OTH
Amberg-Weiden

Advanced Topics in Machine Learning

Winter Semester 2024/2025

Prof. Dr.-Ing. Christian Bergler | OTH Amberg-Weiden

Advanced Topics in Machine Learning

Overview

OTH Amberg-Weiden

Topics From Last Time: Introduction & Deep Learning Recap

- Terminologies
- ML vs. DL
- Supervised, Unsupervised, Semi-Supervised Learning
- Logistic & Softmax Regression
- Forward & Backward Propagation
- Network Initialization, Regularization and Optimization
- Convolutional Neural Networks & Autoencoders

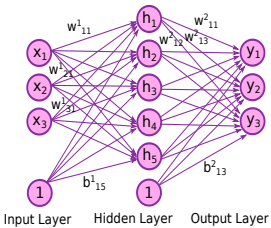
Topics of Today: Attention and Transformer Models

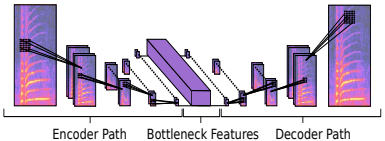
- Recurrent Neural Networks (Forward/Backward Propagation)
- Long Short-Term Memory (LSTM) & Gated Recurrent Unit (GRU) Models
- Attention Mechanism
- Transformer Model

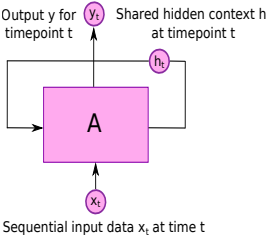
Deep Learning Pipeline

Recap: Basic Architectural Concepts

Fully-Connected Architecture | Convolutional Architecture | Recurrent Architecture







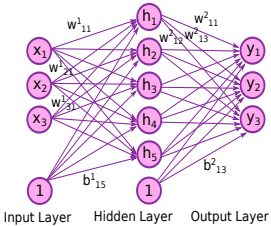
- Different Topologies: Feedforward neural network (e.g. Fully-Connected model, CNN) versus Recurrent neural network (e.g. RNN, LSTM, GRU)

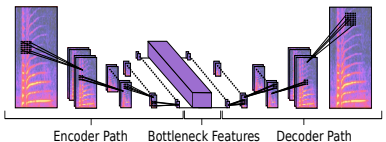
Source: Image (Autoencoder), Bergler et al., "Deep Learning for Orca Call Type Identification – A Fully Unsupervised Approach"

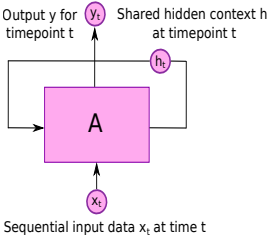
Deep Learning Pipeline

Recap: Basic Architectural Concepts

Fully-Connected Architecture | Convolutional Architecture | Recurrent Architecture







- Different Topologies: Feedforward neural network (e.g. Fully-Connected model, CNN) versus Recurrent neural network (e.g. RNN, LSTM, GRU)
- Combinations of different (basic) architectural paradigms possible, to form more complex models (e.g. Convolutional (Recurrent) Neural Networks (CRNN), Autoencoder (AE), You-Only-Look-Once (YOLO), Generative Adversarial Architectures (GAN), Transformer, etc.)

Source: Image (Autoencoder), Bergler et al., "Deep Learning for Orca Call Type Identification – A Fully Unsupervised Approach"

Sequence Modeling With Recurrent Neural Networks (RNN)

OTH Amberg-Weiden

Motivation and Architecture...

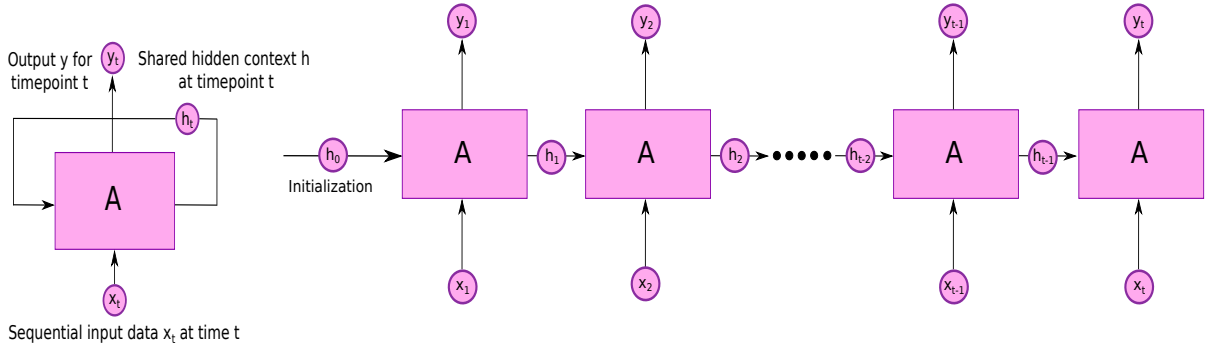
- Feedforward Neural Networks have no feedback (Input → Hidden (processing) → Output)

  → "No memory!"

- Problem: Modeling of sequential data (e.g. speech recognition, stock prices, weather forecasts, driver assistance systems) → Temporal context is of enormous importance!
- Recurrent architectures model neuronal connections across multiple layers (similar to the human brain) → Possibility of temporal coding
- Different architectures for sequence learning (recurrent models in the form of one-to-many, many-to-one, many-to-many, stacked, bidirectional or transformer architectures)

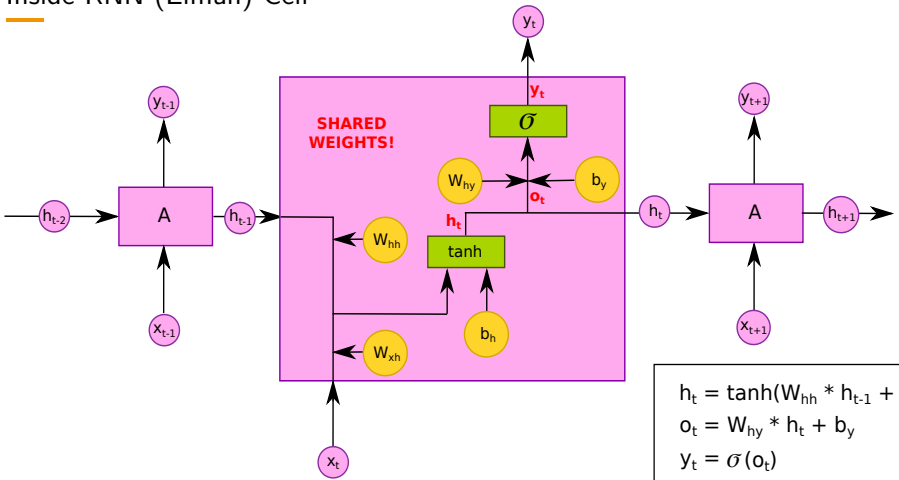
Sequence Modeling With Recurrent Neural Networks (RNN)

Motivation and Architecture...RNN Architecture (Elman Cell)



Sequence Modeling With Recurrent Neural Networks (RNN)

Inside RNN (Elman) Cell



$$h_t = \tanh(W_{hh} * h_{t-1} + W_{xh} * x_t + b_h)$$
$$o_t = W_{hy} * h_t + b_y$$
$$y_t = \sigma(o_t)$$

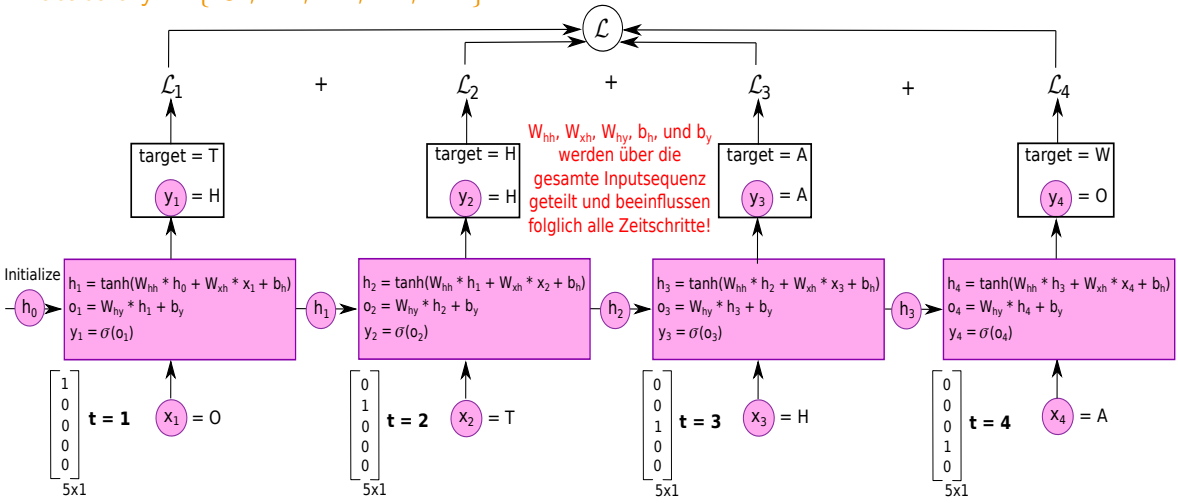
Source: Elman, Jeffrey L. "Finding structure in time" Cognitive science 14.2 (1990): 179-211

Sequence Modeling With Recurrent Neural Networks (RNN)

OTH Amberg-Weiden

Network Training – Forward Propagation

Vocabulary = {'O', 'T', 'H', 'A', 'W'}



Backpropagation Through Time (BPTT)

Brace Yourselves – Math is coming!

Source: Images taken from Link

Backpropagation Through Time (BPTT)

What it is about?

- Backpropagation over a time series is called "Backpropagation Through Time (BPTT)"
  $\rightarrow$ Backpropagation algorithm in connection with RNNs!
- Optimization of W_{xh}, W_{hh}, W_{hy}, b_h, b_y (Note: divided over all time steps)
- Training RNN (classical):
  - Forward propagation over the entire time sequence
  - Calculation of the total costs (loss)
  - Backward Propagation Through Time over the entire time sequence to obtain gradients
  - Gradient of the loss function with respect to W_{xh}, W_{hh}, W_{hy}, b_h, b_y

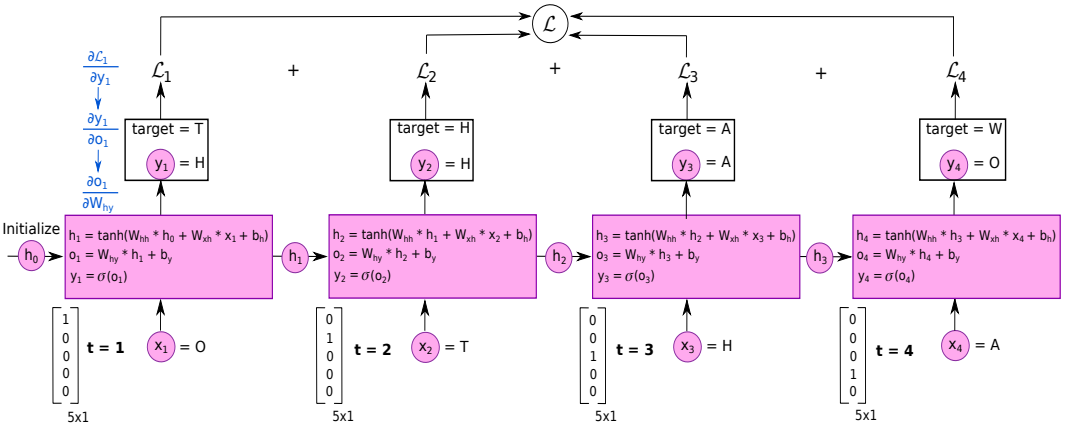
$$\nabla\theta = [\nabla W_{xh}, \nabla W_{hh}, \nabla W_{hy}, \nabla b_h, \nabla b_y]$$

  - Update the parameters with a learning rate η

$$\vec{\theta} = \vec{\theta} - \eta\nabla\vec{\theta}$$

Backpropagation Through Time (BPTT)

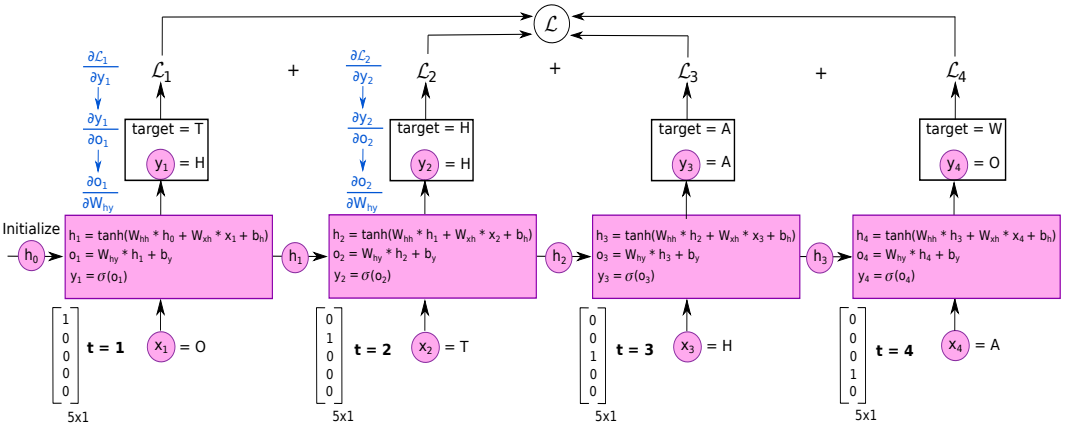
Network Training – Optimization of W_{hy} and b_y



$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_1} \frac{\partial y_1}{\partial o_1} \frac{\partial o_1}{\partial W_{hy}} +$$

Backpropagation Through Time (BPTT)

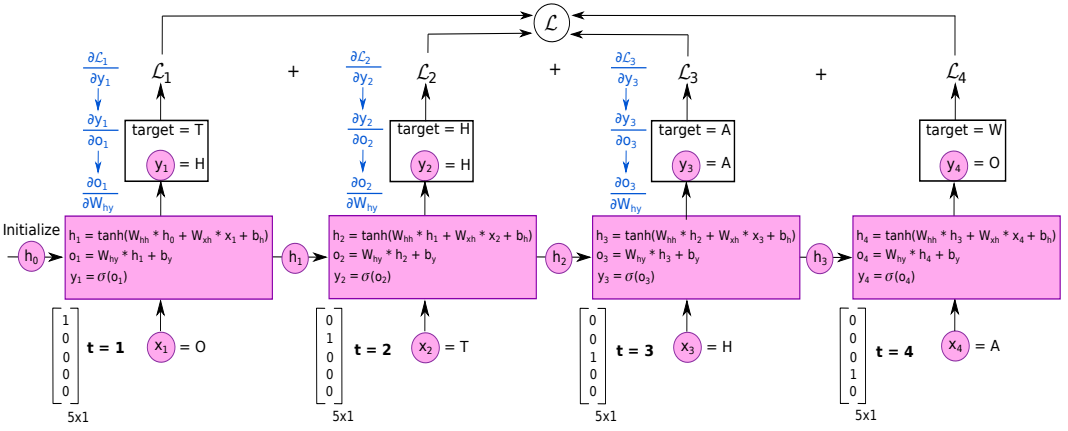
Network Training – Optimization of W_{hy} and b_y



$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_1} \frac{\partial y_1}{\partial o_1} \frac{\partial o_1}{\partial W_{hy}} + \frac{\partial \mathcal{L}_2}{\partial y_2} \frac{\partial y_2}{\partial o_2} \frac{\partial o_2}{\partial W_{hy}}$$

Backpropagation Through Time (BPTT)

