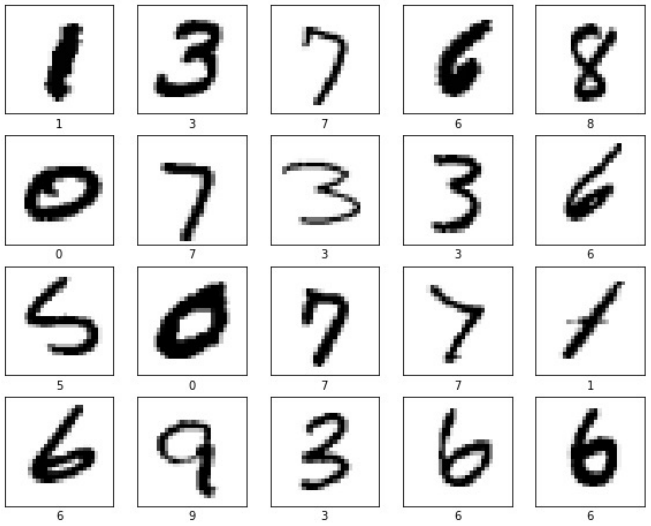
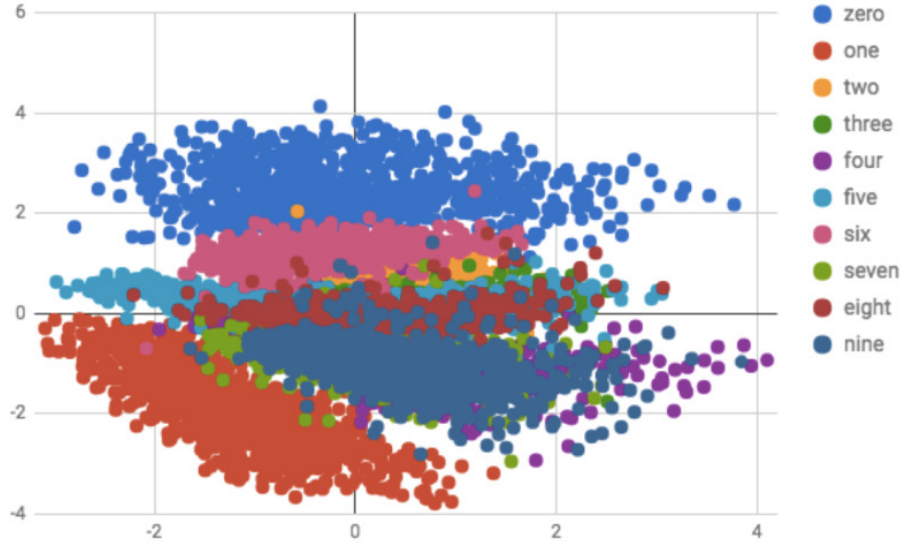


Illustration for images of digits



MNIST dataset

Compressing



2D latent space

Compressing MNIST digits to 2D and visualizing latent vectors for different digits.

Autoencoders, pros and cons

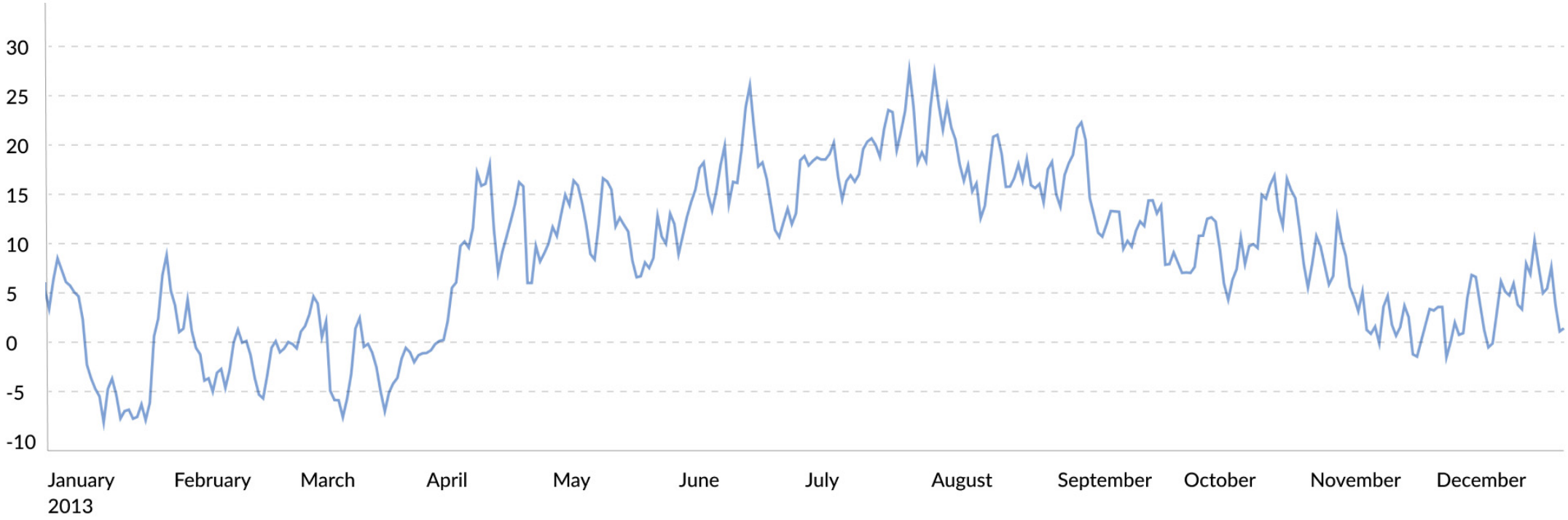
Pros

- Are good for handling large-dimensional objects such as words/sentences or images. One can plug in an arbitrary network architecture for both encoder and decoder. E.g., CNNs would be helpful for dealing with images. Similar approaches can be used for encoding words of text;
- Better for manifold learning (subspace of data that represent certain class due to the regularization built in the training procedure);
- Can be trained separately on unlabeled data or trained together with the downstream components. Notable examples: Word2Vec [Mikolov, 2013], C-BOW [Mikolov, 2013b].

Cons

- Sensitive to hyperparameter choice (e.g., Z dimensionality), thus can be more prone to overfit for large sequences;
- Latent representation can be difficult to interpret.

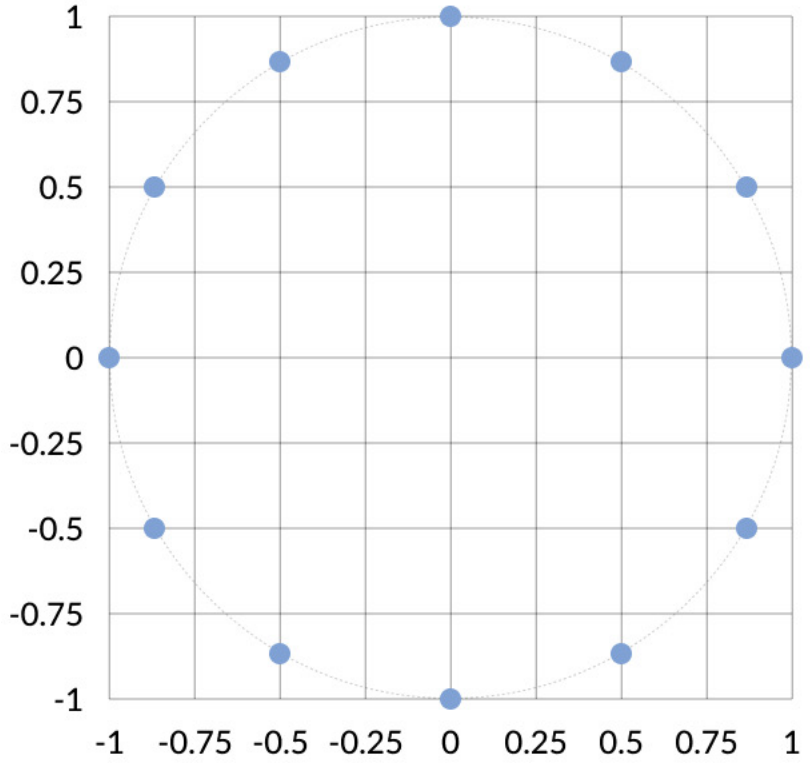
Positional data encoding



Daily temperature measurements at Jena, Germany during year 2011. One can spot a weekly temperature oscillation. How do we bring to a network?

