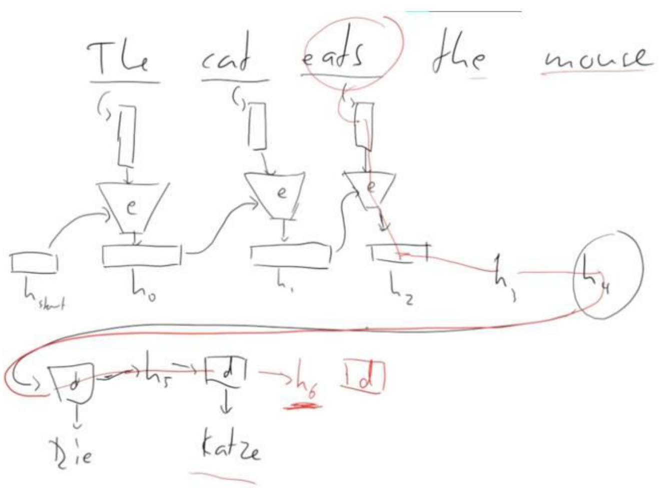
Devised as an alternative to NLP

- *All operations are able to be done in **parallel** (as opposed to RNNs / LSTMs), saving a lot of time*
- Traditionally, you would encode the sentence one word at a time, sequentially, and then you would decode it in target language
- These are called **seq-2-seq tasks**
- Usually solved with RNNs and LSTMs
- **Recurrent Neural Network -**
  - You go over the sentence, word at a time
  - Encode the word into a **word vector**
  - Use an **encoding NN** to turn the word vector into a **hidden state**
  - You then take the second token, represent it as a word vector
  - You then take the word vector, and the hidden state of the previous word, to make a new hidden state
  - You then put the final hidden state into a **decoder**, which gives you an output word vector and output the next hidden state
  - Starts with **h**$_{\textbf{start}}$ and ends with **h**$_{\textbf{end}}$



- **LSTM -** work in a slightly more complicated method
- Basically, in the decoder, the RNN just knows the last word token and the last hidden state
  - This means that the input word had to step through a ton of transformations through tons of tokens and then decodings, and then have to also remember the structure and meaning and stuff like that
    - **Long Range Dependencies**

**Transformers**

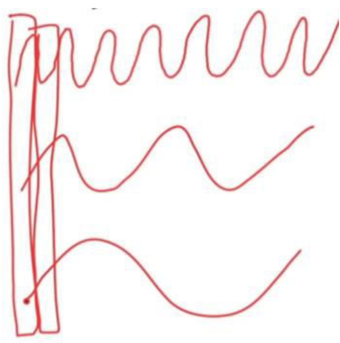
- Transformers, even in the very top layers, can already have pixels that attend to other pixels very far away, which CNNs can't do, until deeper into the network
  - **Attend to very far away things, right at the lower layers**
- Some attention heads can immediately attend very far away
- Transformers are more general than MLPs (which are more general than LSTMs and CNNs)
  - This is to do with inductive biases, i.e. an assumption that the algorithm/model architecture uses (i.e. local attention of CNNs, sequential inputs in LSTMs)
  - Transformers are more general than MLPs because ____
  - This means that they can spot some patterns that perform similarly to less general networks, but are slightly better, because they can learn to modify those patterns in ways that we wouldn't be able to integrate as an inductive prior
    - To use a more general model, you need more data and more computation so that you can train models without the help of those inductive biases