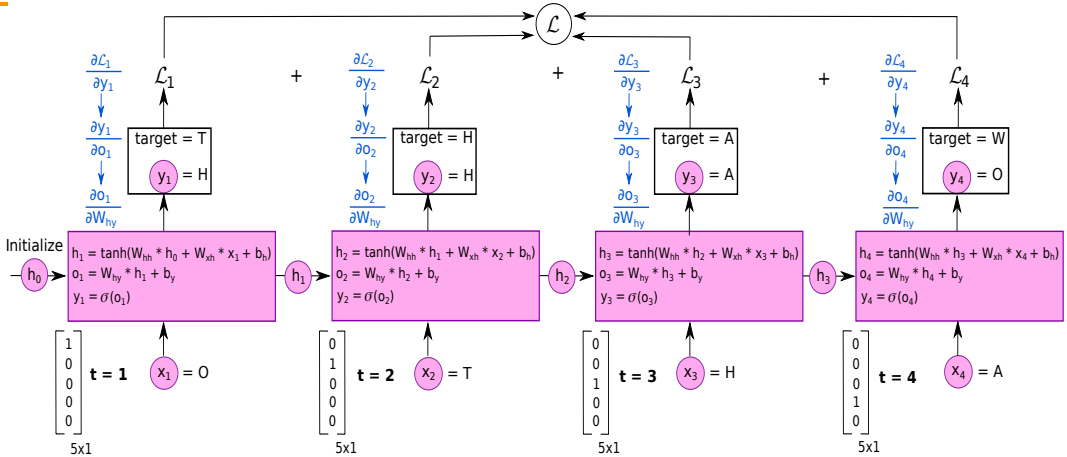
Network Training – Optimization of W_{hy} and b_y



$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial W_{hy}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial W_{hy}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial W_{hy}}$$

Backpropagation Through Time (BPTT)

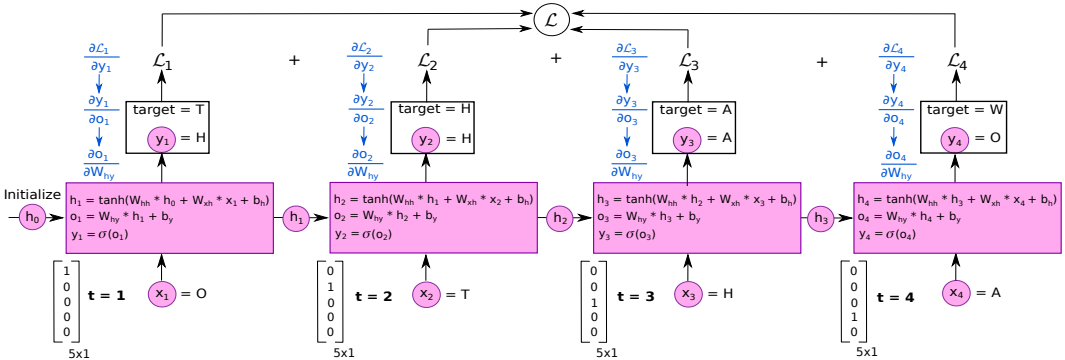
Network Training – Optimization of W_{hy} and b_y



$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial W_{hy}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial W_{hy}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial W_{hy}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial W_{hy}}$$

Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hy} and b_y

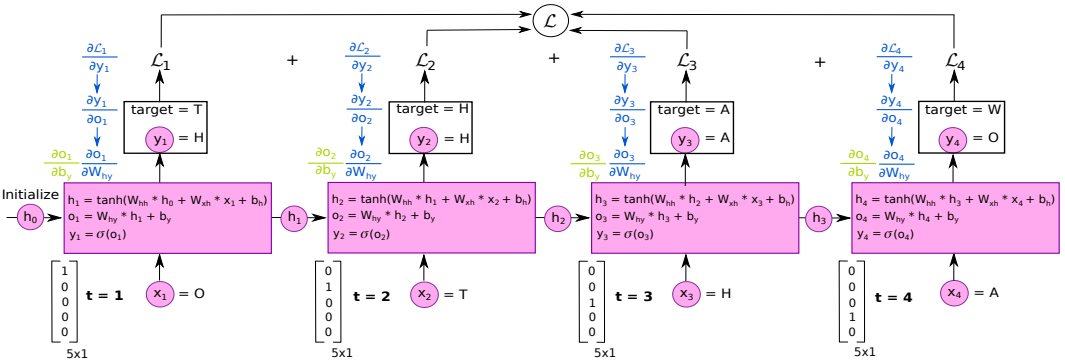


$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial W_{hy}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial W_{hy}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial W_{hy}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial W_{hy}}$$

$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \sum_{t}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial o_t}\frac{\partial o_t}{\partial W_{hy}}$$

Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hy} and b_y

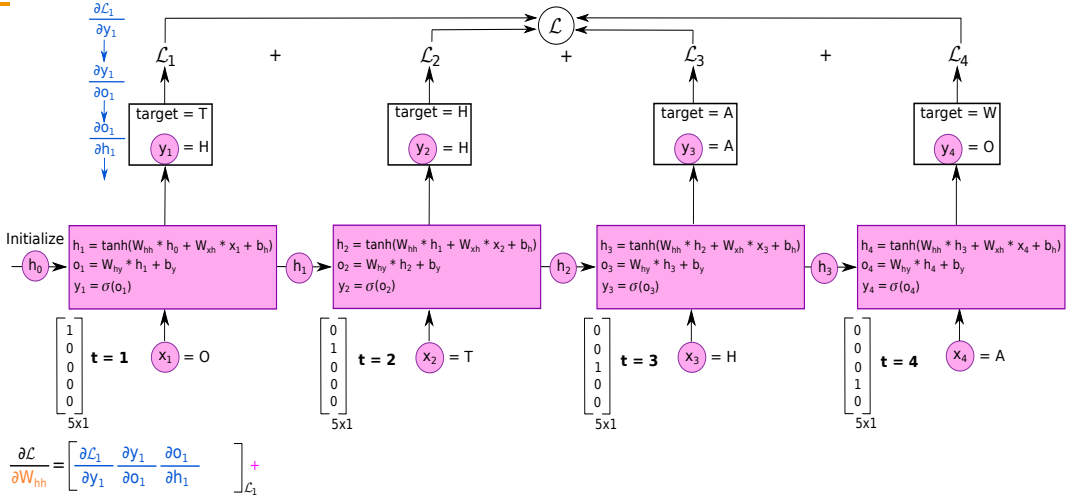


$$\frac{\partial \mathcal{L}}{\partial W_{hy}} = \frac{\partial \mathcal{L}_1}{\partial y_t}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial W_{hy}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial W_{hy}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial W_{hy}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial W_{hy}}$$

$$\frac{\partial \mathcal{L}}{\partial W_{hy}|\partial b_y} = \sum_{t}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial o_t}\frac{\partial o_t}{\partial W_{hy}|\partial b_y}$$

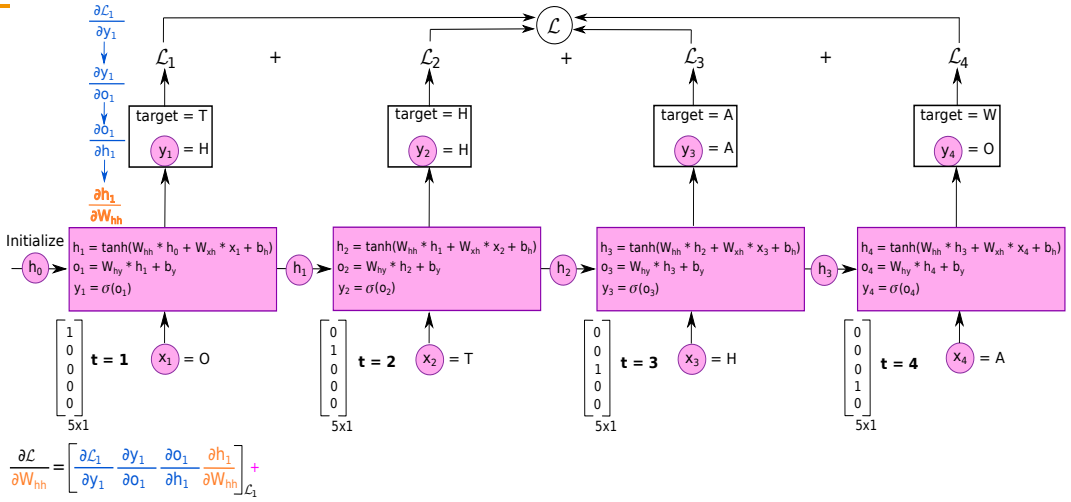
Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hh}, W_{xh} and b_h



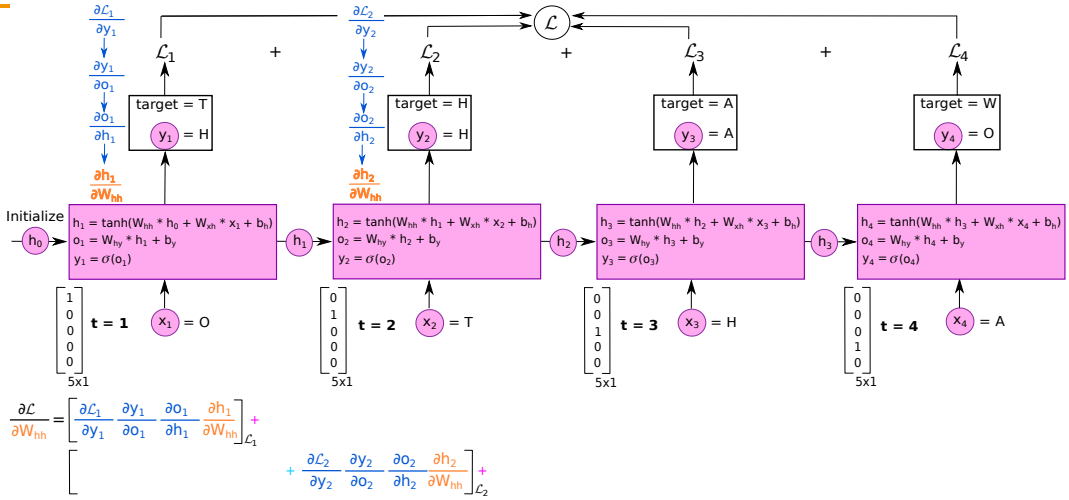
Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hh}, W_{xh} and b_h b_h



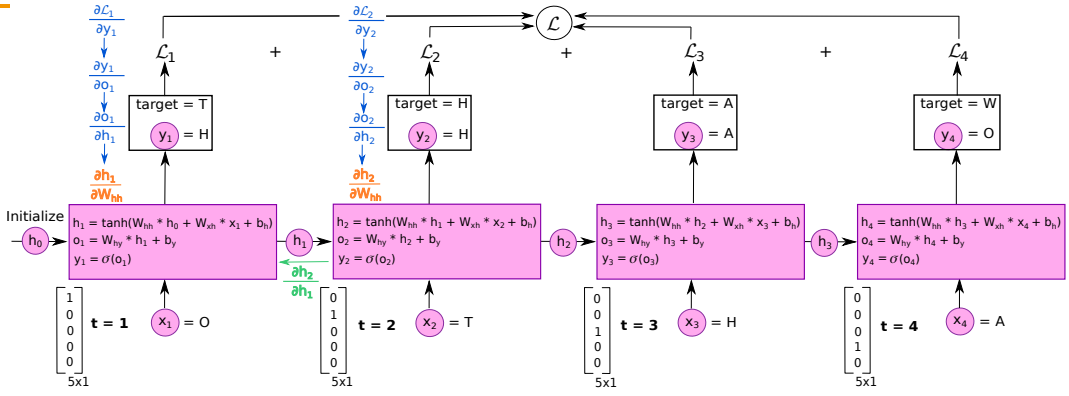
Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hh}, W_{xh} and b_h b_h



Backpropagation Through Time (BPTT)

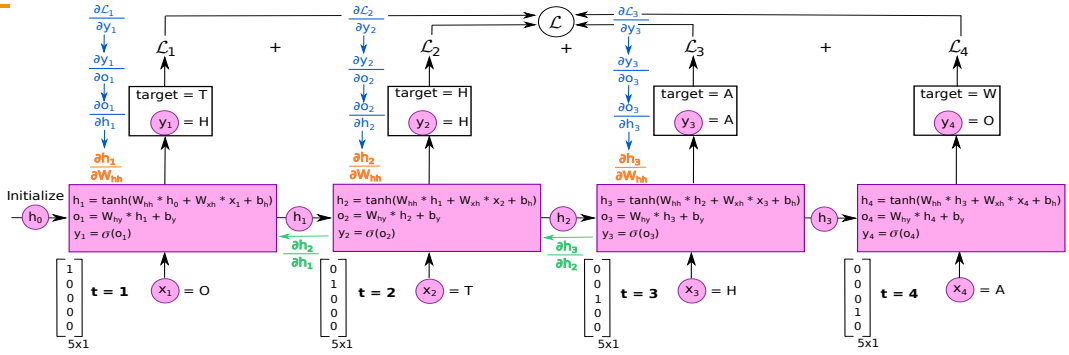
Network Training – Optimization of W_{hh}, W_{xh} and b_h



$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \left[\frac{\partial \mathcal{L}_1}{\partial y_1}\frac{y_1}{o_1}\frac{o_1}{h_1}\frac{h_1}{W_{hh}}\right]_{\mathcal{L}_1} + \left[\frac{\partial \mathcal{L}_2}{\partial y_2}\frac{y_2}{o_2}\frac{o_2}{h_2}\frac{h_2}{h_1}\frac{h_1}{W_{hh}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{y_2}{o_2}\frac{o_2}{h_2}\frac{h_2}{W_{hh}}\right]_{\mathcal{L}_2} +$$

Backpropagation Through Time (BPTT)

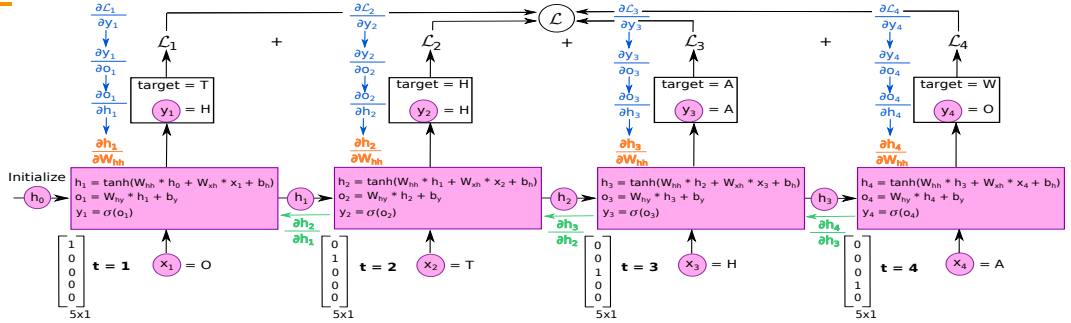
Network Training – Optimization of W_{hh}, W_{xh} and b_h



$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \left[\frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}}\right]_{\mathcal{L}_1} + \left[\frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}}\right]_{\mathcal{L}_2} + \left[\frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}\right]_{\mathcal{L}_3}$$

Backpropagation Through Time (BPTT)

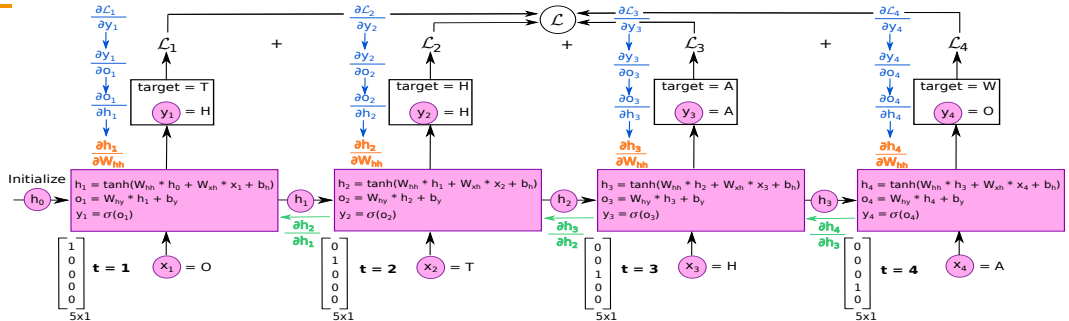
Network Training – Optimization of W_{hh}, W_{xh} and b_h



$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \left[\frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}}\right]_{\mathcal{L}_1} + \left[\frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}}\right]_{\mathcal{L}_2} + \left[\frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}\right]_{\mathcal{L}_3} + \left[\frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial W_{hh}}\right]_{\mathcal{L}_4}$$