Fixed positional encoding

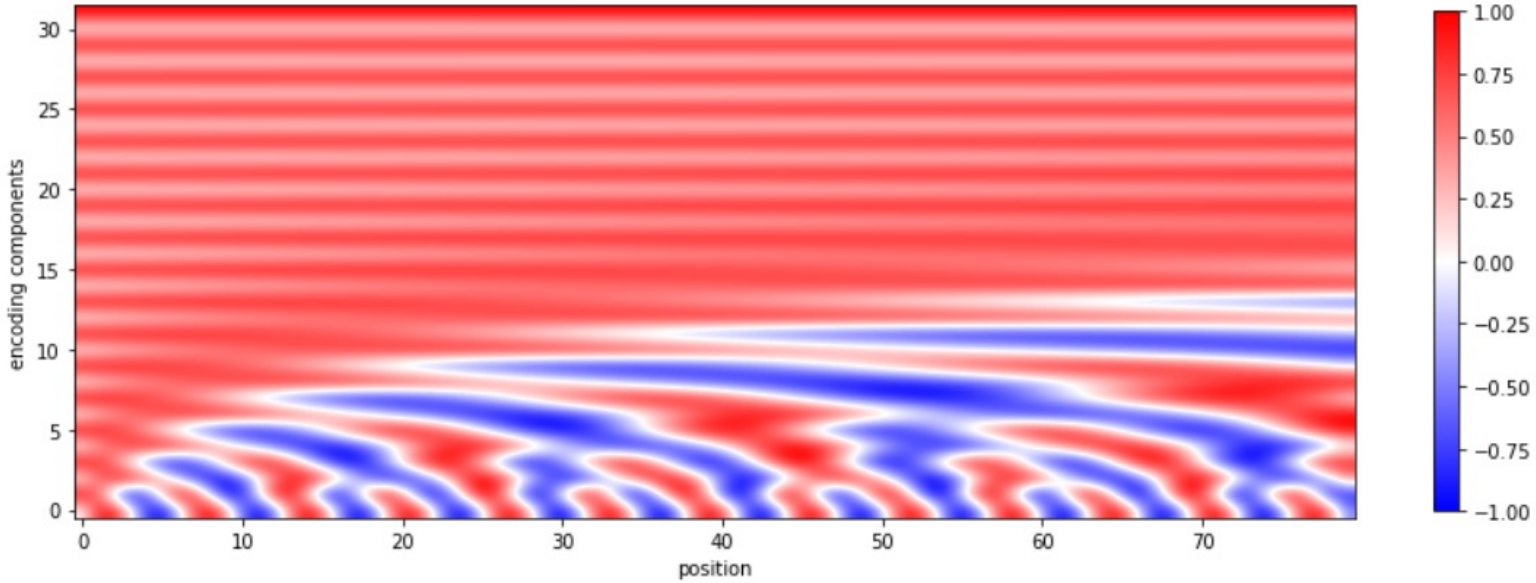
- Intuition: make all Thursdays (DOW = 3) close to Wednesdays (DOW = 2) and Fridays (DOW = 4). However, this approach won't bring Sunday (DOW=6) to Monday (DOW = 0).
- The same happens with months, i.e., encoding month by an integer number won't make December like January.
- A more advanced way to encode this distance is using a periodic function with the period equal to the maximum number of possible elements. I.e., instead of storing just a month number *m*, one can keep a 2D vector: [sin(2π *m*/12), cos(2π *m*/12)]



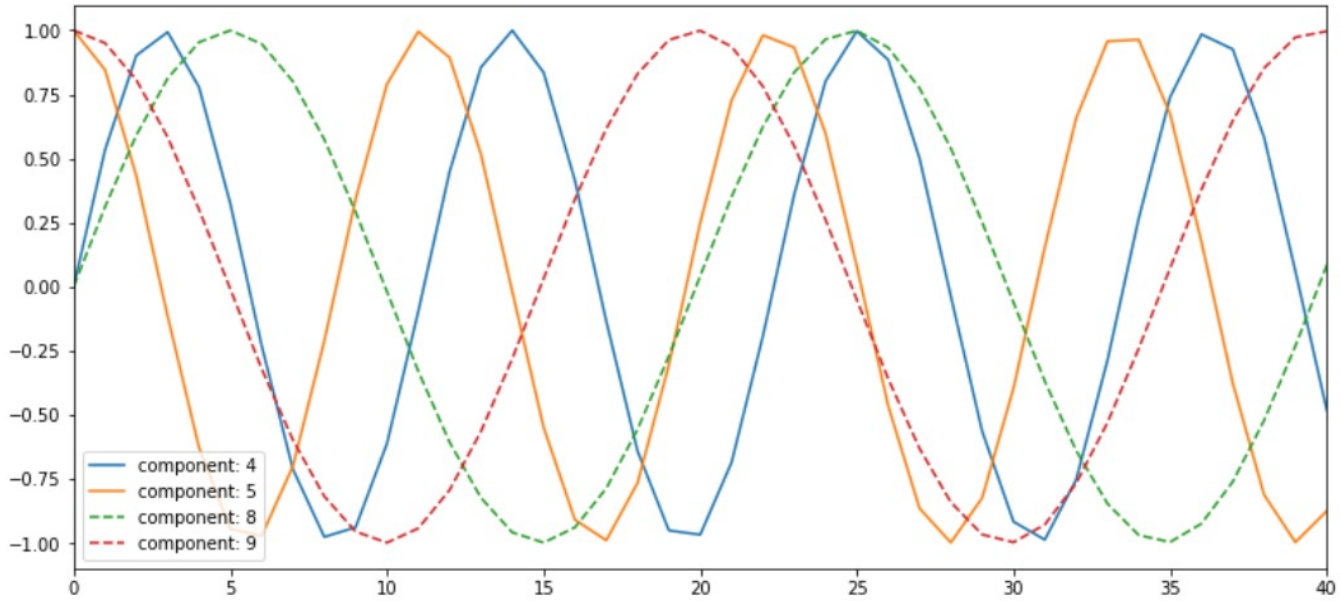
A bit more general approach

If we do not know the period, we can add many features of different periods (or frequencies ω_k) hoping some of those will help, e.g.: ω_k varying in [1, 1e-4]:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin\left(\omega_k \cdot t\right), & \text{if } i = 2k \\ \cos\left(\omega_k \cdot t\right), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}, k \in [0, d/2)$$



Features of such encoding

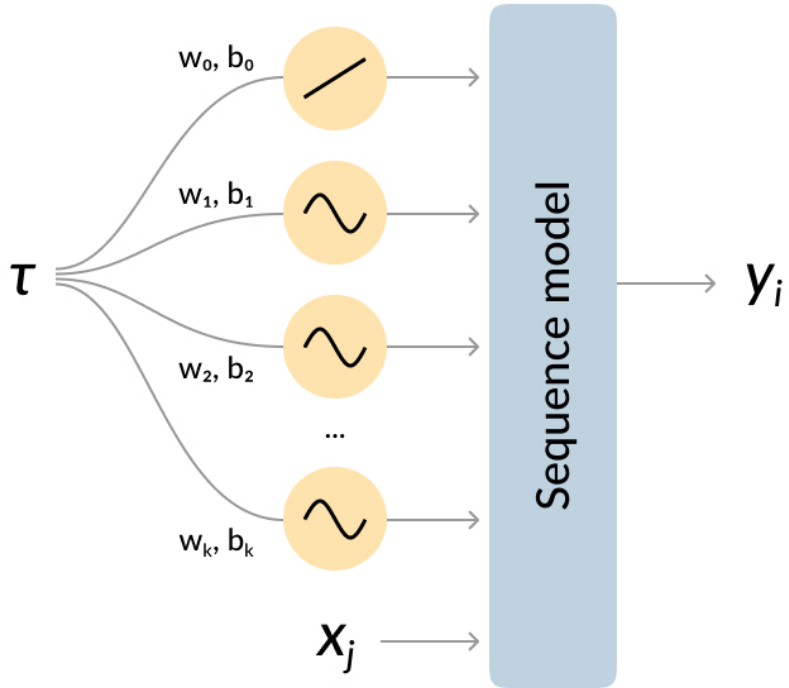


Individual components of positional encoding.

To merge those features with the original ones, you can **sum** or **concat** it with original feature vector.

Learnable positional encoding. Time to vector

In some cases, no a priori information regarding periodicity in your data is available. But you may suspect that it is there. Thus, one can use a so-called learnable positional encoding. Such an encoding could be a vector of dimensionality that coincides with the dimensionality of the basic features vector. You can add it to the original features during a forward pass of your neural network. During a backpropagation step of the model training, positional encoding vector components are learned in the same way as the layers' weights.



Time to vector encoding

For a given moment τ, we compute the encoding vector of dimensionality *k+1*:

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0. \\ \mathcal{F}\left(\omega_i \tau + \varphi_i\right), & \text{if } 1 \leq i \leq k. \end{cases}$$

where ***F*** is a periodic function (e.g., a sine or a cosine), all parameters $W=[w_i]$, $\Phi=[\varphi_i]$ are learned during the model training. The vector ***t2v*** is concatenated with the original feature vector.

Pros:

- time-rescaling invariance
- capturing periodic and a-periodic signals
- Simplicity

Cons:

- Overfitting. One can use stochastic techniques like dropout to make the encoding more generalizable and robust to mitigate the overfitting.

Quiz: How many parameters does Time2vec layer add?

Attention intuition

A woman is throwing a frisbee in a park.

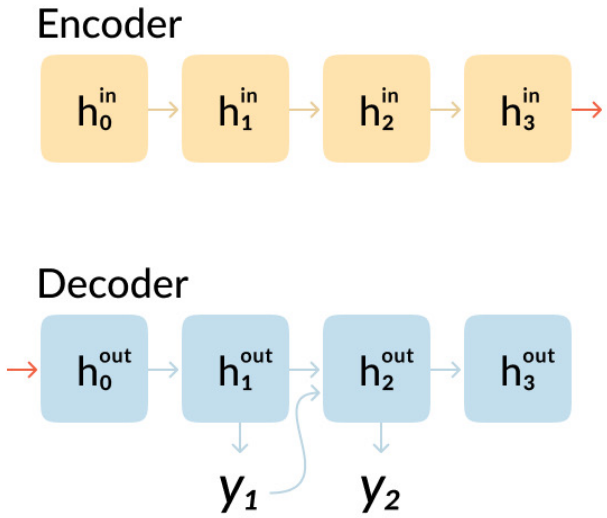
A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

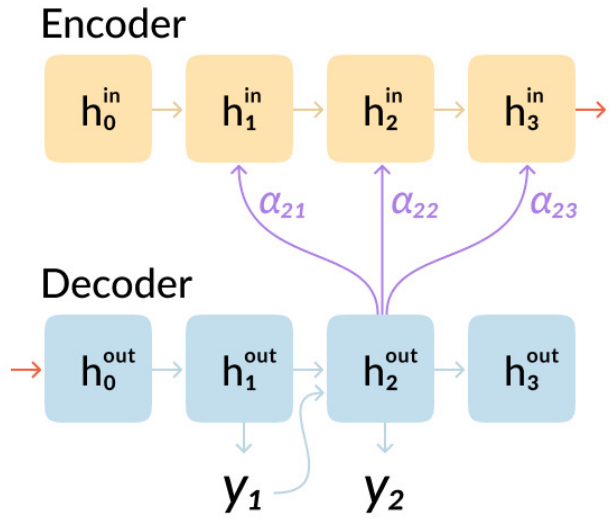
http://akosiorek.github.io/ml/2017/10/14/visual-attention.html

Attention mechanism

- Psychological phenomena of attention helps to distribute out limited resources amongst enormous information flow getting to the brain through our senses.