**Attention**

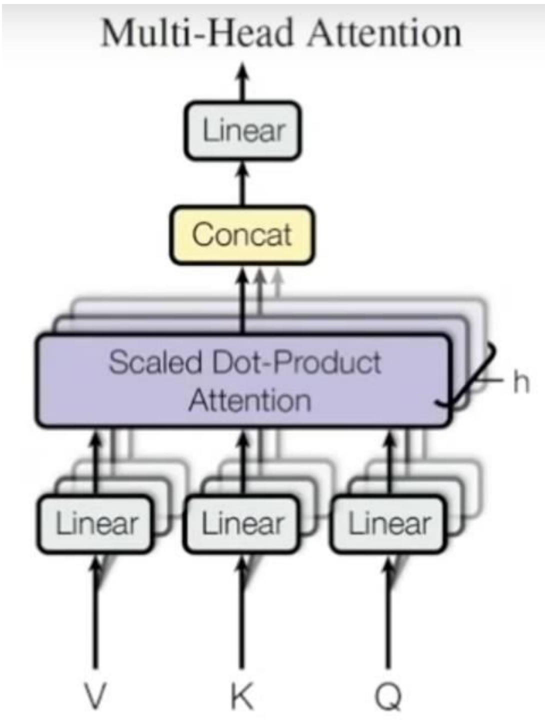
- Attention allows the **decoder** in the NN to go back and look at specific parts of the input
  - In popular attention mechanisms, the decoder can decide to attend to particular parts of the input sentence that it thinks are the most relevant
  - We want the decoder to learn which parts of the input are important to decode the current word
  - All of the hidden states are connected to all of the decoders, so that the decoders can look back at all of the hidden states and decide which ones they want to attend to
  - The decoder outputs a bunch of keys which index the hidden states in a softmax architecture

**The paper**

- Suggests ditching RNNs - they say you don't need recurrence, you just pay attention to everything at the same time
- It has two parts - an **encoder** and a **decoder**
- **For each step** (for each word being decoded)
  - The **entire** source sentence goes into the **input embedding**
  - **All outputs up until the current** decoder goes into the **output embedding**
  - The decoder then gives us probabilities for all words and we pick the one with the highest one
  - We have to backpropagate for every single output
    - Every step in the sentence is a training sample
      - *In RNNs, you only backpropagate once at the end of the whole sentence*
        - Very big change in the NLP paradigm
- **Positional Encoding**
  - Initially, because positional information is lost, we initially put all embeddings through **positional encoding**
    - **Positional Encoding -** You try to encode the positional information to the embedding so that it can be referred to later
      - Usually done with some sort of trigonometric function, positions encoded by a couple of **sine waves with differing frequencies**
      - Continuous way of encoding position and can be compared
        - You can go through each one upwards for different detail
  - **Positional encoding** is **added** to the **input embedding**