Backpropagation Through Time (BPTT)

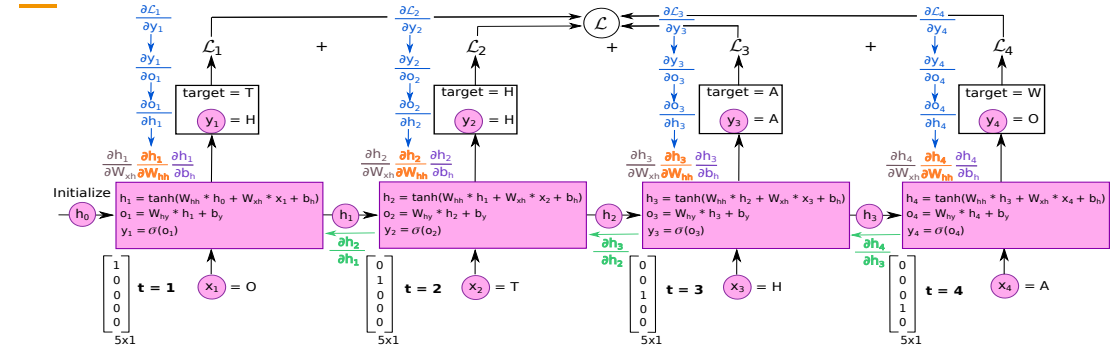
Network Training – Optimization of W_{hh}, W_{xh} and b_h



$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \left(\prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j} \right) \frac{\partial h_k}{\partial W_{hh}}$$

Backpropagation Through Time (BPTT)

Network Training – Optimization of W_{hh}, W_{xh} and b_h



$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \left[\frac{\partial \mathcal{L}_1}{\partial y_1}\frac{\partial y_1}{\partial o_1}\frac{\partial o_1}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}}\right]_{\mathcal{L}_1} + \left[\frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_2}{\partial y_2}\frac{\partial y_2}{\partial o_2}\frac{\partial o_2}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}}\right]_{\mathcal{L}_2} +$$

$$\left[\frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}} + \frac{\partial \mathcal{L}_3}{\partial y_3}\frac{\partial y_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}\right]_{\mathcal{L}_3} +$$

$$\left[\frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}} + \frac{\partial \mathcal{L}_4}{\partial y_4}\frac{\partial y_4}{\partial o_4}\frac{\partial o_4}{\partial h_4}\frac{\partial h_4}{\partial W_{hh}}\right]_{\mathcal{L}_4}$$

$$\frac{\partial \mathcal{L}}{\partial W_{xh}|\,\partial W_{hh}|\,\partial b_h} = \sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial \mathcal{L}_t}{\partial y_t}\frac{\partial y_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\left(\prod_{j=k}^{t-1}\frac{\partial h_{j+1}}{\partial h_j}\right)\frac{\partial h_k}{\partial W_{xh}|\,\partial W_{hh}|\,\partial b_h}$$

Backpropagation Through Time (BPTT)

Network Training – Challenges

- Long-Term Dependency Problem: long temporal sequences lead to exploding or disappearing gradients in the BPTT (see chain rule about the hidden state h_t)

$$\frac{\partial \mathcal{L}}{\partial W_{xh} | \partial W_{hh} | \partial b_h} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \left(\prod_{j=k}^{t-1} \frac{\partial h_{j+1}}{\partial h_j} \right) \frac{\partial h_k}{\partial W_{xh} | \partial W_{hh} | \partial b_h}$$

Vanishing/Exploding Gradient

- Information from previous states has no influence on the present and therefore does not allow long-term memory but only short-term dependencies
- Calculation of gradients is very time-consuming & resource-intensive for long sequences

Backpropagation Through Time (BPTT)

Network Training – Challenges and Solutions

Solutions

- Gradient-Clipping in case of exploding gradients

Backpropagation Through Time (BPTT)

Network Training – Challenges and Solutions

Solutions

- Gradient-Clipping in case of exploding gradients
- Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU), Rectified Linear Unit (ReLU), and/or specific activation functions to counteract vanishing gradients (see also modern Transformer architectures)

Solutions

- Gradient-Clipping in case of exploding gradients
- Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU), Rectified Linear Unit (ReLU), and/or specific activation functions to counteract vanishing gradients (see also modern Transformer architectures)
- Calculating the gradients for an entire (long) sequence is very time-consuming and resource-intensive $\rightarrow$ Truncated Backpropagation Through Time (TBPTT) to save time and resources

Recurrent Neural Network

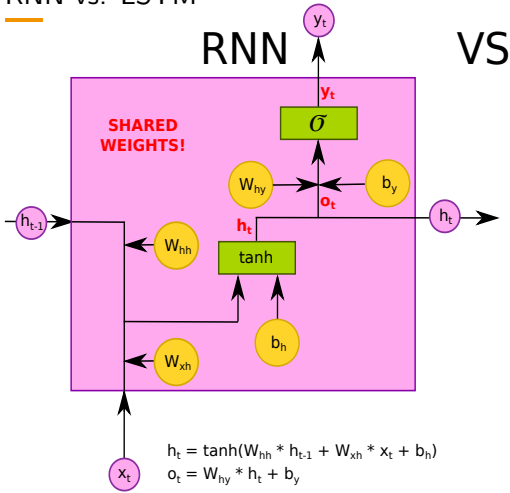
It's not Magic, it's Deep Learning

OTH Amberg-Weiden

- Recurrent neural networks modeling compared to feed-forward models neuron interconnections over multiple layers, similar to a human memory
- RNNs enable the encoding of sequential information at different times
- Terminology "Backpropagation Through Time (BPTT)" is used in connection with the training of recurrent neural networks → Counterpart to the traditional backpropagation algorithm in feed-forward models
- The addition "Through Time" concerns the gradients and their temporal dependency during parameter optimization (→ "shared weights")
- Training of RNN-based models via BPTT proves to be difficult (exploding/disappearing gradients, time and resource consuming) → Approaches/models to counteract these problems!

Long-Short-Term Memory Cell (LSTM)

RNN vs. LSTM



Source: Images taken from link and FAU PRL Lecture Notes: Recurrent Neural Networks Part 1 – 5 (Elman Cell, BPTT, LSTM, GRU) – link

Long-Short-Term Memory Cells (LSTMs)

Architecture

- Input Gate:

  $I_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$
- Forget Gate:

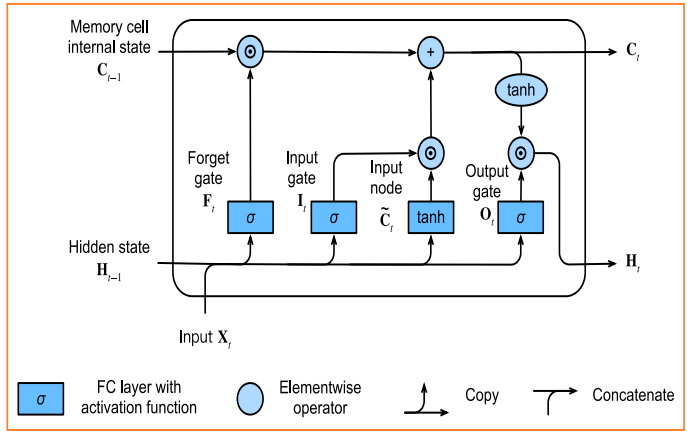
  $F_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$
- Output Gate:

  $O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$
- Internal State/Node:

  $\tilde{C}_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$
- Memory Cell State:

  $C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$
- Hidden State:

  $h_t = o_t \odot tanh(C_t)$



Source: Image, https://d2l.ai/chapter_recurrent-modern/lstm.html

Gated Recurrent Units (GRUs)

Architecture

- Reset Gate:

  $R_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$

- Update Gate:

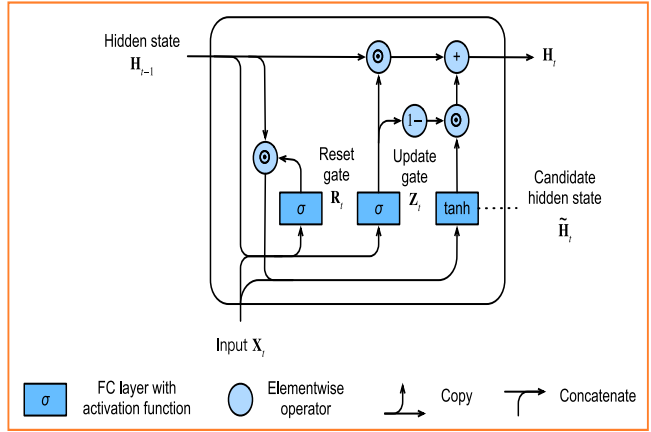
  $Z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$

- Candidate Hidden State:

  $\tilde{h}_t = tanh(W_{xh}x_t + (R_t \odot h_{t-1})W_{hh} + b_h)$

- Hidden State:

  $h_t = Z_t \odot h_{t-1} + (1 - Z_t) \odot \tilde{h}_t$



Source: Image, https://d2l.ai/chapter_recurrent-modern/gru.html

LSTMs and GRUs

General Facts

- **Parametric Complexity and Lack of Interpretability**
  Both are more complex compared to simpler RNN architectures, which does not only increase the training times, but the cell-specific architectures make the models challenging to interpret

LSTMs and GRUs

General Facts

OTH Amberg-Weiden

- **Parametric Complexity and Lack of Interpretability**
  Both are more complex compared to simpler RNN architectures, which does not only increase the training times, but the cell-specific architectures make the models challenging to interpret
- **Overfitting**
  Due to their larger number of parameters, LSTM/GRU models are prone to overfitting

LSTMs and GRUs

General Facts

- **Parametric Complexity and Lack of Interpretability**
  Both are more complex compared to simpler RNN architectures, which does not only increase the training times, but the cell-specific architectures make the models challenging to interpret
- **Overfitting**
  Due to their larger number of parameters, LSTM/GRU models are prone to overfitting
- **Memory Requirements and Computational Costs**
  LSTM and GRU models are more memory-intensive compared to simpler RNN models to store their additional parameters and require significantly more computational resources, especially for huge data corpora

LSTMs and GRUs

General Facts

- Parametric Complexity and Lack of Interpretability

  Both are more complex compared to simpler RNN architectures, which does not only increase the training times, but the cell-specific architectures make the models challenging to interpret
- Overfitting

  Due to their larger number of parameters, LSTM/GRU models are prone to overfitting
- Memory Requirements and Computational Costs

  LSTM and GRU models are more memory-intensive compared to simpler RNN models to store their additional parameters and require significantly more computational resources, especially for huge data corpora
- Lack of Parallelism

  LSTM and GRU models process data in a sequential manner, which limits parallelism. This can be a bottleneck for efficient training and inference, especially for long sequences

LSTMs and GRUs

General Facts

- Parametric Complexity and Lack of Interpretability

  Both are more complex compared to simpler RNN architectures, which does not only increase the training times, but the cell-specific architectures make the models challenging to interpret
- Overfitting

  Due to their larger number of parameters, LSTM/GRU models are prone to overfitting
- Memory Requirements and Computational Costs

  LSTM and GRU models are more memory-intensive compared to simpler RNN models to store their additional parameters and require significantly more computational resources, especially for huge data corpora
- Lack of Parallelism

  LSTM and GRU models process data in a sequential manner, which limits parallelism. This can be a bottleneck for efficient training and inference, especially for long sequences
- Vanishing/Exploding Gradients

  Despite gating mechanisms, internal memory cell states, LSTM/GRU models are still affected by vanishing/exploding gradients, particularly with very deep architectures and/or long sequences

Transformer – Optimus Prime

Let's Dive Into the Concept of Attention and Transformer Models...

Attention is All you Need!!!

- Original Work on Transformer: Paper Attention is All you Need

Source: Image from https://medium.com/@mladenkorunoski/transformers-9e9e76572b77

Deep Learning – Transformer Architecture

Let's Dive Into the Concept of Attention and Transformer Models...

- Transformer...? → One step after each other: Let's have a look at the following concepts first:

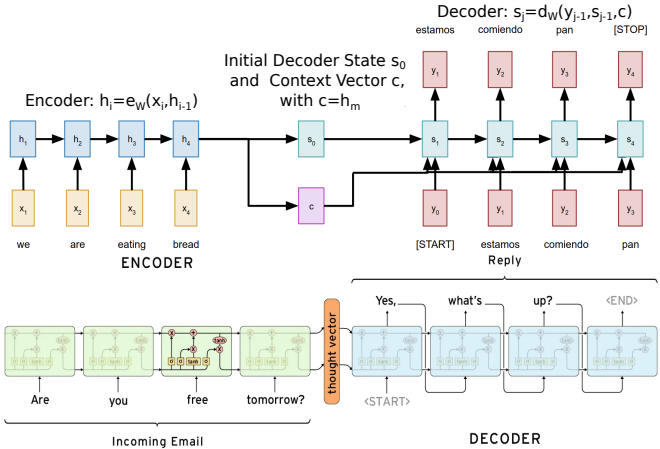
Deep Learning – Transformer Architecture

Let's Dive Into the Concept of Attention and Transformer Models...

- Transformer...? → One step after each other: Let's have a look at the following concepts first:
  → Sequence-to-Sequence Learning, Attention, Self-Attention, Multi-Head Attention, and the Transformer Architecture

Traditional Sequence-to-Sequence Learning

General Idea

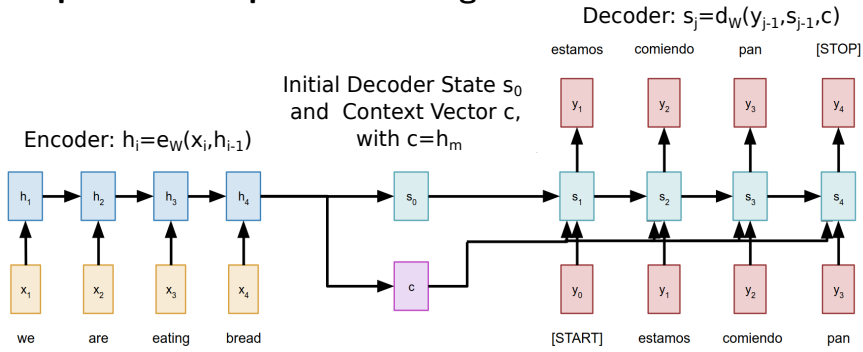


- RNN-Based Encoder-Decoder Structure applied to a sequential input sequence
- Encoder part encodes a given input sequence, while the (final) model output (hidden state = "memory") is used to initialize the decoder part → Different architectural designs (Cho et al. or Sutskever et al.)!

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf
Source: Image from https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8

Traditional Sequence-to-Sequence Learning

General Idea

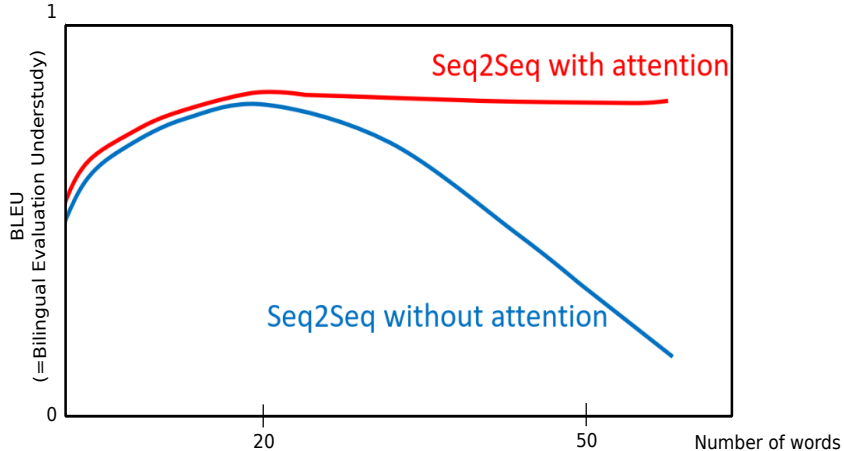


- Machine Translation, Image Captioning, Text Summarization, Question Answering/Chatbot, etc.
- Critical Disadvantage: the final (hidden) state can not remember long sequences (no long-term memory!), introducing artifacts while using this information to derive and decode the output sequence
- How to counteract this long-term dependency phenomenon?