Regular sequence decoding



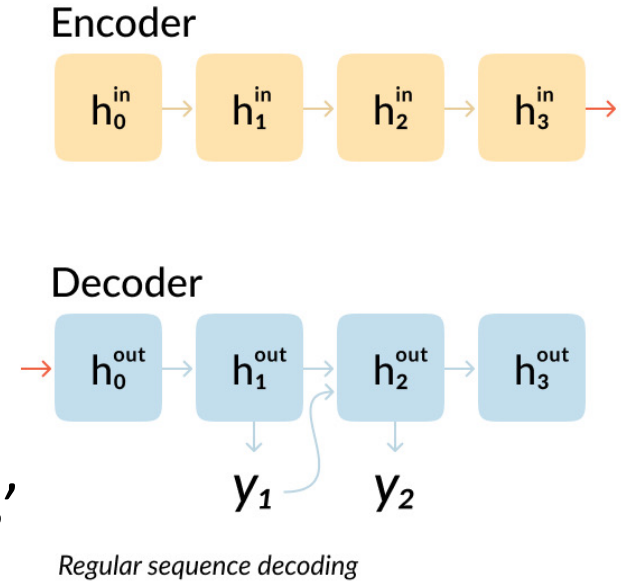
Attention-based sequence decoding

- Researchers in the field of neural language processing were looking for a universal way to deal with long sequences of loosely formalized text structures, e.g., sentences, that would help extract as much meaning as possible from it.

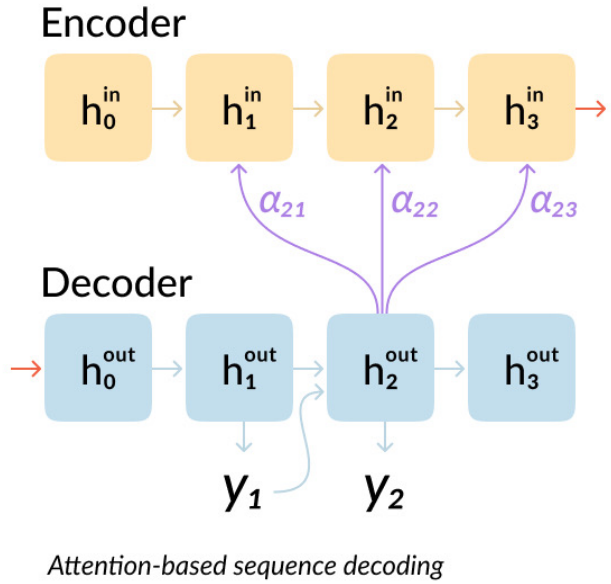
Attention mechanism formalization. Encoder

- Suppose, we have in input sequence X = {x_i}, that is encoded into vector sequence V = {$\mathbf{v}_i$}
- Given the input vectors as $\mathbf{v}_i \in \mathbf{R}^d$
- we have a task-specific pattern vector **u**, that 'directs' attention computed as an output of an 'alignment function' a(**u**, $\mathbf{v}_i$).

Intuitively, $\mathbf{v}_i$ that closely matches the pattern **u** receives a large weight and dominates the final encoding.



Regular sequence decoding



Attention-based sequence decoding

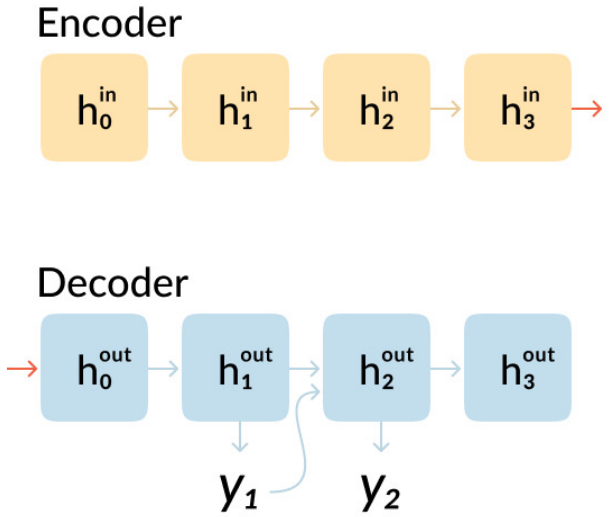
$$e_i = a(\boldsymbol{u}, \boldsymbol{v}_i) \text{ (attention scores)}$$

$$\alpha_i = \frac{e_i}{\sum_i e_i} \text{ (normalized attention scores)}$$

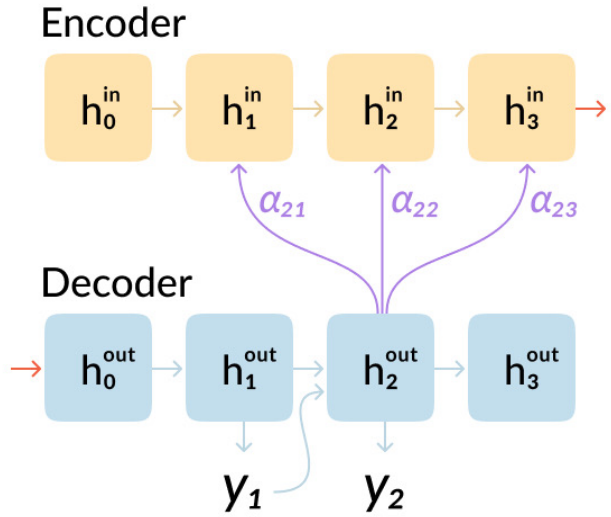
$$\boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v}_i \text{ (encoded attention vector)}$$

Attention mechanism formalization. Decoder

- Suppose, we have in input sequence X = {x_i}, that is encoded into vector sequence V = {$\mathbf{v_i}$}
- Given **c**, the *context encoding vector,* and y_{i-1} – a previously decoded symbol
- Decoder can be represented by function $y_i = f(y_{i-1}, \mathbf{v_{i-1}}, \mathbf{c})$



Regular sequence decoding



Attention-based sequence decoding

$$e_i = a(\boldsymbol{u}, \boldsymbol{v_i}) \text{ (attention scores)}$$
$$\alpha_i = \frac{e_i}{\sum_i e_i} \text{ (normalized attention scores)}$$
$$\boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v_i} \text{ (encoded attention vector)}$$

Alignment function examples

- Simplest alignment function:
  - $a(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$
- Adding a learnable tensor W:
  - $a(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T W\mathbf{v}$
- Concatenation:
  - $a(\mathbf{u}, \mathbf{v}) = \boldsymbol{\omega}_2^T \tanh(W_1[\mathbf{u}; \mathbf{v}])$, where W_1 and $\boldsymbol{\omega}_2$ are learnable matrix and a vector
- A simple neural network:
  - $a(u, v) = \sigma(\omega_2^T \tanh(W_1[\mathbf{u}; \mathbf{v}]+\mathbf{b}_1)+b_2)$, where $W_1, \boldsymbol{\omega}_2, b_1, b_2$ are learnable parameters, σ – non-linear function like sigmoid

Tough example

Sam walks into the kitchen. (1)

Sam picks up an apple. (2)

Sam walks into the bedroom. (3)

Sam drops the apple. (4)

Q: Where is the apple?

A: Bedroom

There is no direct relation between the question and the answer (3).

All linear or MLP-like models would not allow focusing on the word ‘bedroom’ given ‘apple’ as a pattern vector [Sukhbaatar, 2015].

Self-attention

- Conversely to approaches above, in self-attention, we can use sequence V to match the similarity between its elements; thus, the output can be interpreted as *self-attention*. [Cheng, 2016].
- Each token $\mathbf{v}_i$ maintains a distributed relation to all other tokens $\mathbf{v}_j$, and the complex pairwise relationship can be easily interpreted from the assigned scores.
- Examples (generic alignment, MLP):

$$e_{ij} = a\left(\boldsymbol{v}_i, \boldsymbol{v}_j\right) \qquad \alpha_{ij} = \mathrm{softmax}\left(\tanh\left(\boldsymbol{w}^T\left[\boldsymbol{v}_i; \boldsymbol{v}_j\right] + b\right)\right)$$
$$\alpha_{ij} = \mathrm{softmax}\left(e_{ij}\right)$$

Self-attention

The current word $\mathbf{v_i}$ is in red, and the intensity of the blue shade indicates the attention score α_{ij}
(from [Cheng, 2016])

$$\alpha_{ij} = \mathrm{softmax}\left(\tanh\left(\boldsymbol{w}^T\left[\boldsymbol{v}_i; \boldsymbol{v}_j\right] + b\right)\right)$$

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

The FBI is chasing a criminal on the run.