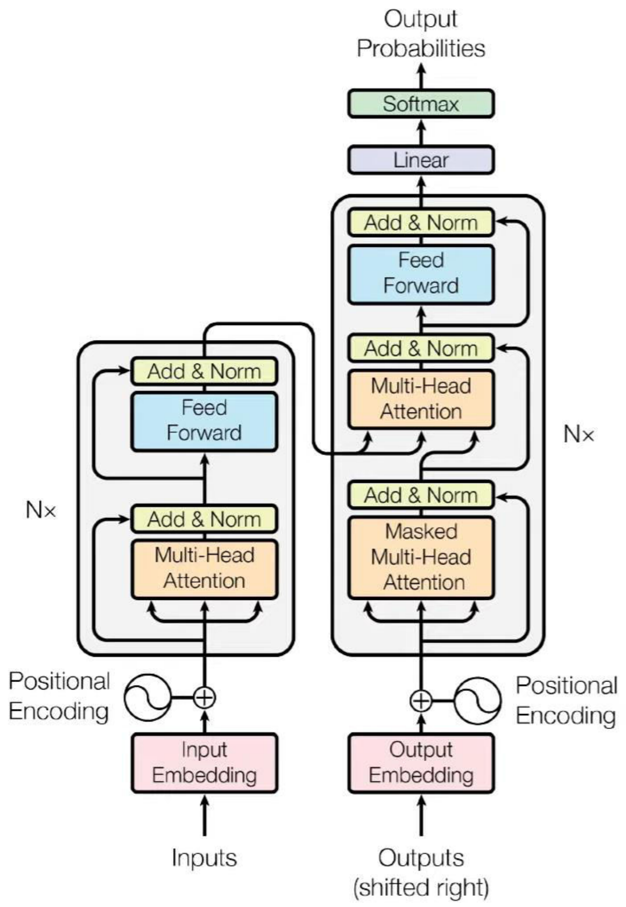


- **Attention**
  - There are **3 kinds of attention**
    1. Attention over the input sentence
       a. **The input sentence needs to be encoded into a representation all at once**
          i. This attention learns which part of the input to focus on
    2. **Attention over the outputs up to the current point**
       a. Outputs encoded all at once
          i. Attention will learn which part of the output to focus on
    3. W
       a. **Most interesting part of the attention mechanism**
          i. Combines the source sentence with target sentence up to this point
          ii. Has three connections going into it
          iii. **The Keys, Values and Queries**
- **Keys, Values and Queries**
  - **Keys and Values** are output by the inputs
  - **Queries** are produced by the outputs
  - **Attention**
    1. Computes the **dot product of QK**
    2. **Softmaxes** $QK^T$
    3. **Dot products** it with **V**
  - If the query and the key **align**, the dot products will be high
- **Keys** are used to index the **Values**, which are things of **interest**
- Queries are compared to keys, to see **which key fits the query most**
- **Value -** things that could be of interest
- **Attention -** selects the **V** that is referred to by the **K** that most fits **Q**
  - **Attention** is then put through the linear part of the NN
- **Input Encoder - discovers** interesting things about the source sentence
  - It builds the **key-value pairs** that **index** points of interest
- **Output Decoder -** builds up **queries** to query the interesting things with



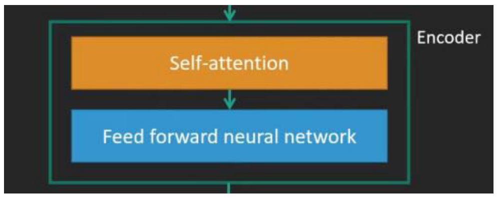
**Reason for Performance**

- Attention **reduces** the **path lengths** in the network
- It **reduces the number of transformations** that information has to flow through
- Transformations (computation steps) can make you **lose information**



**Implementation**

- The **attention head** takes as input keys, values and queries
- The arrows are just **skipped connections**
- The output of the **attention head** is **added** to the **input**
- **Positional Embedding -** used because transformers are **permutationally invariant**, they output the same thing regardless of word order
- **Masked multi-head attention -**
    - **Masking -**
- The embedding gets split into different parts, called **heads**
    - At the end of the **multi-head attention**, the outputs are concatenated into one (length = |embedding|)
- We **stack numerous encoders and decoders** on top of one another
    - This means we process the sequence numerous times
    - **Stacking idea -** capture raw properties first, before discovering more complicated properties

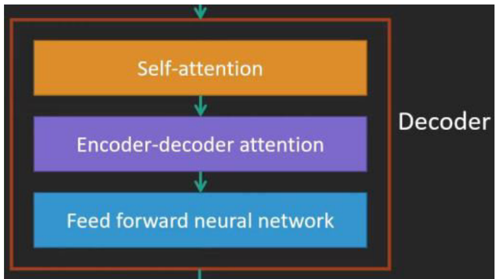


**Encoders**

- Made out of **multi-head** **self-attention** and *feed-forward NNs*

**Decoders**

- Similar to encoders, you get **self-attention, encoder-decoder attention**, and *feed-forward NNs*



**Attention -** attention is when we have an input sequence and an output sequence, and we compute which parts of the input are relevant to which parts of the output

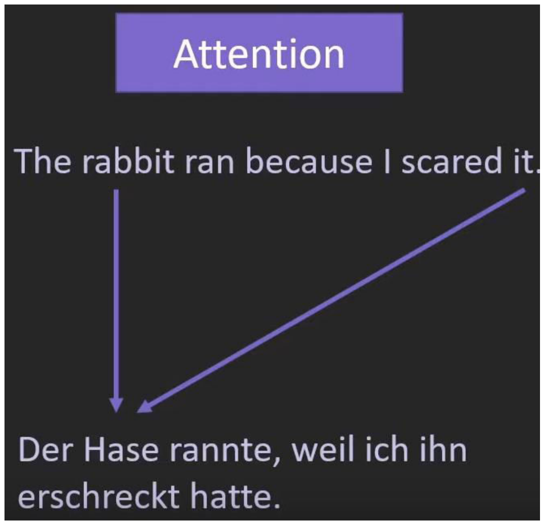
- We compute this in **encoder-decoder attention**

**Self-attention -** compute importance between parts of the input with itself

- Tells us which word in the input is relevant to which other input word
- Outputs new vectors in place of the input vectors (same number/size)

**Residual connection -** the input of each layer is added to its output

-


Attention
The rabbit ran because I scared it.
Der Hase rannte, weil ich ihn erschreckt hatte.