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Traditional Sequence-to-Sequence Learning

General Idea



- BLEU = Algorithm for evaluating the quality of text which has been machine-translated from one language to another

Attention Mechanism

What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)

Attention Mechanism

What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)
  General concept:
  Rather than feeding the same context vector c to every time step within the decoder part, the idea is to introduce different context vectors c_j at each decoder step

Attention Mechanism

What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)
  General concept:
  Rather than feeding the same context vector c to every time step within the decoder part, the idea is to introduce different context vectors c_j at each decoder step
- Idea of Attention:
  - The encoder produces an individual output representation h_i at each encoding step (in this case the hidden state itself)

Attention Mechanism

What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)
  General concept:
  Rather than feeding the same context vector c to every time step within the decoder part, the idea is to introduce different context vectors c_j at each decoder step
- Idea of Attention:
  - The encoder produces an individual output representation h_i at each encoding step (in this case the hidden state itself)
  - The decoder receives at each decoding step j an individual context vector c_j

Attention Mechanism

What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)
  General concept:
  Rather than feeding the same context vector c to every time step within the decoder part, the idea is to introduce different context vectors c_j at each decoder step
- Idea of Attention:
  - The encoder produces an individual output representation h_i at each encoding step (in this case the hidden state itself)
  - The decoder receives at each decoding step j an individual context vector c_j
  - The context vector c_j represents a weighted sum $c_j = \alpha_1 h_1 + \ldots + \alpha_m h_m$, between $\alpha_1, \ldots, \alpha_m$ as normalized weights ($0/1-$Softmax Output) and the respective output representations h_i at the encoding step $1, \ldots, m$

Attention Mechanism

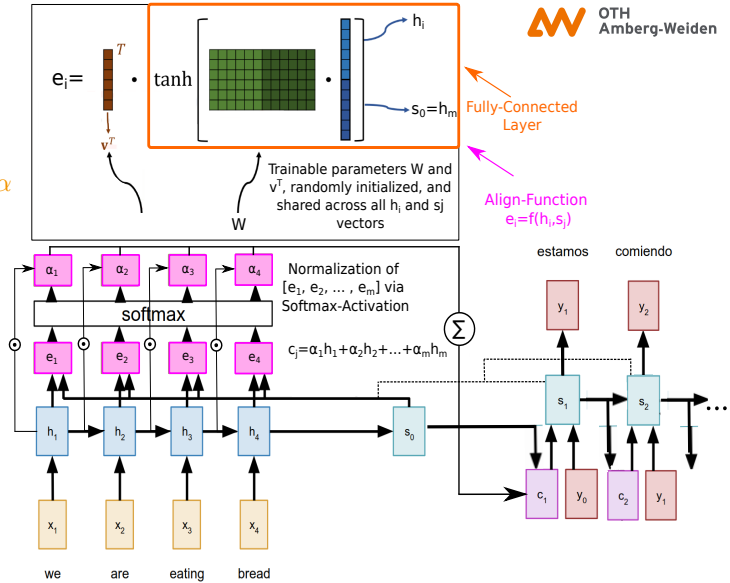
What it is about?

- Attention mechanism: originally introduced to improve and enhance the encoder-decoder RNN architectures for sequence-to-sequence learning (Bahdanau et al.)
  General concept:
  Rather than feeding the same context vector c to every time step within the decoder part, the idea is to introduce different context vectors c_j at each decoder step
- Idea of Attention:
  - The encoder produces an individual output representation h_i at each encoding step (in this case the hidden state itself)
  - The decoder receives at each decoding step j an individual context vector c_j
  - The context vector c_j represents a weighted sum $c_j = \alpha_1 h_1 + \ldots + \alpha_m h_m$, between $\alpha_1, \ldots, \alpha_m$ as normalized weights ($0/1-$Softmax Output) and the respective output representations h_i at the encoding step $1, \ldots, m$
  - The α values (Attention Weights) allow to pay a specific Attention (Focus) on particular important parts of the encoder input sequence at each time step due to individual weighting

Attention Mechanism

What it is about?

- Different algorithms to build the input-specific Attention Weights α
- Context Vector c_j:
  $c_j = \alpha_1 h_1 + \ldots + \alpha_m h_m$
- For each output state in the decoder s_j a new set of α parameters is computed via the Align Function $f \rightarrow \alpha_i = f(h_i, s_j)$
- Therefore, also a new Context Vector c_j is generated, paying Attention to different input characteristics

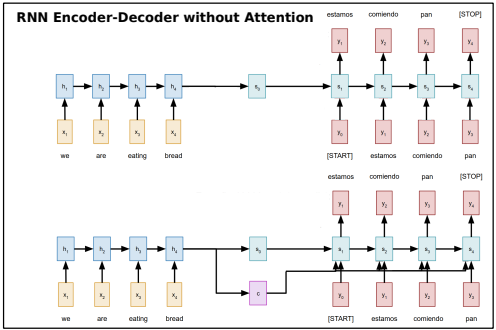


Source: Image from https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

- In simple RNN-based encoder-decoder architectures without attention Context Vectors c_j are often used in different ways



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

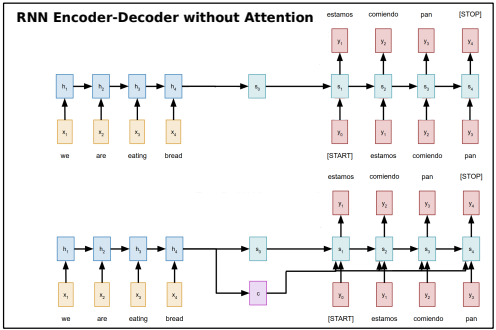
Attention Mechanism

What it is about?

- In simple RNN-based encoder-decoder architectures without attention Context Vectors c_j are often used in different ways

  1) The model does not use an additional Context Vectors c_j as input in the decoder stage to compute the next decoder state

$$s_j = tanh\left(W \cdot \begin{bmatrix} y_{j-1} \\ s_{j-1} \end{bmatrix} + b\right)$$



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

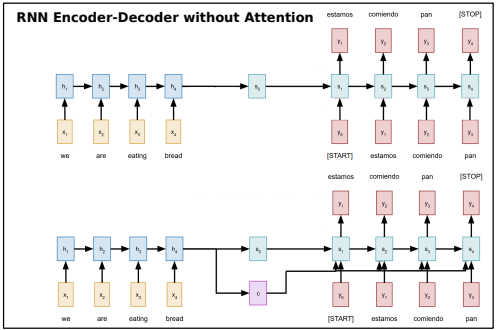
- In simple RNN-based encoder-decoder architectures without attention Context Vectors c_j are often used in different ways

  1) The model does not use an additional Context Vectors c_j as input in the decoder stage to compute the next decoder state

$$s_j = tanh\left(W \cdot \begin{bmatrix} y_{j-1} \\ s_{j-1} \end{bmatrix} + b\right)$$

  2) The model uses an additional Context Vectors c_j as input in the decoder stage, which is equal to the final encoder output state h_m and shared across all decoder state

$$s_j = tanh\left(W \cdot \begin{bmatrix} y_{j-1} \\ s_{j-1} \\ c = h_m \end{bmatrix} + b\right)$$



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

- In simple RNN-based encoder-decoder architectures without attention Context Vectors c_j are often used in different ways

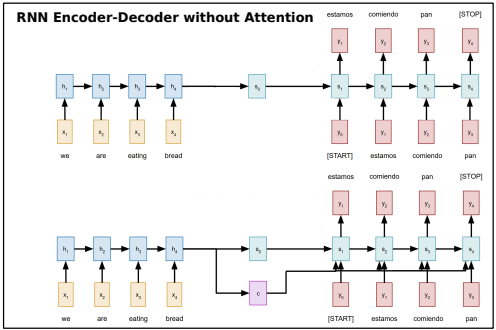
  1) The model does not use an additional Context Vectors c_j as input in the decoder stage to compute the next decoder state

  $$s_j = tanh\left(W \cdot \begin{bmatrix} y_{j-1} \\ s_{j-1} \end{bmatrix} + b\right)$$

  2) The model uses an additional Context Vectors c_j as input in the decoder stage, which is equal to the final encoder output state h_m and shared across all decoder state

  $$s_j = tanh\left(W \cdot \begin{bmatrix} y_{j-1} \\ s_{j-1} \\ c = h_m \end{bmatrix} + b\right)$$

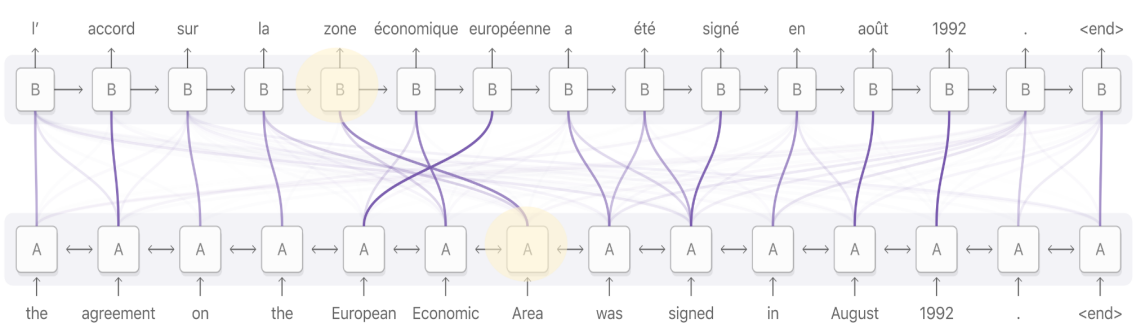
- The model architecture with attention uses individual Context Vectors c_j, which are built on the weighted sum between "learned" attention weights α_i and the respective encoder output h_i, resulting in $c_j = \alpha_1 h_1 + \ldots + \alpha_m h_m$, together with the previous (hidden) state s_{j-1} and current input y_{j-1} compute s_t



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

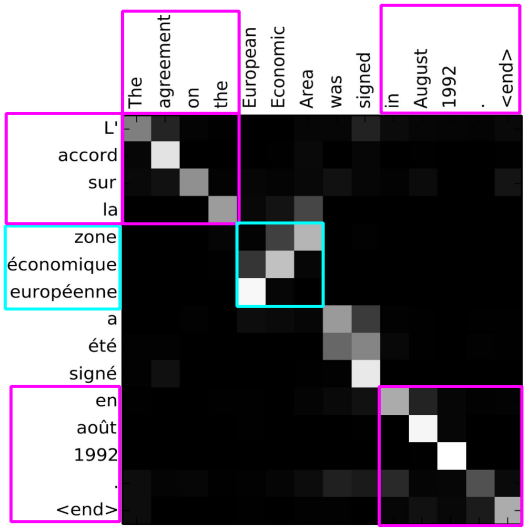


- Machine Translation – English vs. French – Attention Weights α show the focus!

Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf
Source: Image from https://distill.pub/2016/augmented-rnns/

Attention Mechanism

What it is about?



- Machine Translation – Diagonal attentions refer to the ordered word correspondences (pink boxes), while there exist also attentions ignoring the word ordering (blue box)

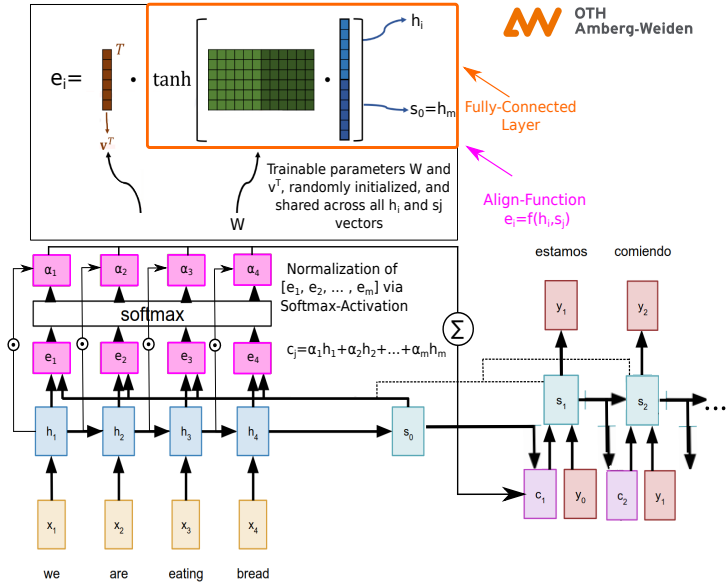
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

- The decoder treats the hidden states as an unordered set because it relies on the attention mechanism to capture dependencies between tokens, rather than explicitly utilizing the positional information of the hidden states

  → Therefore, any set of input is possible! Does not have to be a sequentially-ordered input!



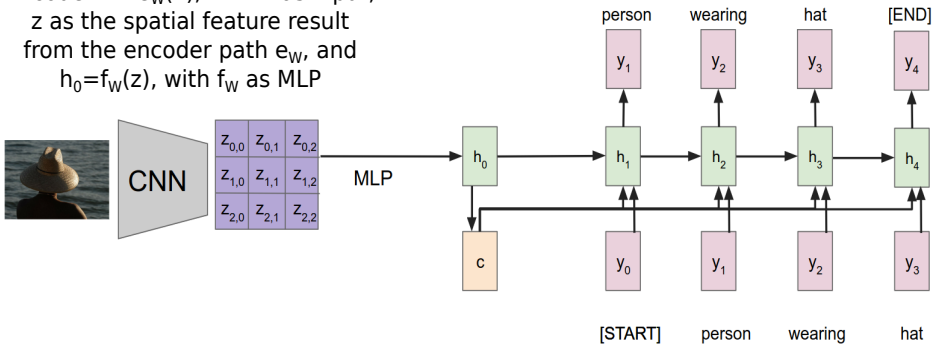
Source: Image from https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

Encoder: $z=e_W(x)$, with x as input, z as the spatial feature result from the encoder path e_W, and $h_0=f_W(z)$, with f_W as MLP

Decoder: $h_j=d_W(y_{j-1},s_{j-1},c)$, with $c=h_0$

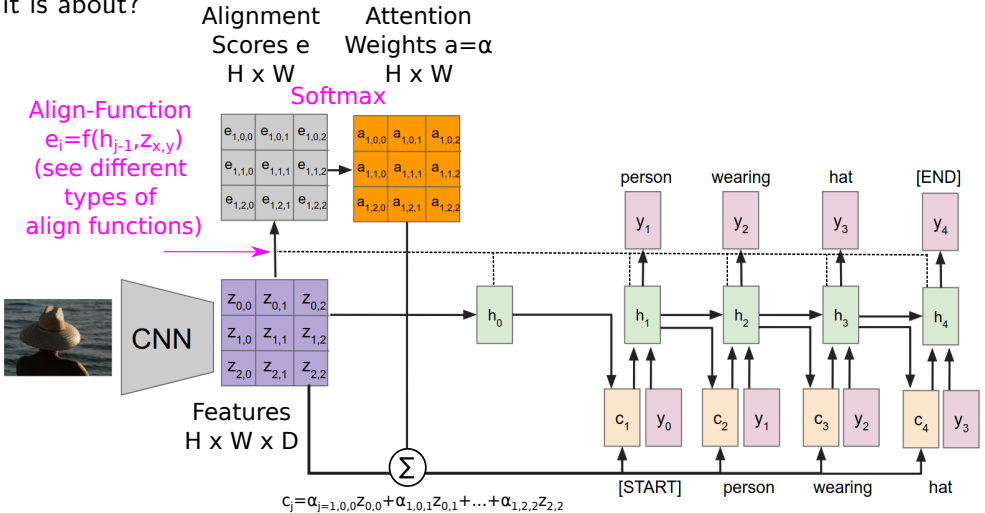


- Image Captioning – Encoder path is realized via a CNN-based model, while the final feature representation z is linearized via a fully-connected architecture (Multi-Layer-Perceptron – MLP)
- Linearized output represents the shared context $c = h_0$ and initial RNN (hidden) state h_0
- No Attention!!! → Problem in case of really long image descriptions, all the information in c!