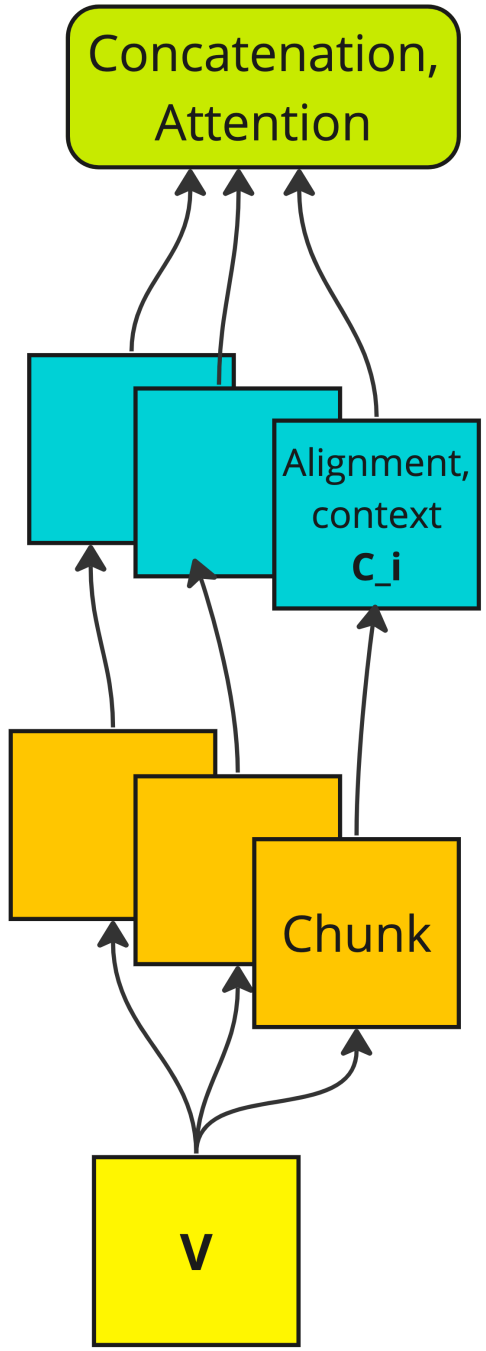
Multi-head self-attention

- Finally, one can extend the model mentioned above by splitting the long initial sequence V into *h* smaller chunks and computing the attention representation C_i in parallel .
- The resulting context encoding would be the result of the concatenation of individual attention vectors C_i and computing a linear mapping on top of it

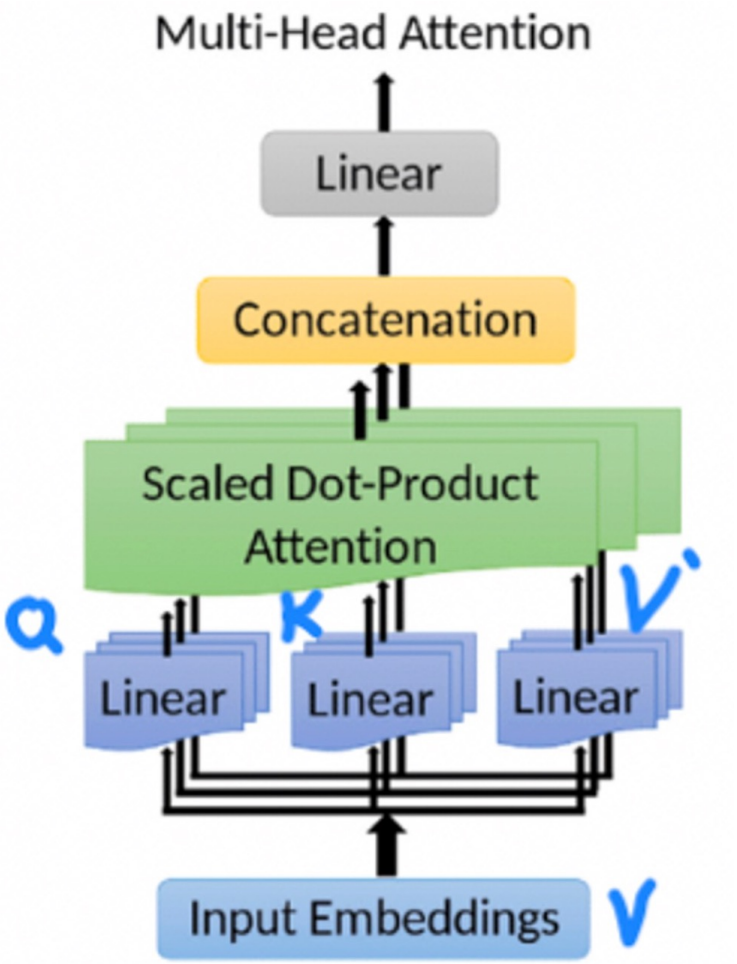
$$A_i = \text{softmax}\left[\left(V_i W_1^i\right)\left(V_i W_2^i\right)^T\right]$$
$$C_i = A_i^T\left(V_i W_3\right)$$

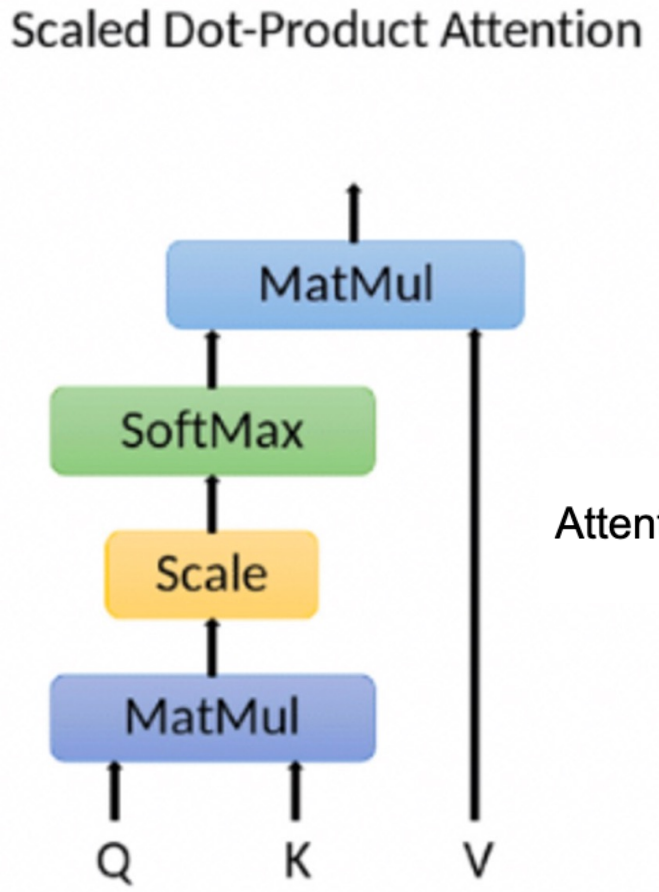
$$\text{MultiHeadAttention}(\mathbf{V}) = [C_1; \ldots; C_h]\mathbf{W}^o$$

- W^0, W^j_i are learnable matrices
- The model can learn larger models in parallel, which results in faster training and inference.



Multi-head attention





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

https://bit.ly/41iQ6g7

Memory-based attention intuition

- Goal: add memory-like mechanism that supports discovering latent dependencies in sequences.
- Attention as an answer to a query that is to be found in key-value storage. Represent information by a key-value pair ($\mathbf{k}_i$, $\mathbf{v}_i$) and let's learn to find the most relevant value to a given query **q**.
- Attention works as a soft memory addressing by **q** and giving answers as values from memory by attention scores α_i.
- This approach reduces to vanilla attention in a trivial scenario when all key-value pairs are equal.

1. $e_i = a(\boldsymbol{q}, \boldsymbol{k_i})$ (address memory)
2. $\alpha_i = \frac{\exp(e_i)}{\sum_i \exp(e_i)}$ (normalize)
3. $\boldsymbol{c} = \sum_i \alpha_i \boldsymbol{v_i}$ (read contents)

Training algorithm:

1. $initialize\ \boldsymbol{q} = question$
2. $e_i = a(\boldsymbol{q}, \phi_k(\boldsymbol{k_i}))$ (address the memory)
3. $\alpha_i = \frac{exp(e_i)}{\sum_i exp(e_i)}$ (normalize)
4. $\boldsymbol{c} = \sum_i \alpha_i \phi_v(\boldsymbol{v_i})$ (retrieve contents)
5. $\boldsymbol{q} = update\_query(\boldsymbol{q}, \boldsymbol{c})$ (update query)
6. $goto\ 2$ (multi-hop)