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

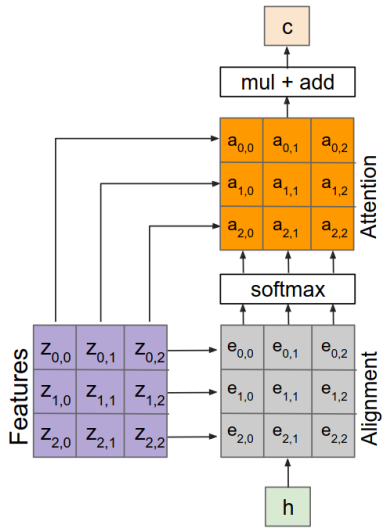
Source: Image from Xu et al, Neural Image Caption Generation with Visual Attention, https://proceedings.mlr.press/v37/xuc15.html

Attention Mechanism

What it is about?

Attention in Image Captioning

- Input features z of shape $H \times W \times D$ and Query h_j of shape $D \times 1$
- Computations:
  - Align function: $e_{x,y} = align(h_{j-1}, z_{x,y})$
  - Attention weights: $a = \alpha = Softmax(e)$
  - Context Vector: $c_j = \sum_{x,y} \alpha_{x,y} z_{x,y}$
- Output context vector c_j of shape $D \times 1$



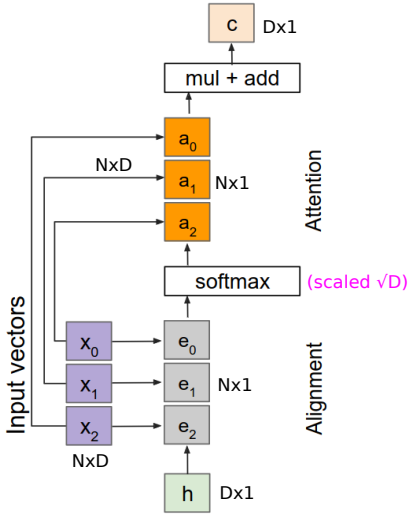
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- The attention mechanism/operation is permutation invariant



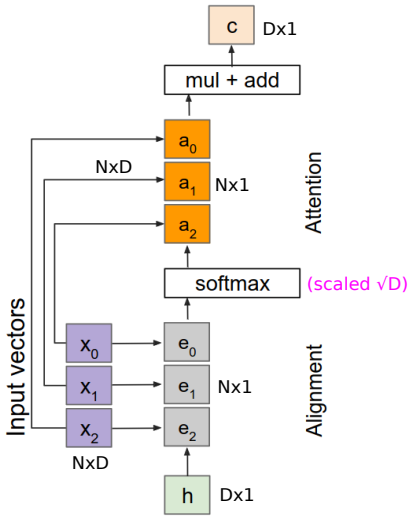
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- The attention mechanism/operation is permutation invariant
- 3D-Feature maps of shape $H \times W \times D$ can be stretched/linearized via $N = H \times W$, resulting in an input feature vector x of shape $N \times D$



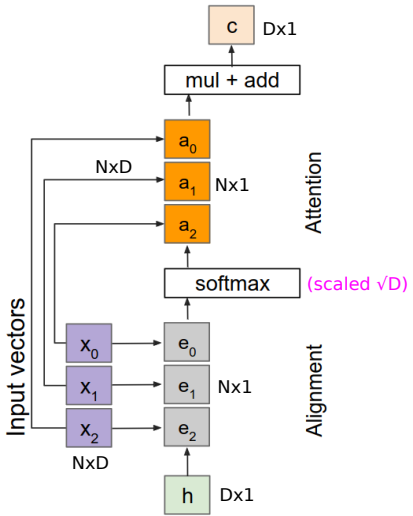
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- The attention mechanism/operation is permutation invariant
- 3D-Feature maps of shape $H \times W \times D$ can be stretched/linearized via $N = H \times W$, resulting in an input feature vector x of shape $N \times D$
- Computations:
  - Align function: $e_i = align(h_j, x_i)$, with *align* as a scaled dot product, leading to $e_i = \frac{h_j \cdot x_i}{\sqrt{D}}$ of shape $N \times 1 \rightarrow$ Large dot-product, higher vectorial similarity, lower post-softmax entropy (confident)!
  - Attention weights: $a = \alpha = Softmax(e)$
  - Context Vector: $c_j = \sum_i \alpha_i x_i$



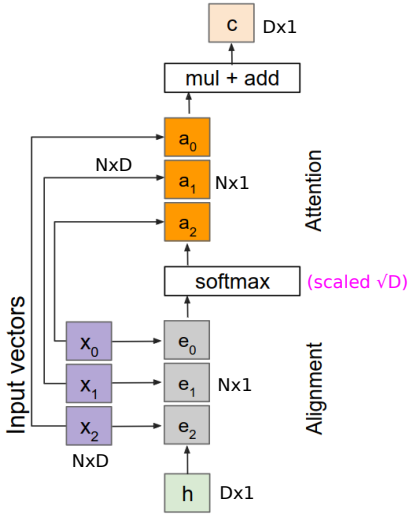
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- The attention mechanism/operation is permutation invariant
- 3D-Feature maps of shape $H \times W \times D$ can be stretched/linearized via $N = H \times W$, resulting in an input feature vector x of shape $N \times D$
- Computations:
  - Align function: $e_i = align(h_j, x_i)$, with *align* as a scaled dot product, leading to $e_i = \frac{h_j \cdot x_i}{\sqrt{D}}$ of shape $N \times 1 \rightarrow$ Large dot-product, higher vectorial similarity, lower post-softmax entropy (confident)!
  - Attention weights: $a = \alpha = Softmax(e)$
  - Context Vector: $c_j = \sum_i \alpha_i x_i$
- Output context vector c_j of shape $D \times 1$



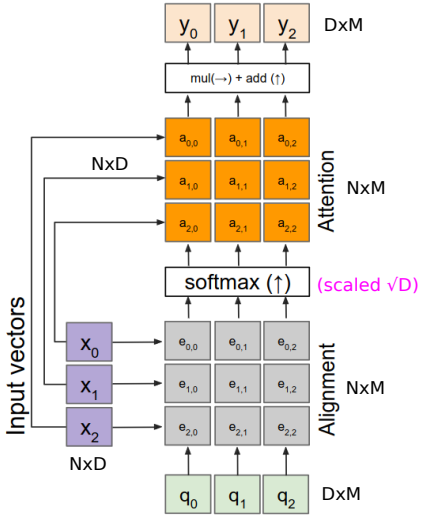
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Extend to multiple query-patterns, while each additional query vector produces a new context vector $y_j = c_j$



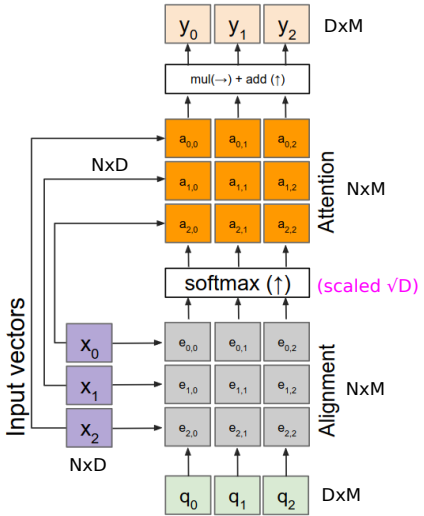
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Extend to multiple query-patterns, while each additional query vector produces a new context vector $y_j = c_j$
- Input vector matrix x with shape $N \times D$ is used for both, the algin function, next to the attention calculations



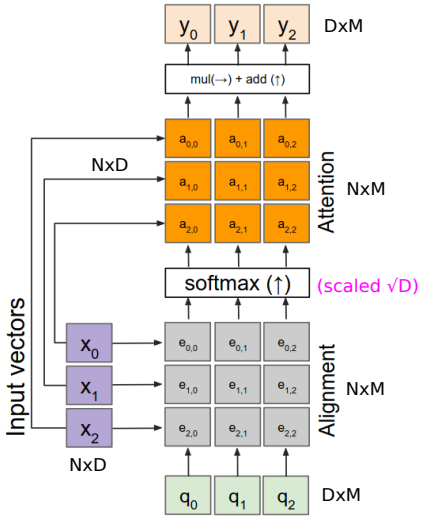
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Extend to multiple query-patterns, while each additional query vector produces a new context vector $y_j = c_j$
- Input vector matrix x with shape $N \times D$ is used for both, the algin function, next to the attention calculations
- Adding more expressivity by introducing a different fully-connected layer, before each of the two steps



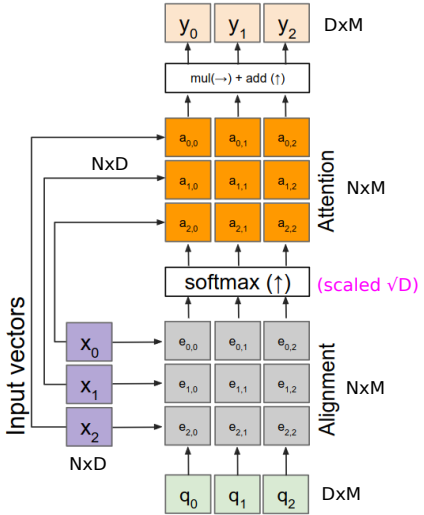
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Extend to multiple query-patterns, while each additional query vector produces a new context vector $y_j = c_j$
- Input vector matrix x with shape $N \times D$ is used for both, the algin function, next to the attention calculations
- Adding more expressivity by introducing a different fully-connected layer, before each of the two steps
- Introduction of the Query, Key, Value concept, which is nowadays used in the modern Transformer architectures



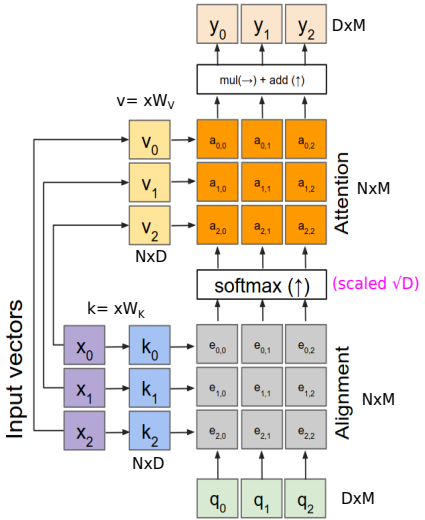
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Introducing two additional computation steps, by $k = xW_K$ and $v = xW_V$, both using the input x and two different weight matrices W_K and W_V, to compute the Keys and Values



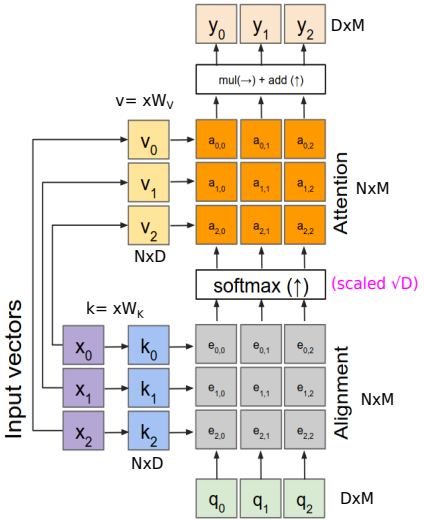
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Introducing two additional computation steps, by $k = xW_K$ and $v = xW_V$, both using the input x and two different weight matrices W_K and W_V, to compute the Keys and Values
- Align function: $e_i = align(q_j, k_i)$, with *align* as a scaled dot product, leading to $e_{i,j} = \frac{q_j \cdot k_i}{\sqrt{D}}$ of shape $N \times M$



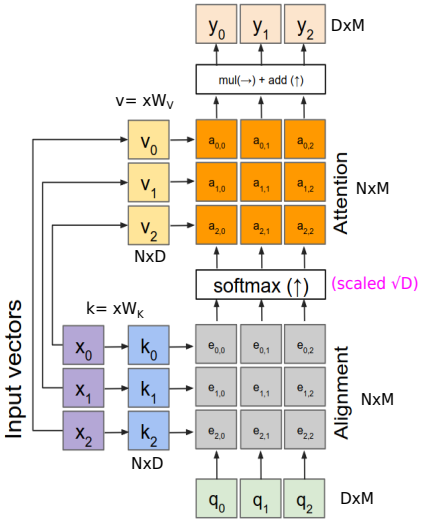
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Introducing two additional computation steps, by $k = xW_K$ and $v = xW_V$, both using the input x and two different weight matrices W_K and W_V, to compute the Keys and Values
- Align function: $e_i = align(q_j, k_i)$, with *align* as a scaled dot product, leading to $e_{i,j} = \frac{q_j \cdot k_i}{\sqrt{D}}$ of shape $N \times M$
- How to also get rid of the external Query vectors q_j (recap: from updated hidden states s_j / h_j)?



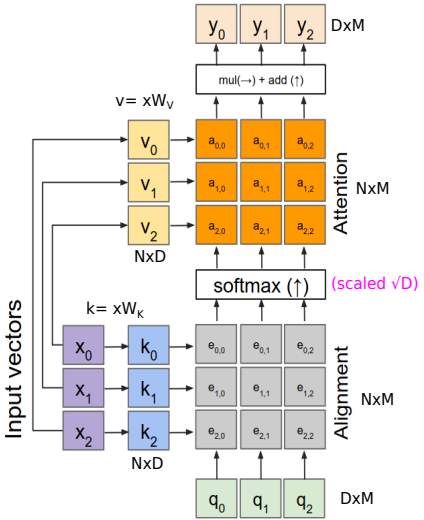
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer

- Introducing two additional computation steps, by $k = xW_K$ and $v = xW_V$, both using the input x and two different weight matrices W_K and W_V, to compute the Keys and Values
- Align function: $e_i = align(q_j, k_i)$, with *align* as a scaled dot product, leading to $e_{i,j} = \frac{q_j \cdot k_i}{\sqrt{D}}$ of shape $N \times M$
- How to also get rid of the external Query vectors q_j (recap: from updated hidden states s_j / h_j)?
- Solution: Replace the Query vectors q_j also via a fully-connected layer and the given input x



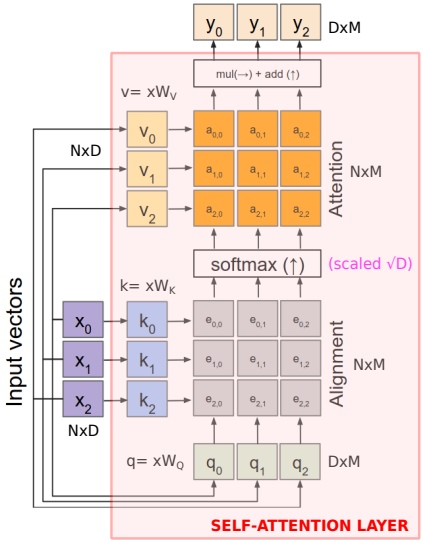
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer → Self-Attention!

- Introducing another computation step, by $q = xW_Q$, using the input x and a weight matrix W_Q, to compute the Keys and Values



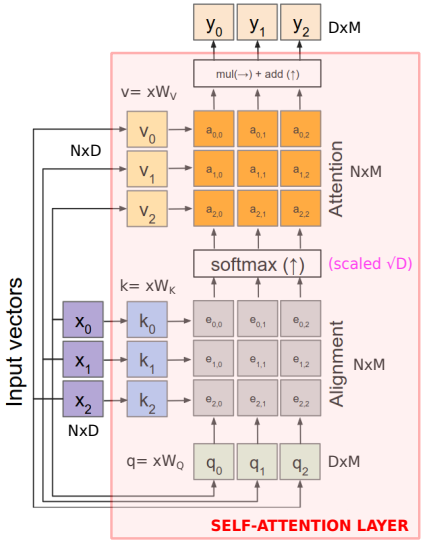
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer → Self-Attention!

- Introducing another computation step, by $q = xW_Q$, using the input x and a weight matrix W_Q, to compute the Keys and Values
- No input query vectors q_j anymore → Self attention layer: attends over sets of its own inputs



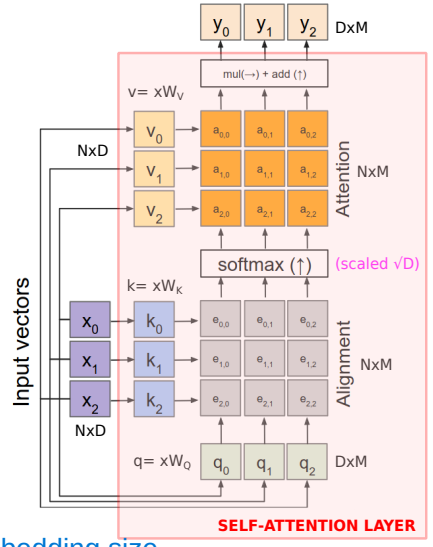
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention Layer → Self-Attention!

- Introducing another computation step, by $q = xW_Q$, using the input x and a weight matrix W_Q, to compute the Keys and Values
- No input query vectors q_j anymore → Self attention layer: attends over sets of its own inputs
- Computation Self-Attention Layer:
  - Key vectors: $k = xW_K$
  - Value vectors: $v = xW_V$
  - Query vectors: $q = xW_Q$
  - Align function: $e_{i,j} = \frac{q_i k_i}{\sqrt{D}}$
  - Attention: $a = \alpha = Softmax(e)$
  - Context Vector (Layer Output): $y_j = \sum_i \alpha_{i,j} v_i$



Source: https://web.stanford.edu/class/cs224n/slides/2022/lecture_11_ruohan.pdf

d is embedding size
m is number of words in a sentance

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- Inner Product:
  - $e_i = k_i^T \cdot q_j$, for $i = 1, ..., m$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- Inner Product:
  - $e_i = k_i^T \cdot q_j$, for $i = 1, ..., m$
- Normalization (Weighting Factors):
  - $\alpha_i = Softmax(e_i)$, for $i = 1, ..., m$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- Inner Product:
  - $e_i = k_i^T \cdot q_j$, for $i = 1, ..., m$
- Normalization (Weighting Factors):
  - $\alpha_i = Softmax(e_i)$, for $i = 1, ..., m$
- Weighted (Attention) Output:
  - Context Vector $c_j = \sum_{i=1}^{m} \alpha_i \cdot v_i$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- Inner Product:
  - $e_i = k_i^T \cdot q_j$, for $i = 1, ..., m$
- Normalization (Weighting Factors):
  - $\alpha_i = Softmax(e_i)$, for $i = 1, ..., m$
- Weighted (Attention) Output:
  - Context Vector $c_j = \sum_{i=1}^{m} \alpha_i \cdot v_i$

$\rightarrow$ Algorithm Known as – Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Different (More Modern) Algorithm to Compute the Attention Weights $\alpha = f(h_i, s_j)$

- Linear Maps:
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- Inner Product:
  - $e_i = k_i^T \cdot q_j$, for $i = 1, ..., m$
- Normalization (Weighting Factors):
  - $\alpha_i = Softmax(e_i)$, for $i = 1, ..., m$
- Weighted (Attention) Output:
  - Context Vector $c_j = \sum_{i=1}^{m} \alpha_i \cdot v_i$

sj and hi is same for our case equal to x

→ Algorithm Known as – Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$

→ Same Technique as in the Modern Transformer Architecture!

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html
Source: https://github.com/wangshusen/DeepLearning/blob/master/Slides/9_RNN_8.pdf

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$
- "Query (q_j)" – refers to the input that is being attended, searched, or queried for information

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$
- "Query (q_j)" – refers to the input that is being attended, searched, or queried for information
- "Key" (k_i) – represents the sequence of information against which the query q_j is compared to, determining the overlap, matching & relevance of possible keys and query

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$
- "Query (q_j)" – refers to the input that is being attended, searched, or queried for information
- "Key" (k_i) – represents the sequence of information against which the query q_j is compared to, determining the overlap, matching & relevance of possible keys and query
- "Value" (v_i) – corresponds to the (weighted) best-matching answer given query and keys

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$
- "Query (q_j)" – refers to the input that is being attended, searched, or queried for information
- "Key" (k_i) – represents the sequence of information against which the query q_j is compared to, determining the overlap, matching & relevance of possible keys and query
- "Value" (v_i) – corresponds to the (weighted) best-matching answer given query and keys

$\rightarrow [\alpha_1 = 0, \alpha_2 = 1, \ldots, \alpha_m = 0]$ = one-hot-vector, which is like a traditional database query

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

What it is about?

Attention Pooling $A(q_j, D = Dataset) = \sum_{i=1}^{m} \alpha_i(q_j, k_i)v_i$ – More Information and Details!

Idea of a Dictionary Look-Up: the similarity between a Query q_j and the Keys k_i in the dictionary is specified via the computed attention weights α_i, which define the degree of attention/focus and are multiplied with the Values v_i

- Dataset D – consider the data as dictionary $D = \{(k_1, v_1), (k_2, v_2), ..., (k_m, v_m)\}$
- "Query (q_j)" – refers to the input that is being attended, searched, or queried for information
- "Key" (k_i) – represents the sequence of information against which the query q_j is compared to, determining the overlap, matching & relevance of possible keys and query
- "Value" (v_i) – corresponds to the (weighted) best-matching answer given query and keys

weighted with attention score

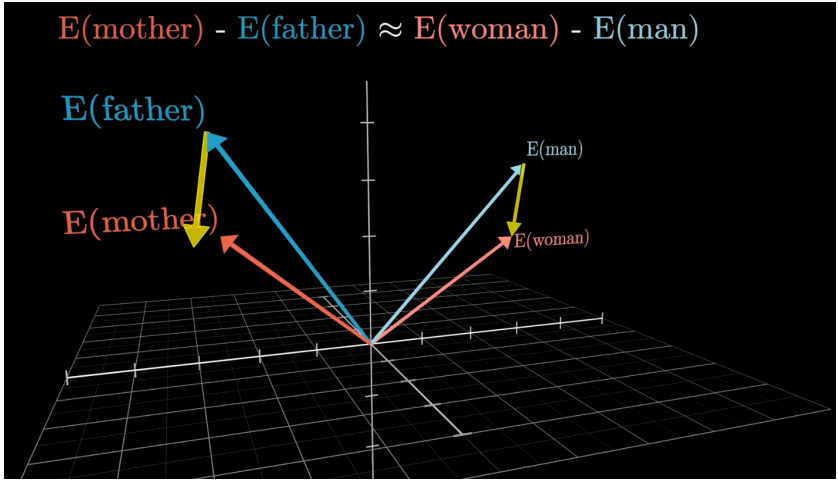
$\rightarrow [\alpha_1 = 0, \alpha_2 = 1, \ldots, \alpha_m = 0]$ = one-hot-vector, which is like a traditional database query

$\rightarrow [\alpha_1 = \frac{1}{m}, \alpha_2 = \frac{1}{m}, \ldots, \alpha_m = \frac{1}{m}]$ = average pooling, while every entry is equally important

Source: https://d2l.ai/chapter_attention-mechanisms-and-transformers/queries-keys-values.html

Attention Mechanism

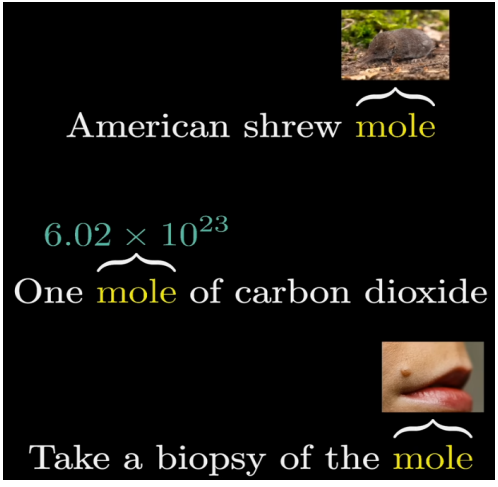
Motivation Word Embeddings



Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings

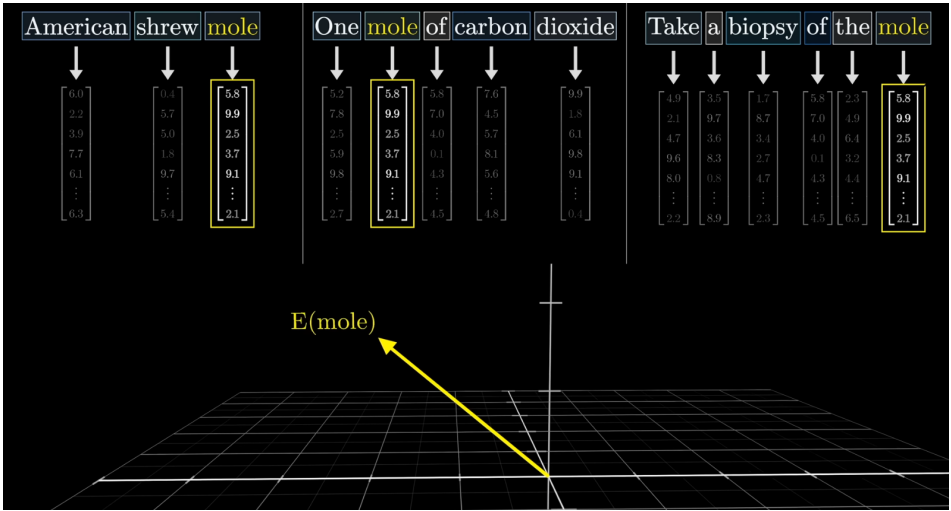


- Multiple semantic meanings for a single word! $\rightarrow$ How about the word embedding?

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings

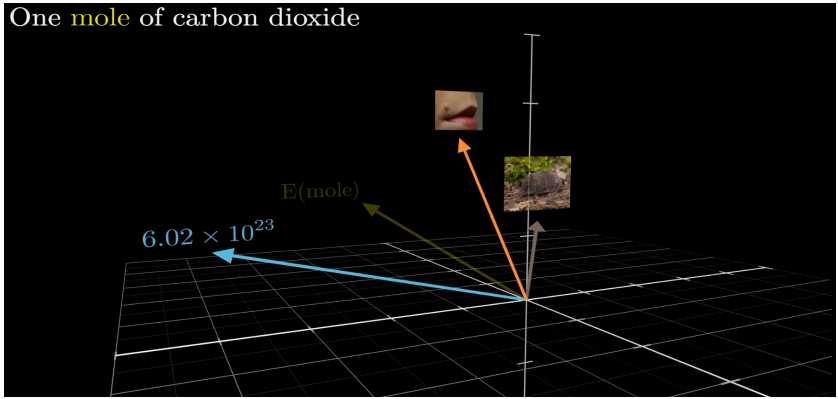


Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings

we need to add somethin gto the vector to move in vector space



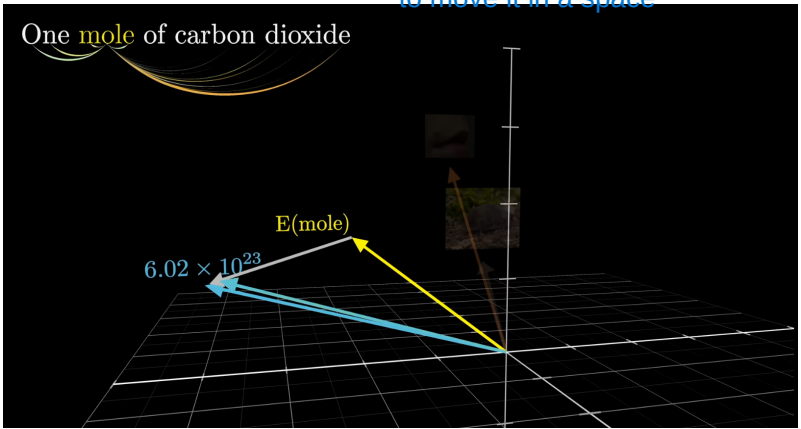
- Generic embedding for the stand-alone word “mole” is always the same!
- Based on the context it should have also different directions in the embedding space
- Surrounding context is important → Attention needs to be paid!

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings

the white vector is the value vector that are adding to the original vector to move it in a space

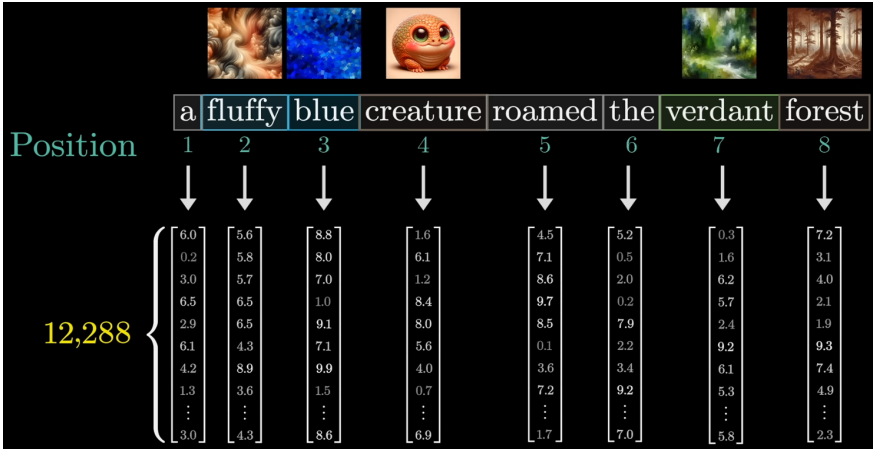


- Idea of Attention: "What you need to add to the generic embedding for the word "mole" to move it to the correct vectorial direction in space, based on the given context!"

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings

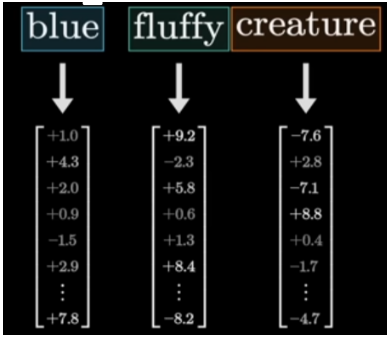


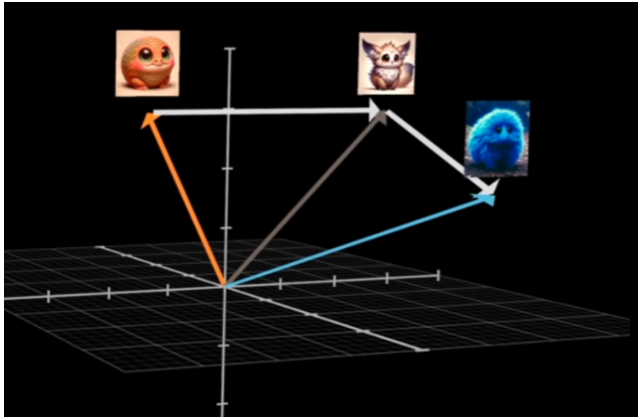
- Embedding vectors $\vec{E}_i$ $(1 \times D)$, with $D =$ embedding size for each single token (word)
- How to identify important semantic structures, relevant for a given word?

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Motivation Word Embeddings



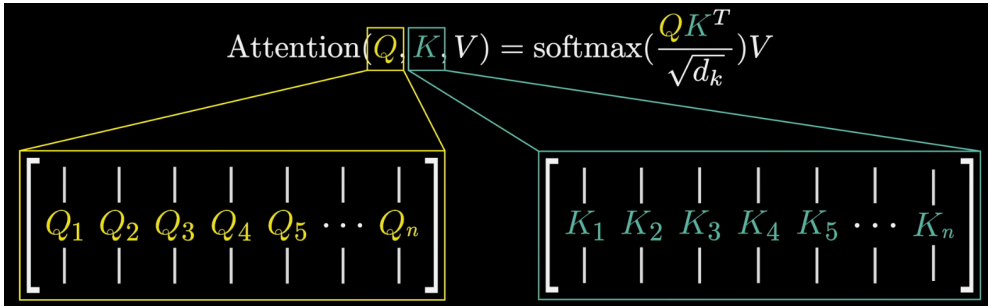


- Generic word embedding for “creature” (orange vector)
- According to the relevant semantic meaning, the word “creature” is affected within the vectorial embedding space by the words (embeddings) “fluffy” and “blue”

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Query & Key Concept



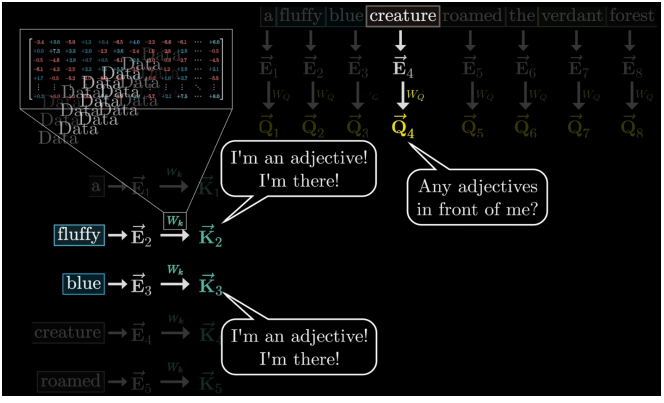
- $Q \cdot K^T = A$ (Raw Attention Scores!)
- $\sqrt{d_k}$ = Scaling Factor for Attention Scores (with d_k = Embedding Size per Head!) $\rightarrow$ In case of Multi-Head Attention $d_k = D_h = \frac{D}{H}$, with H = Number of Heads
- V = Value Matrix (later!)