Memory-based attention illustration

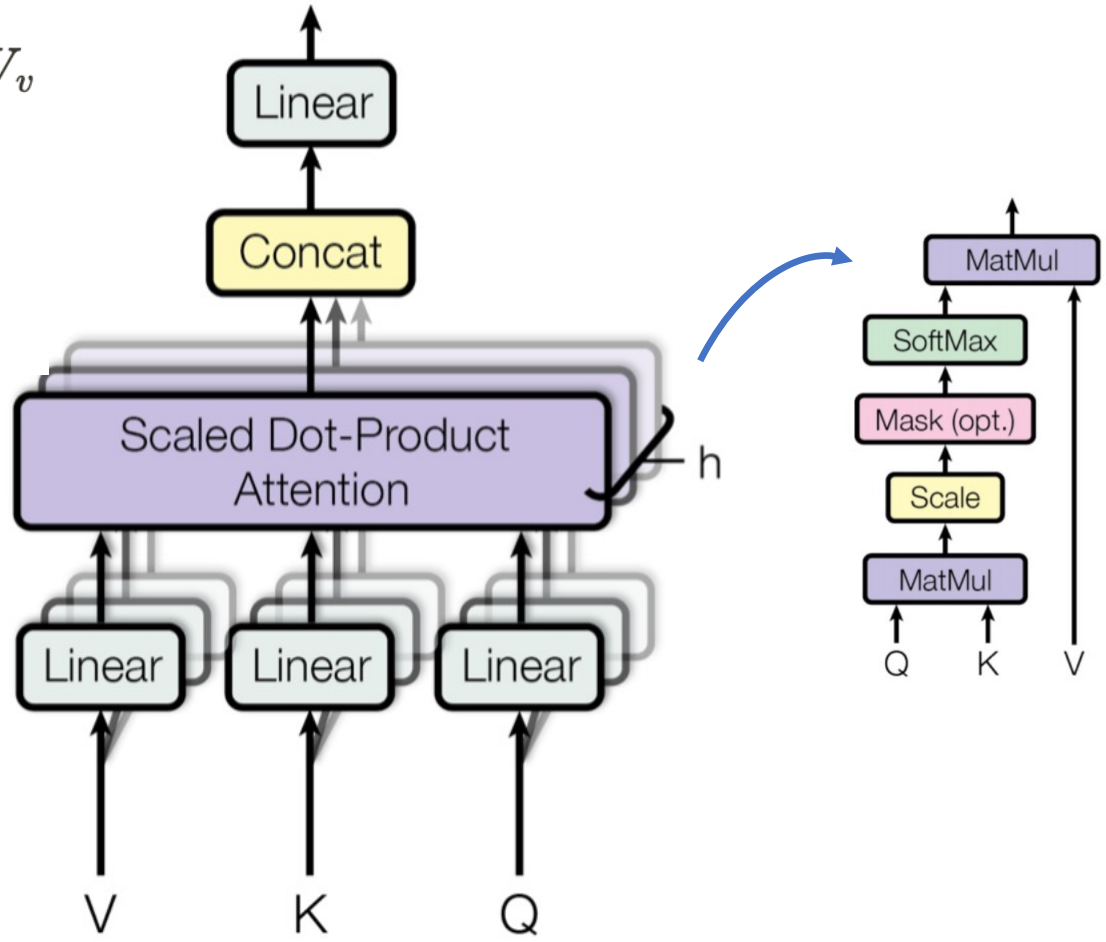
- In each iteration, the query is updated with new contents, and an updated query is used for getting the relevant answers.
- An illustration of this approach for solving question answering
- You can see how it starts with the original query and updates it at each iteration by appending it with the current value content.

Sam walks into the kitchen. (1)
Sam picks up an apple. (2)
Sam walks into the bedroom. (3)
Sam drops the apple. (4)
Q: Where is the apple?
$\boldsymbol{q}^{(0)}$= Where is the apple?
$\boldsymbol{c}^{(0)}$= Sam drops the apple
$\boldsymbol{q}^{(1)}$= [Where is the apple?; Sam drops the apple]
$\boldsymbol{c}^{(1)}$= Sam walks into the bedroom.
A: bedroom

Memory-based attention. Math.

$$K = VW_k, Q = VW_q, \hat{V} = VW_v$$
$$A = \text{softmax}\left[KQ^T\right]$$
$$C = A^T\hat{V}$$

$$\text{MultiHeadAttention}(\mathbf{V}) = [C_1; \ldots; C_h]\mathbf{W}^o$$



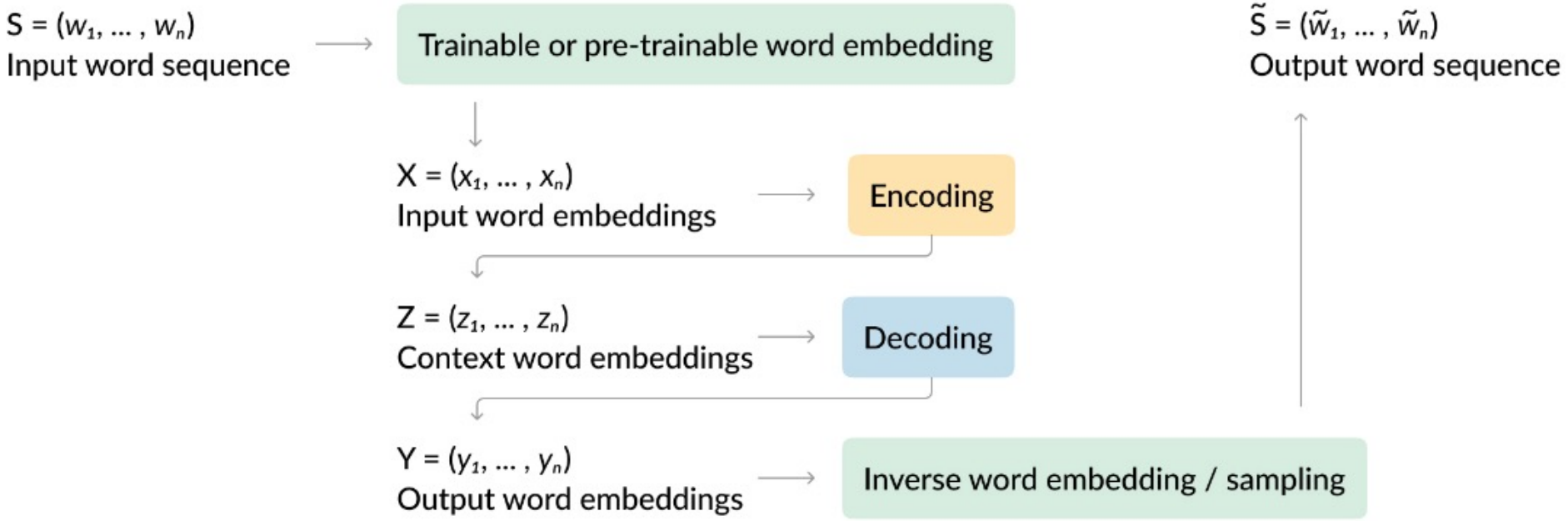
Illustrations of the multi-head self-attention mechanism. (Image source: [Vaswani, 2017])

Attention summary

- Attention mechanism allows for greater flexibility while dealing with sequences and higher-dimensional entities, i.e., the model can jointly attend to information from different representation subspaces at different positions.
- This mechanism becomes essential when dealing with natural language representations or time-series, for example, for the tasks of language modeling or machine translations.
- In the next section, we will see how the combination of the attention mechanisms, sequence encoding, and positional encoding contribute to a principal new model that sets a new state-of-the-art record for a variety of tasks.

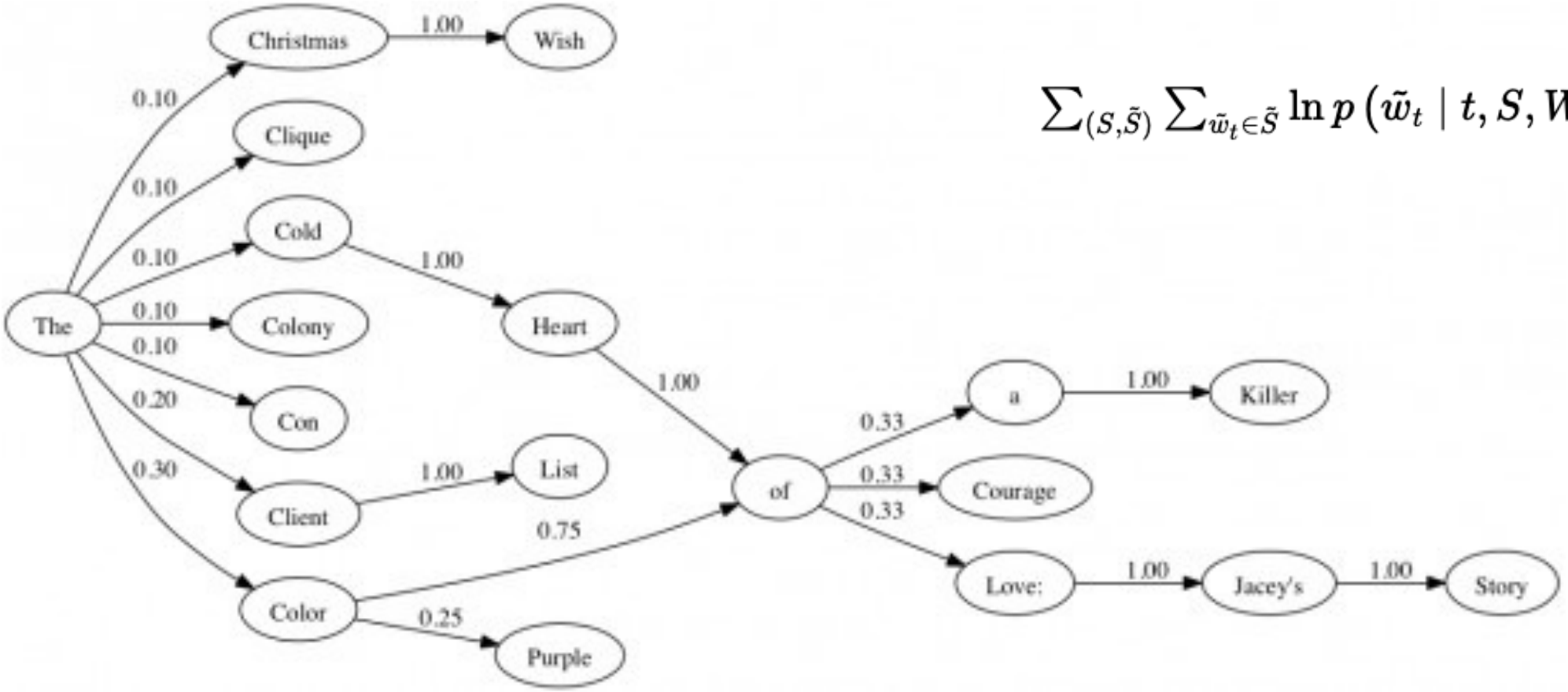
Transformer architecture

- Let's put it all together: embedding + attention + encoding/decoding = **transformer** [Vaswani, 2017].