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

d is embedding dimention and h is number of heads

Attention Mechanism

Query & Key Concept

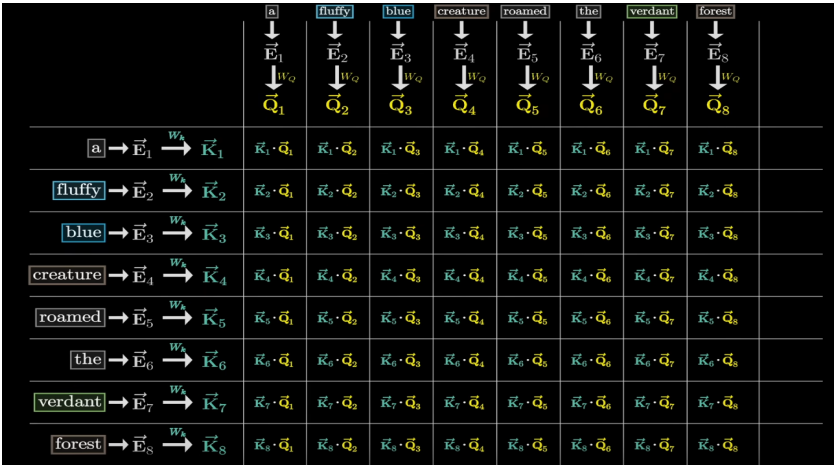


- Query $Q = W_Q \cdot E$, and $E(D_h \times M)$, with $M =$ Number of Tokens
- Key $K = W_K \cdot E$, and $E(D_h \times N)$, with $N =$ Number of Tokens
- Dimensions Parameters: W_Q, W_K, W_V $(D_h \times D_h)$, with $D_h = \frac{D}{H}$

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Query & Key Concept

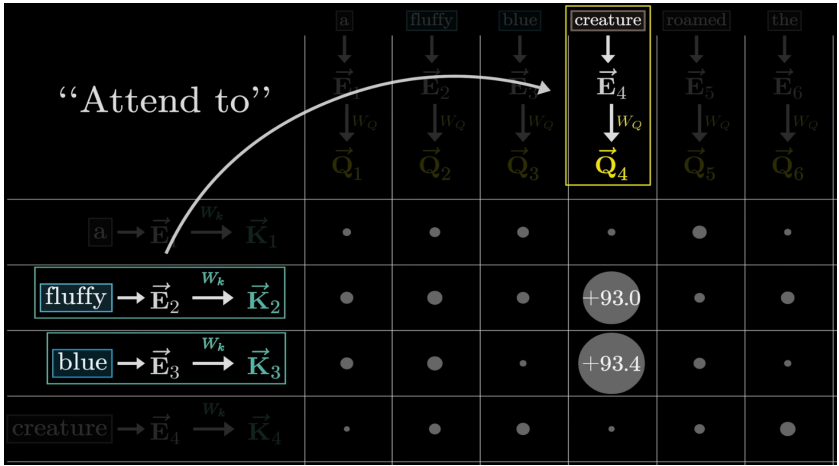


- $Q \cdot K^T = A(N \times M)$ (Raw Attention/Similarity Scores!) $\rightarrow$ Self-Attention $M = N$!

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Query & Key Concept



- Big values indicate (directional) similarity/importance ($\rightarrow$ How closely 2 vectors align!)

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

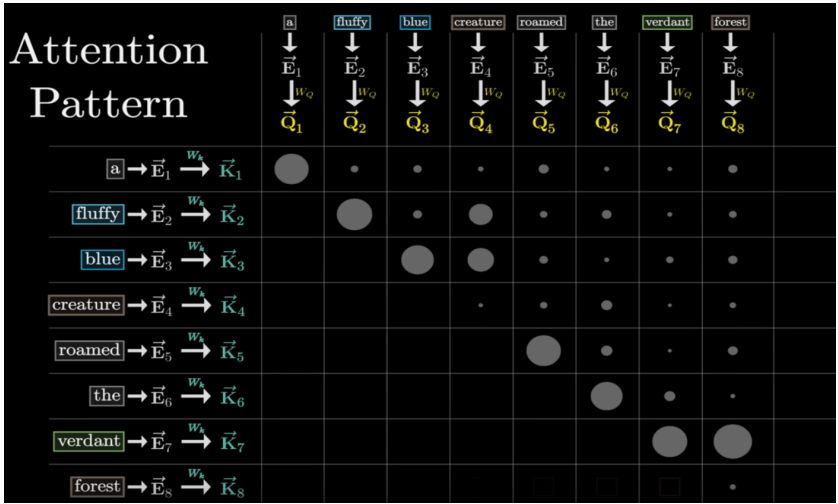
Query & Key Concept

	a	fluffy	blue	creature	roamed	the	verdant	forest
	$\vec{E}_1$	$\vec{E}_2$	$\vec{E}_3$	$\vec{E}_4$	$\vec{E}_5$	$\vec{E}_6$	$\vec{E}_7$	$\vec{E}_8$
	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$	$\downarrow W_Q$
	$\vec{Q}_1$	$\vec{Q}_2$	$\vec{Q}_3$	$\vec{Q}_4$	$\vec{Q}_5$	$\vec{Q}_6$	$\vec{Q}_7$	$\vec{Q}_8$
a $\vec{E}_1 \xrightarrow{W_K} \vec{K}_1$	+0.7	−83.7	−24.7	−27.8	−5.2	−89.3	−45.2	−36.1
fluffy $\vec{E}_2 \xrightarrow{W_K} \vec{K}_2$	−73.4	+2.9	−5.4	**+93.0**	−48.2	−87.3	−49.7	+7.8
blue $\vec{E}_3 \xrightarrow{W_K} \vec{K}_3$	−53.4	−5.7	+1.8	**+93.4**	−55.6	−56.0	−26.1	−62.1
creature $\vec{E}_4 \xrightarrow{W_K} \vec{K}_4$	−21.5	−29.7	−56.1	+4.9	−32.4	−92.3	−9.5	−28.1
roamed $\vec{E}_5 \xrightarrow{W_K} \vec{K}_5$	−20.1	−40.9	−87.8	−55.4	+0.6	−64.7	−96.7	−18.9
the $\vec{E}_6 \xrightarrow{W_K} \vec{K}_6$	−87.9	−33.3	−22.6	−31.4	+5.5	+0.6	−4.6	−96.8
verdant $\vec{E}_7 \xrightarrow{W_K} \vec{K}_7$	−41.2	−55.5	−42.3	−59.8	−79.0	−97.9	+3.7	**+93.8**
forest $\vec{E}_8 \xrightarrow{W_K} \vec{K}_8$	−58.9	−75.5	−91.1	−90.6	−75.6	−89.0	−70.8	+4.7

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

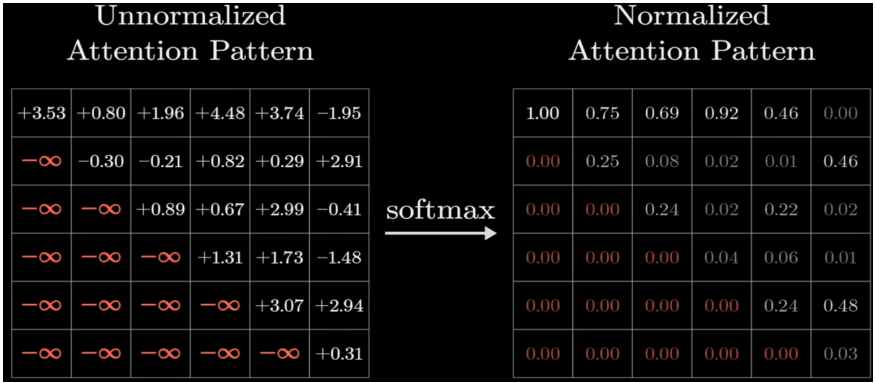
Query & Key Concept (Masked Self-Attention)



Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Final Attention Scores



- Attention $(Q, K, V = \underbrace{softmax(\frac{QK^T}{\sqrt{d_k}})}_{A = Final\ Attention\ Scores,\ N \times M} V)$ → Value Matrix still missing!

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

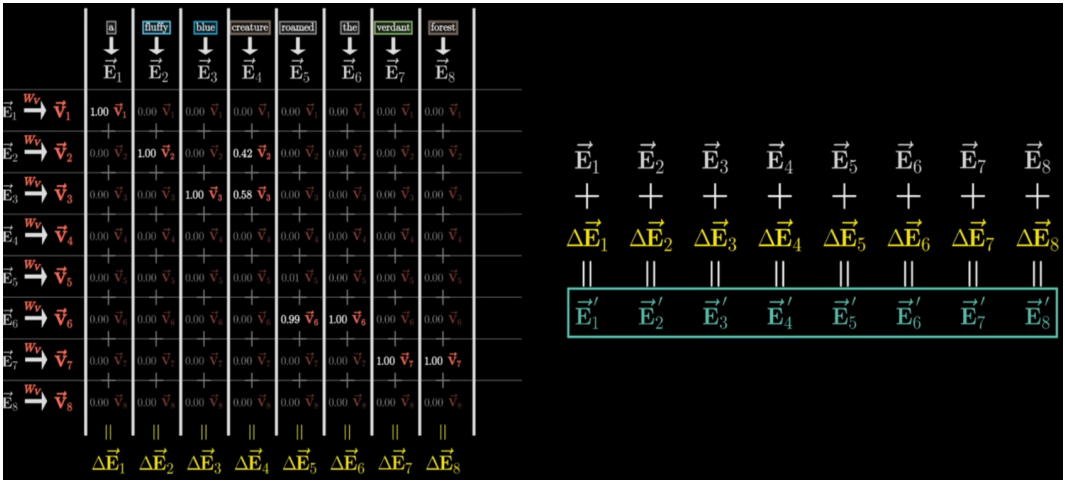
Attention Score & Value Concept

- Value $V = W_V \cdot E$, and $E(D_h \times N)$, with $M =$ Number of Tokens
- Output $O = V \cdot A$, and $A(N \times M)$, with $N =$ Number of Tokens
- Dimensions Parameters: W_V $(D_h \times D_h)$, with $D_h = \frac{D}{H}$, Output O $(D_h \times M)$

Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Attention Score & Value Concept



Source: Images taken from YouTube 3Brown1Blue – https://www.youtube.com/watch?v=eMlx5fFNoYc

Attention Mechanism

Attention in the GPT-3 Model

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$
- Embedding Space Transform: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$; all (B, T, D)

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$
- Embedding Space Transform: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$; all (B, T, D)
- Multi-Head Attention: Embedding Size for each (Self-)Attention Head $D_h = \frac{D}{H}$, resulting in (B, T, D_h) for Q, K, V

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$
- Embedding Space Transform: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$; all (B, T, D)
- Multi-Head Attention: Embedding Size for each (Self-)Attention Head $D_h = \frac{D}{H}$, resulting in (B, T, D_h) for Q, K, V
- Attention Scores: $Q \cdot K^T = A$, with (B, T, T) (Raw Attention/Similarity Scores!)

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$
- Embedding Space Transform: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$; all (B, T, D)
- Multi-Head Attention: Embedding Size for each (Self-)Attention Head $D_h = \frac{D}{H}$, resulting in (B, T, D_h) for Q, K, V
- Attention Scores: $Q \cdot K^T = A$, with (B, T, T) (Raw Attention/Similarity Scores!)
- Attention-Scaled Values: Dividing Similarity Scores in A by $\sqrt{D_h}$ together with applying *softmax*, followed by $O_h = VA$, with (B, T, D_h)

Attention in GPT-3

General Model Setup and Parameter

- Input Self-Attention Layer: (B, T, D), with Batch-Size $(= B)$, Sequence-Length $(= T)$, Embedding Size $(= D)$
- Query, Key, Value Matrices: W_Q, W_K, W_V with $D \times D$
- Embedding Space Transform: $Q = X \cdot W_Q$, $K = X \cdot W_K$, $V = X \cdot W_V$; all (B, T, D)
- Multi-Head Attention: Embedding Size for each (Self-)Attention Head $D_h = \frac{D}{H}$, resulting in (B, T, D_h) for Q, K, V
- Attention Scores: $Q \cdot K^T = A$, with (B, T, T) (Raw Attention/Similarity Scores!)
- Attention-Scaled Values: Dividing Similarity Scores in A by $\sqrt{D_h}$ together with applying *softmax*, followed by $O_h = VA$, with (B, T, D_h)
- Merging/Concatenation Multi-Head Information: Head-Specific Output O_h (B, T, D_h) is combined/concatenated to the overall Output O (B, T, D) and transformed via W_O

Attention in GPT-3

General Model Setup and Parameter

- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$

Attention in GPT-3

General Model Setup and Parameter

- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$

Attention in GPT-3

General Model Setup and Parameter

- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$
- Model Input Shape $X = (1, 2048, 12288)$
- W_Q, W_K, W_V all with Shape $D \times D = 12,288 \times 12,288$
- Q, K, V all with Shape $(1, 2048, 12288)$, and per Head with $(1, 2048, 128)$

Attention in GPT-3

General Model Setup and Parameter

- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$
- Model Input Shape $X = (1, 2048, 12288)$
- W_Q, W_K, W_V all with Shape $D \times D = 12,288 \times 12,288$
- Q, K, V all with Shape $(1, 2048, 12288)$, and per Head with $(1, 2048, 128)$
- Attention Scores A per Head with Shape $(1, 2048, 2048)$ and scaled via $\sqrt{128}$
- Final Output after combining all 96 individual Heads: $O = (1, 2048, 12288)$ followed by a linear projection via W_O with $12,288 \times 12,288$

Attention in GPT-3

General Model Setup and Parameter

- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$
- Model Input Shape $X = (1, 2048, 12288)$
- W_Q, W_K, W_V all with Shape $D \times D = 12,288 \times 12,288$
- Q, K, V all with Shape $(1, 2048, 12288)$, and per Head with $(1, 2048, 128)$
- Attention Scores A per Head with Shape $(1, 2048, 2048)$ and scaled via $\sqrt{128}$
- Final Output after combining all 96 individual Heads: $O = (1, 2048, 12288)$ followed by a linear projection via W_O with $12,288 \times 12,288$
- Parameters per Multi-Head Attention: $12,288 \cdot 128 \cdot 96 \cdot 96 \cdot 4 = 57,982,058,496$

Attention in GPT-3

General Model Setup and Parameter

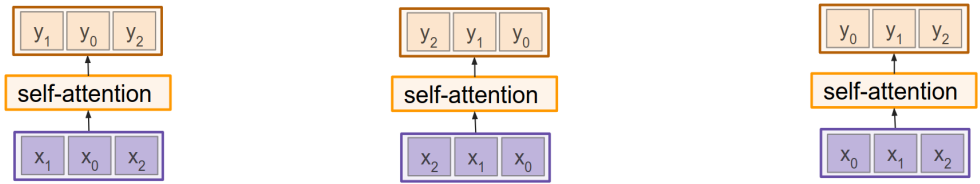
- Batch-Size $B = 1$, Sequence-Length $T = 2,048$
- Embedding Size $D = 12,288$, Number Attention Heads/Layer $H = 96$
- Number of Layer $L = 96$, Embedding Size/Head $D_h = \frac{D}{H} = \frac{12,288}{96} = 128$
- Model Input Shape $X = (1, 2048, 12288)$
- W_Q, W_K, W_V all with Shape $D \times D = 12,288 \times 12,288$
- Q, K, V all with Shape $(1, 2048, 12288)$, and per Head with $(1, 2048, 128)$
- Attention Scores A per Head with Shape $(1, 2048, 2048)$ and scaled via $\sqrt{128}$
- Final Output after combining all 96 individual Heads: $O = (1, 2048, 12288)$ followed by a linear projection via W_O with $12,288 \times 12,288$
- Parameters per Multi-Head Attention: $12,288 \cdot 128 \cdot 96 \cdot 96 \cdot 4 = 57,982,058,496$
- Total number of weights: $175,181,291,520$ (1/3 Attention is All you Need!!!)