Neural Machine Translation task

Probabilistic sequence modeling



$$\sum_{(S,\tilde{S})} \sum_{\tilde{w}_t \in \tilde{S}} \ln p\left(\tilde{w}_t \mid t, S, W\right) \rightarrow \max_W$$

Sample words given distribution $p(x_i|x_{1:i})$ at each step and generate sequence $x_i \sim p(x_i|x_{1:i})$

Neural Machine Translation metrics

$$\sum_{(S,\tilde{S})} \sum_{\tilde{w}_t \in \tilde{S}} \ln p\left(\tilde{w}_t \mid t, S, W\right) \rightarrow \max_W$$

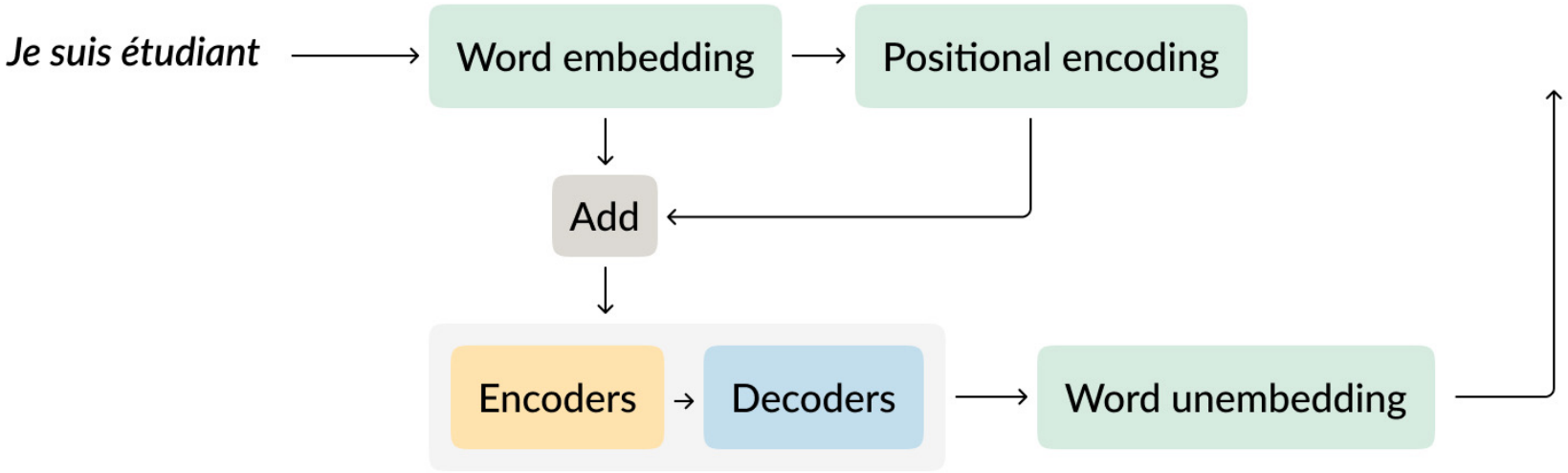
- Where p (w|t) is a posterior estimation of probability of word w at position/time t.
- Effectively this function represent a language model that is used for sampling output word sequences.
- Human-interpretable metric (It is called BiLingual Evaluation Understudy (BLEU):

$$BLEU = \min\left(1, \frac{\sum \text{len}(S)}{\sum \text{len}(S_0)}\right) \text{mean}_{(S_0,S)} \left(\prod_{n=1}^{4} \frac{\#n\text{-gram from } S, \text{ included in } S_0}{\#n\text{-gram from } S}\right)^{\frac{1}{4}}$$

- Non-differentiable, human-friendly:

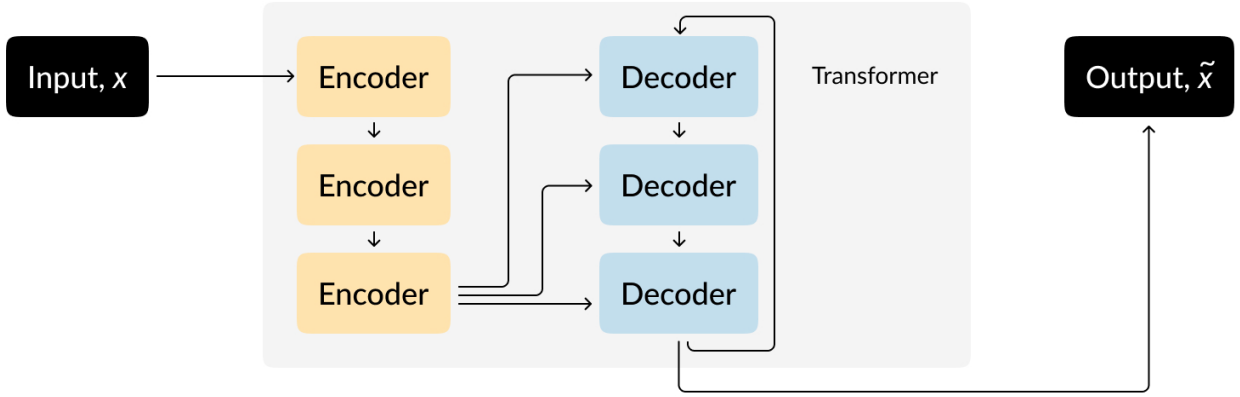
30 – 40	Understandable to good translations
40 – 50	High quality translations
50 – 60	Very high quality, adequate, and fluent translations
>60	Quality often better than human

Overall transformer setup



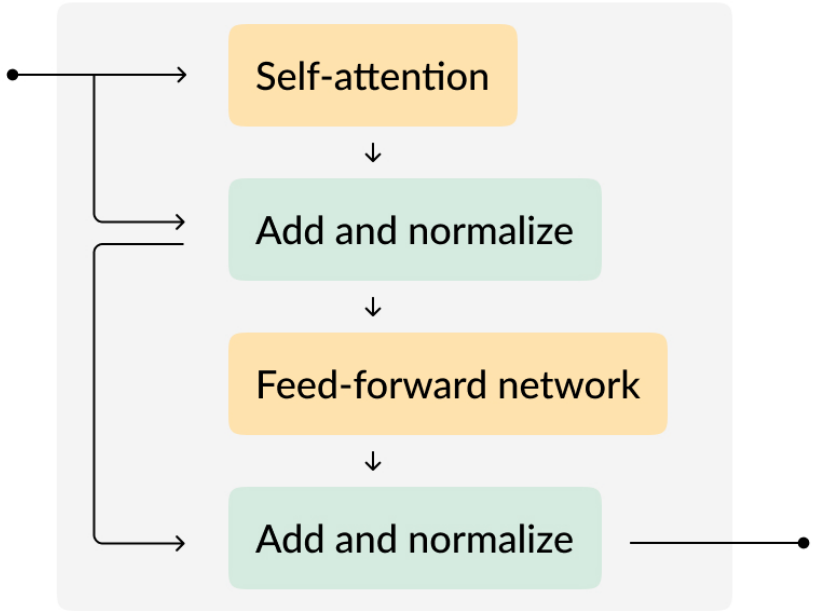
- The transformer assumes the input is given as a sequence of vectors that embed original words into a space of given dimensionality. The paper [Vaswani, 2017] has chosen this dimensionality to be equal to 512.
- Once we transform work tokens to the embedding, one computes positional encoding and sums it with the word embedding.
- Afterward, encoders inside the transformer find the best representation for the input sequence, and decoders transform it to the destination domain, translated back to words using inverse embedding.

Going deeper



- Each encoder consists of several stacked encoders, i.e., the output of each encoder is sent to the following one.
- The output of the final one is sent to all the decoders. The number of encoders and decoders is equal. The original paper suggested having six blocks for each. However, you can consider it as an architecture hyper-parameter.
- Each decoder gets output from the top encoder and previous decoder. The first decoder also gets a symbol generated on the previous iteration (which equals a unique token ‘BEGINNING OF SEQUENCE’ at the very beginning).

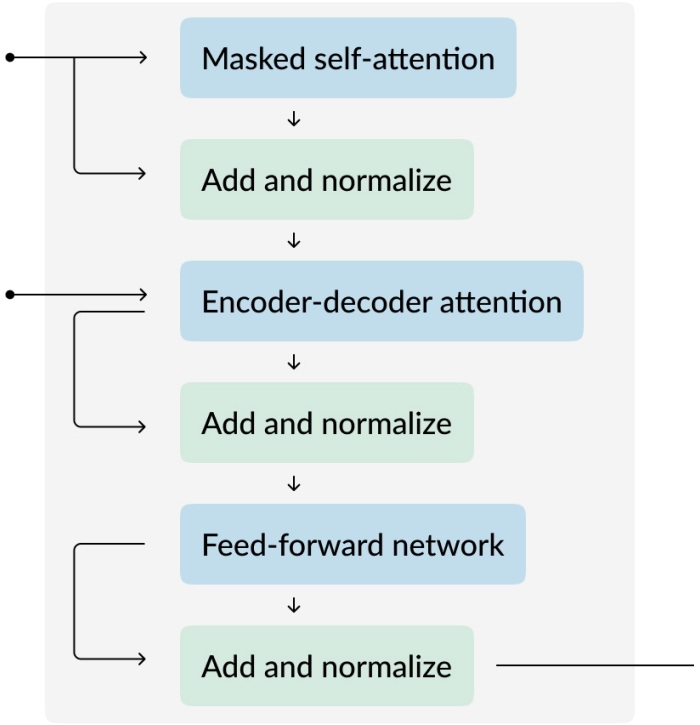
Encoders



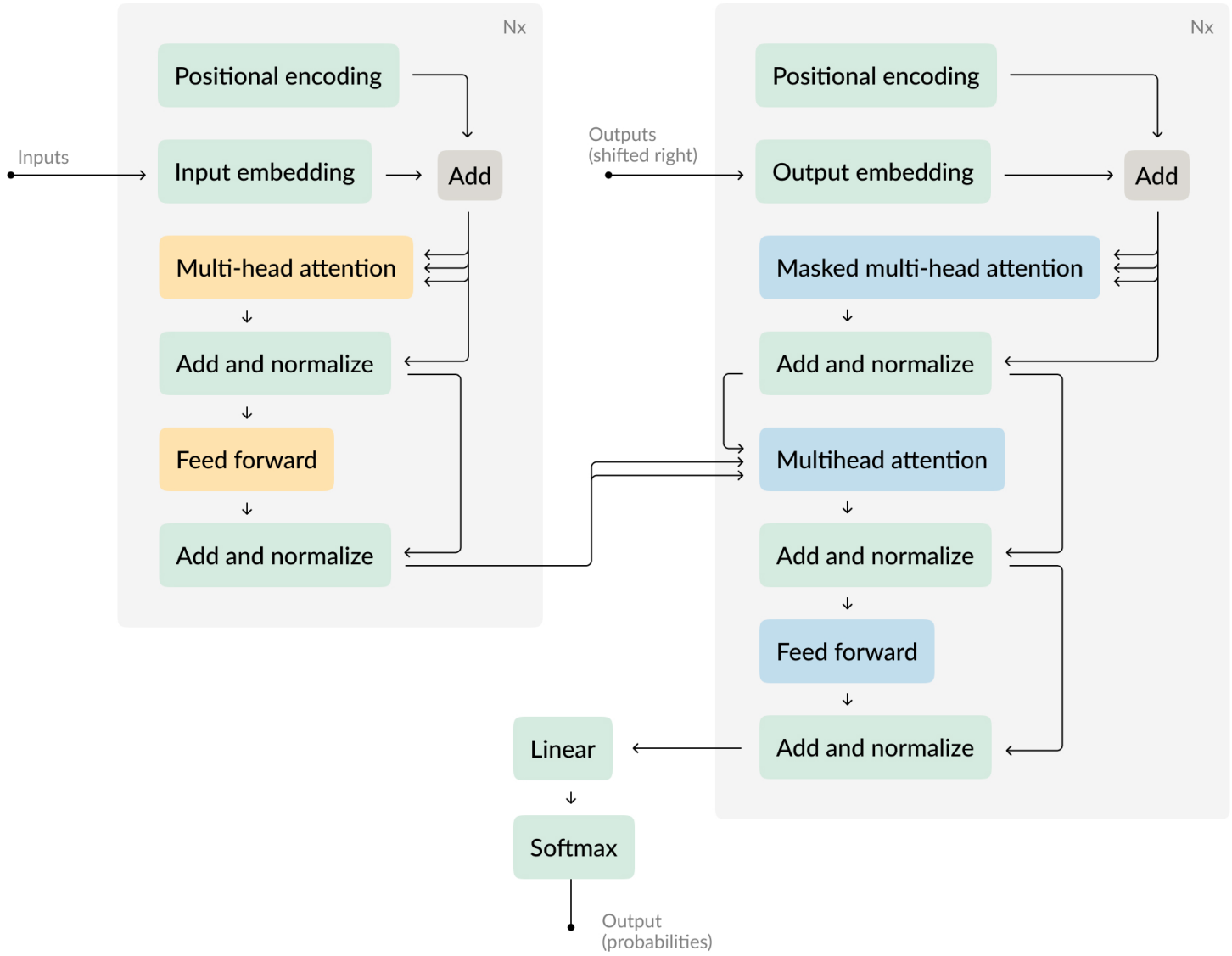
- Each encoder has two layers inside:
  - a multi-head self-attention layer and
  - a basic Feed-Forward Network (FFN, i.e., torch.nn.Linear + torch.nn.ReLU).
- The multi-head self-attention layer splits the input into 8 parts and computes K, V, and Q independently, trying to identify essential parts of the input.
- All individual head outputs are concatenated together and multiplied by a weight matrix W_c.
- The dimensionality of the vector didn't change after the self-attention layer, so its output is added to the input akin to residual connection and normalized through a normalization layer, which is conceptually like the Batch Normalization
- The output of the normalization layer is fed to the FFN. The trick with the residual connection and normalization is repeated with the FFN output.

Decoder

- The final encoder output is the output of the whole encoder block, which is sent to all the decoder blocks.
- Each decoder block is more complicated since it has two attention blocks. The first (right) attention layer takes the input from either the previous decoder block or takes a generated output vector y_{t-1} embedded and summed with positional encoding.
- This layer tweaks a regular self-attention a bit by so-called masking, which prevents it from peeking into the future (i.e., getting tokens of index higher than t). After the corresponding summation and normalization, we get vector h'_t
- The other attention block combines h'_t and encoder output Z, h'_t as a query and Z as key and value matrices for computing attention.
- The output of the second attention layer is added, normalized, and sent through an FFN like in the encoder block.



Overall architecture with details



Implementation details from [Vaswani 2017]

- There are several places where the attention mechanism is used inside the Transformer:
  - encoder self-attention
  - decoder self-attention, and
  - encoder-decoder attention
- In different contexts, it uses different matrices for computing the attention output. In all places, the authors have used multi-head attention, which allows for parallel training, increasing GPU utilization quite significantly.
- Also, they have used an attention function that is different a bit from what a regular dot-product-attention: they divide the dot product of Q and K by a multiplier sqrt(d_{key}), where d_{key} is the dimensionality of the *key* vector.
  - The reason for the normalization is that “for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients” [Vaswani, 2017].

Translation performance comparison

- To appreciate the effectiveness of the Transformer model, we quote here a table with a performance comparison of various models on English-to-German and English-to-French datasets. Besides the BLEU metric from [Vaswani, 2017], it shows how much training time was required to get the corresponding score.
- We can appreciate that the Transformer model achieves better BLEU scores than previous state-of-the-art models at just a fraction of the training cost.

Model	***BLEU***		***Training cost (FLOPs)***	
	En—De	***En—Fr***	***En—De***	***En—Fr***
ByteNet	23.75	—	—	—
Deep-Att + PosUnk	—	39.2	—	$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble	—	40.4	—	$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.3	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Random considerations

- The authors used fixed cos/sin encoding for positional encoding, which sums with the encoded vectors.
- Word embedding can be trained along with the whole Transformer
- Size of the vanilla transformer was 65M parameters
- People tried to merge attention models with CNNs and RNNs, but it turned out to be redundant
- Transformer model works faster and more accurate compared to RNNs
- Transformer model is easy to pre-train, and then one can re-use it for various tasks
- It is relatively easy to adapt Transformers to graphs and images
- It can be shown that multi-head self-attention is equivalent to a convolutional network [Cordonnier, 2020]

Transformer architecture summary

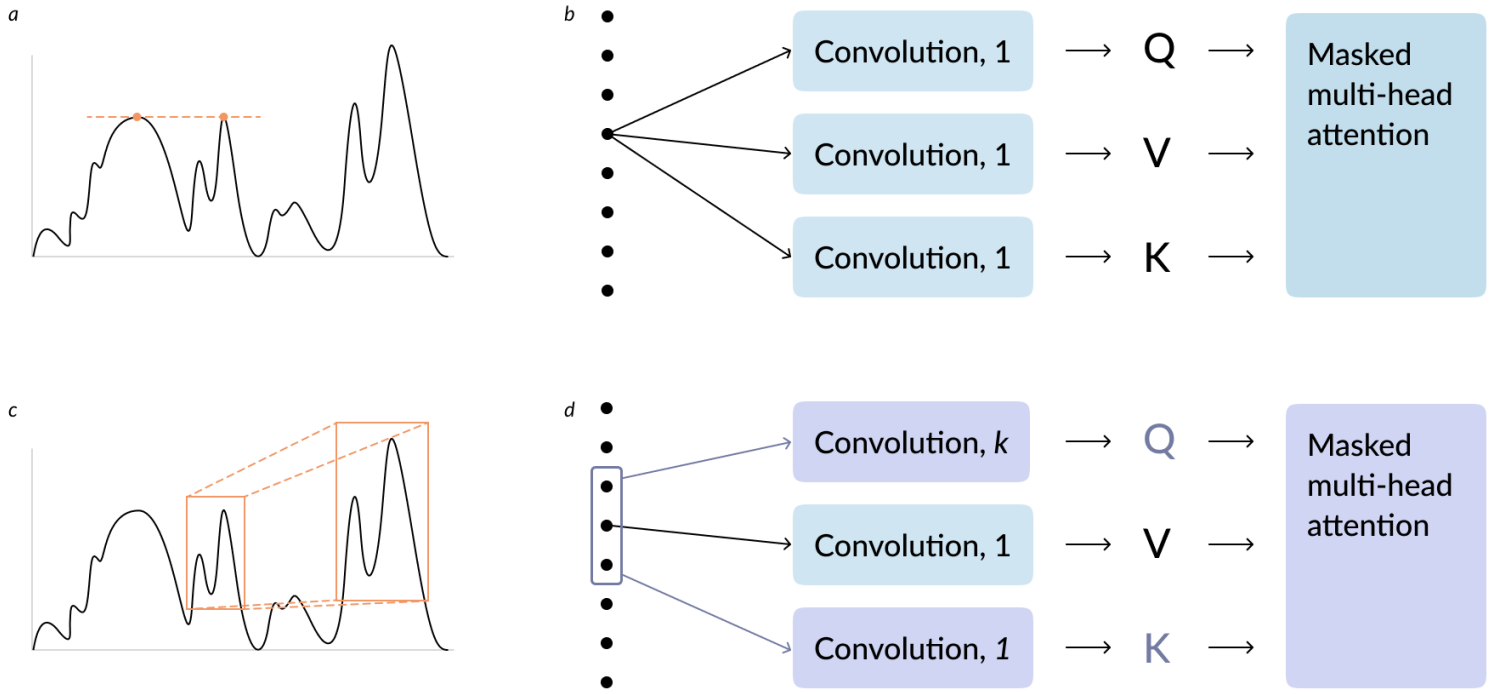
- Transformers are based on a variety of mechanisms we have covered previously.
- This model has been proposed numerous variations since the introduction of vanilla transformer: the original paper has more than 35k citations so far.
- As an excellent introduction to this zoo, we'd recommend starting with such papers as [Weng 2018, Han 2020, Kalyan 2021] that will help digest and organize the variations.
- **Quiz:** Why does transformer use masking inside attention?

Special architectures

Transformers for time-series

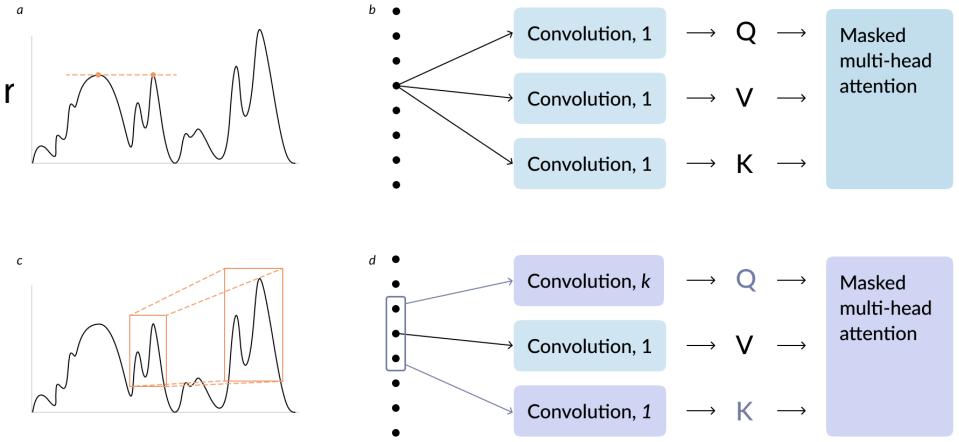
- One of the differences between time-series and regular text strings is that seasonality (or periodicity) may span much larger periods (i.e., yearly or even centennial cycles). Thus, attention mechanism from the vanilla Transformer that grows quadratically with the length of attention span needs an upgrade. The second issue is that vanilla Transformer might be too-sensitive to outliers that are usually easy to detect, considering local context properties like the shape of the time-series curve.
- Developing a reasonable method to allow a moderate vanilla transformer extension to a time-series domain did not take long. Indeed, the authors of [Li, 2019] have suggested several improvements to the architecture that fixed the issues mentioned earlier and enabled the Transformer to play a strong for time-series forecasting. The list of conceptual differences introduced by the paper is the following:
  - Causal self-attention
  - LogSparse Transformer

Convolutional and Causal self-attention



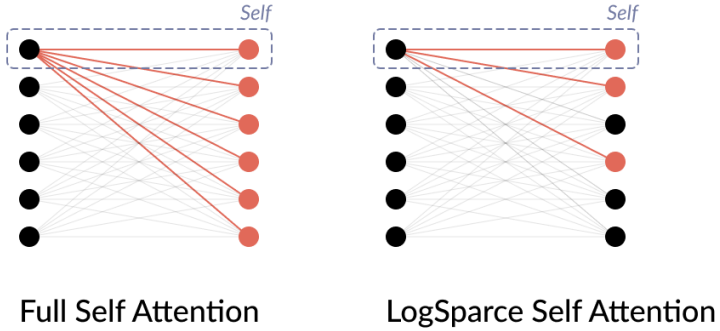
Convolutional and Causal self-attention

- Computation of attention between vectors u and v is originally a point-wise operation, i.e., it does not consider the surroundings of those vectors.
- For example, Fig. 1a compares two readings at different time moments with similar values (vertical coordinate). However, those are definitely are parts of completely different patterns that one can notice just by eye.
- One can use convolutions to make attention more accurate regarding neighbor readings or context, as shown in Figures 1c.
- Technically it will mean that instead of a dot product of raw vectors, one can compare a result of two convolutions with a given kernel size ("Conv, k" denotes a convolution of kernel size {1, k} with stride 1).
- It makes improvements, but it may introduce a look-ahead bias, which is the last thing you want dealing with time-series forecasting.
- Thus, one has to restrict the convolution to look into the past to consider only potential causes of the current vector value.
- Hence, the procedure justifies the name 'causal self-attention'. See figure 1d, where the convolution kernel is applied only to the neighbors from the past (to the left from the current position).



LogSparse transformer

- Once a transformer estimates the attention, it needs to have access to all sequence elements and compute the score between each pair of items. Thus, for a sequence of length L, it takes $O(L^2)$ memory for each layer, making forecasting for long-term patterns computationally prohibitive (left)
- So, the second improvement of the [Li, 2019] paper was targeted at reducing this memory footprint. The so-called LogSparse transformer method does not compute dot products between current and every previous position, but instead, it selects previous vectors separated by an increasingly growing number of steps.
- Thus, it only needs to calculate O(logL) dot products for each cell in each layer. In addition, to avoid losing essential dependencies, they stack up to O(logL) attention layers, thus giving access to every reading. Intuitively, one can compare the current vector with readings that are separated by an exponentially growing distance from the current one. (right)



Comparison

The following table from [Li, 2019] demonstrates the dominance of the Transformer-based approach for time-series forecasting. As a benchmark, the authors have selected two datasets: the electricity dataset consists of electricity consumption of 370 customers recorded every hour. Similarly, the traffic dataset contains occupancy rates for 963 freeway in San Francisco recorded every hour.

	ARIMA	ETS	TRMF	DeepAR	DeepState	Transformer
electricity_{1d}	0.154	0.101	0.084	0.075	0.083	**0.059**
electricity_{7d}	0.283	0.121	0.087	0.082	0.085	**0.070**
traffic_{1d}	0.223	0.236	0.186	0.161	0.167	**0.122**
traffic_{7d}	0.492	0.509	0.202	0.179	0.168	**0.139**

Mean absolute error of forecasting methods on electricity and traffic datasets. The 1st and the 3rd rows compute the error for rolling daily predictions of 7 days, while rows 2 and 4 show error for predictions directly 7 days ahead.

Higher-energy physics applications

- Jet studies: reconstruction and tagging
  - Point Cloud Transformers applied to Collider Physics
  - Permutationless Many-Jet Event Reconstruction with Symmetry Preserving Attention Networks
  - An Attention Based Neural Network for Jet Tagging
  - Zero-Permutation Jet-Parton Assignment using a Self-Attention Network
- Particle tagging and reconstruction
  - ABCNet: An attention-based method for particle tagging
  - SPANet: Generalized Permutationless Set Assignment for Particle Physics using Symmetry Preserving Attention
- Pile-up mitigation
  - Pile-Up Mitigation using Attention

Task 1. What is the total memory footprint for the full transformer using the LogSparse attention mechanism?

- $O(L^2)$
- $O(L \log(L)^2)$
- $O(log(L)^2)$

Anc

Going further

- Generative pre-training transformers [Radford, 2018]
- BERT [Devlin, 2019] work, which is essentially a pre-trained encoder part and can capture entirely meaningful patterns.
- [Zhou, 2021] introduces even more robust architecture called Informer and compares it with a bunch of other architectures.
- Akin to BERT, an approach called Time Series Transformer [Zerveas, 2021] also enables pre-training of a model and dealing with multivariate time-series simultaneously.

Conclusion

- Transformer architecture is a relatively new and powerful neural network architecture.
- This architecture allows faster training and reduced memory footprint compared to RNN and LSTM.
- It offers superior performance too.
- Due to its novelty, we introduced many new techniques, namely input representation encoding, positional encoding, and self-attention.
- Transformers by design are running nicely on text sequences from the beginning. However, multiple recent works have shown that it can handle time-series and images and videos with a noticeable improvement compared to the existing state-of-the-art.

References

- Hu, D. (2019, September). An introductory survey on attention mechanisms in NLP problems. In Proceedings of SAI Intelligent Systems Conference (pp. 432-448). Springer, Cham.
- Sukhbaatar, S., Weston, J., & Fergus, R. (2015). End-to-end memory networks. Advances in neural information processing systems, 28.
- Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. arXiv preprint arXiv:1601.06733.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
- Kalyan, K. S., Rajasekharan, A., & Sangeetha, S. (2021). Ammus: A survey of transformer-based pretrained models in natural language processing. arXiv preprint arXiv:2108.05542
- Cordonnier et al. On the relationship between self-attention and convolutional layers. 2020
- Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., ... & Tao, D. (2020). A survey on visual transformer. arXiv e-prints, arXiv-2012.
- Kalyan, K. S., Rajasekharan, A., & Sangeetha, S. (2021). Ammus: A survey of transformer-based pretrained models in natural language processing. arXiv preprint arXiv:2108.05542
- Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450
- Weng, L., Attention? Attention! 2018 https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y. X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. Advances in Neural Information Processing Systems, 32.
- Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021, February). Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of AAAI.
- Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., & Eickhoff, C. (2021, August). A transformer-based framework for multivariate time series representation learning. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (pp. 2114-2124).
- Devlin, J, Chang M., Lee K., Toutanova K., BERT: pre-training of deep bidirectional transformers for language understanding. 2019.
- Radford, A., et al. "Improving language understanding by generative pre-training." (2018).