12288 * 12288 multiplied by 96 which is number of layers and 4 is 3 weight matrix and one another added at end

Attention Mechanism

What it is about?

Self-Attention Layer – Permutation Equivariant!



- Self-attention does not take into account the sequence of inputs
- However, we want to encode ordered sequences like language or spatially ordered image features as well!
- Keyword: Positional Encoding!

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Positional Encoding

- Idea: Introducing position-specific and additional information to a given input sequence of length m, in order to preserve the original token order → Positional Encodings!
- Positional Encodings can be learned or fixed a priori
- As part of the famous Transformer architecture (Vaswani et al.) a fixed positional scheme was introduced, based on sine and cosine functions

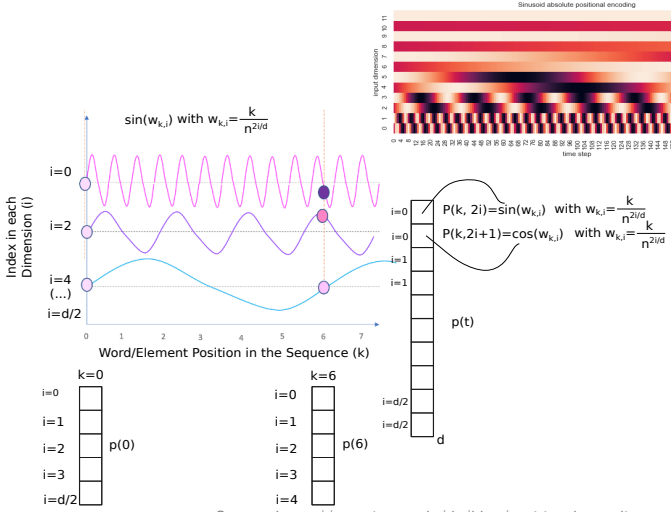
$$p(t) = \begin{bmatrix} sin(\omega_1 \cdot t) \\ cos(\omega_1 \cdot t) \\ sin(\omega_2 \cdot t) \\ cos(\omega_2 \cdot t) \\ \vdots \\ sin(\omega_{d/2} \cdot t) \\ cos(\omega_{d/2} \cdot t) \end{bmatrix}_d \quad with\ \omega_d \cdot t = w_{k,i} = \frac{k}{n^{2i/d}}$$

- d =input dimensionality, i =index of each dimension, k =element position in the sequence, n =user-defined scalar (n =10,000 in the original Transformer Paper)

Attention Mechanism

What it is about?

Positional Encoding



Source: https://www.inovex.de/de/blog/positional-encoding-everything-you-need-to-know/

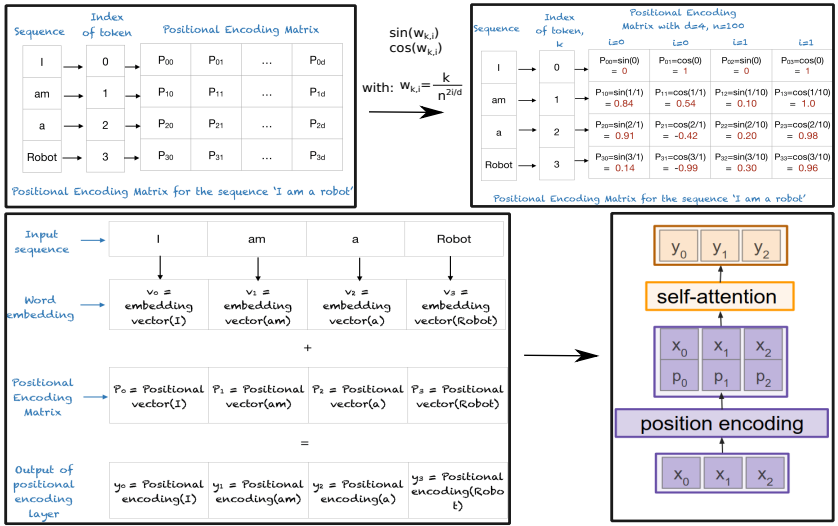
Source: https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model

Source: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Attention Mechanism

What it is about?

Positional Encoding



Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Source: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

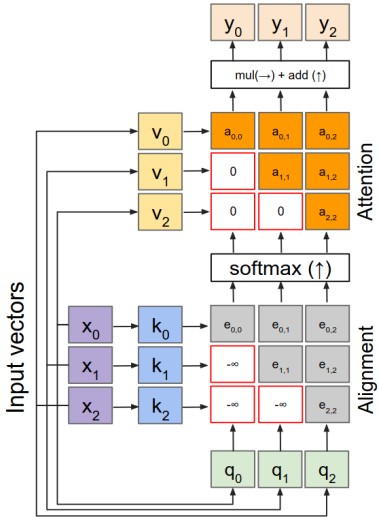
Attention Mechanism

What it is about?

OTH Amberg-Weiden

Masked Self-Attention Layer

- Prevent vectors from looking at future vectors (cheating!)
- Manually set alignment scores to "-infinity", which results in a Softmax-activation $= 0$

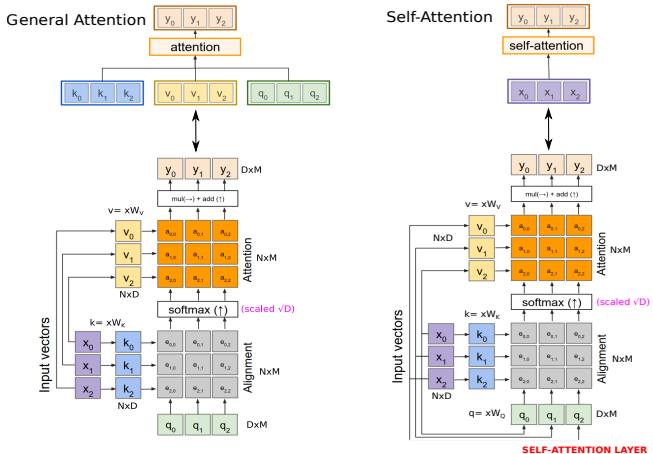


Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

General Attention versus Self-Attention

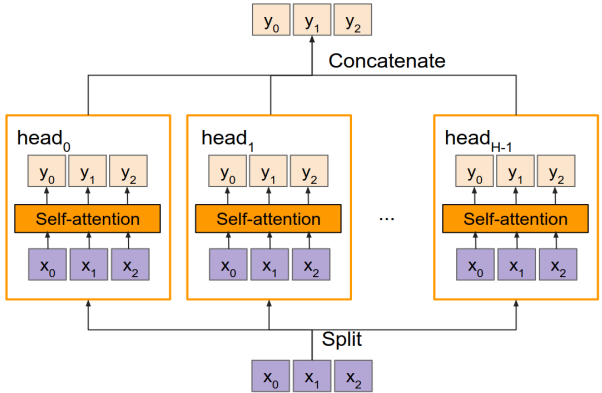


Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

Multi-Head Self-Attention

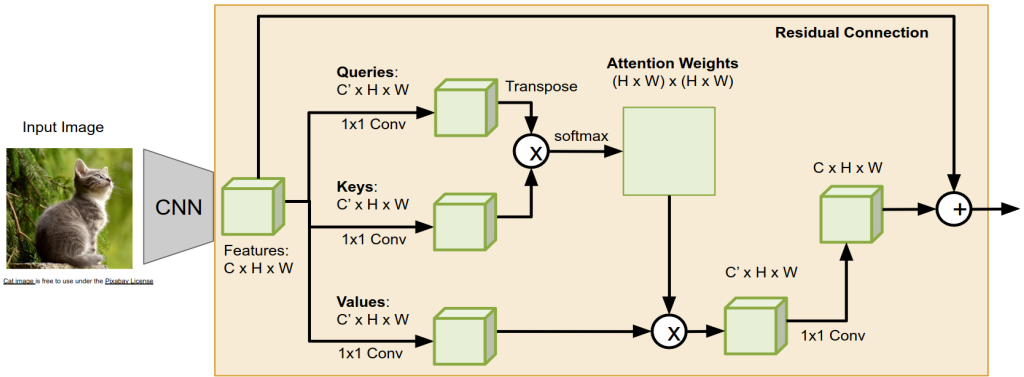


- Multiple self-attention layer/heads, using h (number of heads) different query, key, value triplets in parallel, which are all results of the linear transformation via independent weight matrices
- Computational Efficiency: implementations usually consists of an initial Q, K, V (shape $N \times D$) computation, using W_Q, W_K, W_V with shape $(D \times D)$, and slicing the outputs (Q, K, V) into $D_h = \frac{D}{h}$ large sub-embeddings with shape $(N \times D_h)$ and feed it into a single self-attention block
- Each sub-embedding output of shape $(N \times D_h)$ per head is concatenated across all heads $(N \times D)$, and projected via a fully-connected layer (W_O Output Matrix!)

Attention Mechanism

What it is about?

CNN with Self-Attention



Self-Attention Module

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf
Source: Image (cat) from https://pixabay.com/de/photos/katze-jungtier-k%C3%A4tzchen-graue-katze-2083492/
Source: Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Attention Mechanism

What it is about?

Recurrent Neural Networks

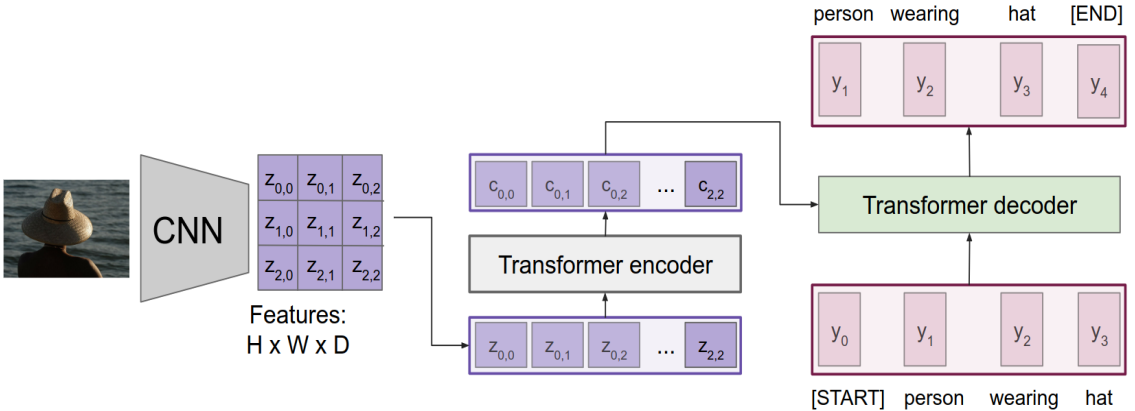
- (+) LSTMs perform comparatively well in terms of long sequences
- (−) Ordered sequences as inputs are required
- (−) Computation in a sequential ordering and subsequent hidden states can only be computed after the the computation of previous hidden states is done

Transformer Networks

- (+) Attentions look at the entire sequence at once, which is the reason why the models perform well on long sequences
- (+) Can be used with an unordered set of inputs, but also ordered sequences, together with positional encodings
- (+) Computation can be parallelized, by computing all alignment and attention scores
- (−) Huge memory consumption, due to $N \times M$-large alignment & attention values, which need to be calculated and stored as part of a single self-attention head $\rightarrow$ However, ongoing hardware improvements!

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Transformer Model for Image Captioning



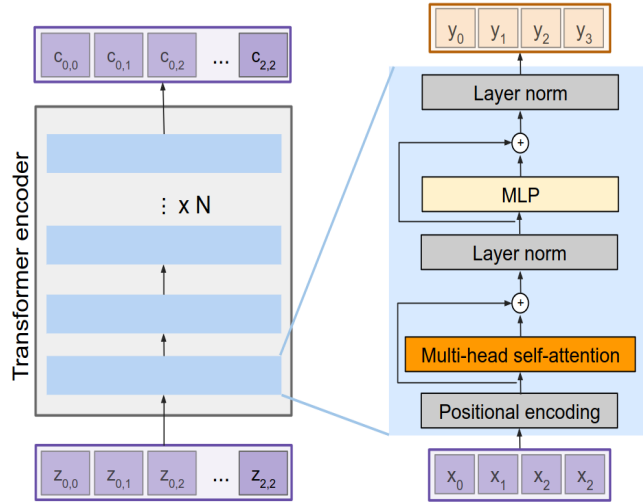
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

The Transformer Encoder Block

- Input/Output: Set of vectors x and y
- Sequentially-ordered process:
  - Positional Encoding (adding)
  - Multi-Head Self-attention
  - Residual Connection
  - Layer Normalization
  - MLP
  - Residual Connection
  - Layer Normalization
- Interaction between vectors only in case of the self-attention block
- All other operations are performed vector-specific



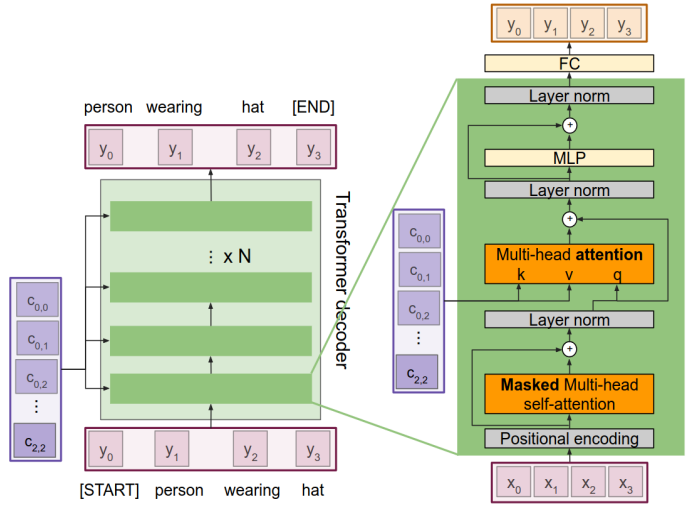
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

The Transformer Decoder Block

- Sequentially-ordered process:
  - Positional Encoding (adding)
  - Masked Multi-Head Self-attention
  - Residual Connection
  - Layer Normalization
  - General Multi-Head Attention
  - Residual Connection
  - Layer Normalization
  - MLP
  - Residual Connection
  - Fully-Connected Layer
- General multi-head attention block attends over the encoder outputs
- Feature injection to the decoder



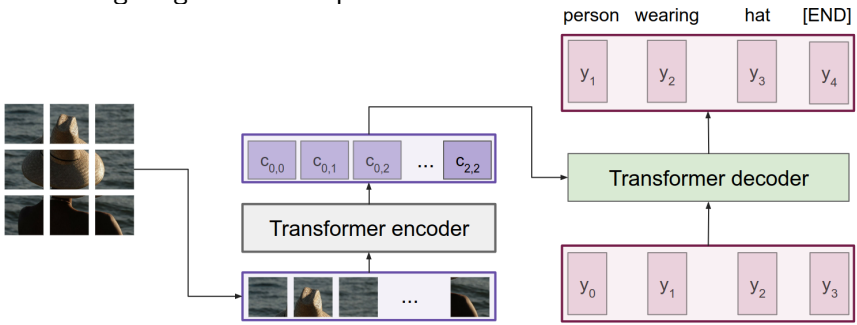
Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

Vision Transformer – Pixel to Text

- Remove the convolutional feature learning part (backbone)
- Slice an image into sub-sections (image regions, smallest is equal to a pixel)
- Feed the individual image regions as a sequence to the encoder



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020

Source: Image from http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf

Attention Mechanism

What it is about?

Summary

- The concept of “attention” is a specific type of layer architecture which learns a given set of weighting factors (attention weights), allowing to focus on relevant input parts
- Query (“to match others”), Key (“to be matched”), Query (“to be weighted averaged”)
  - Query $q_j = W_Q \cdot s_j$
  - Key $k_i = W_K \cdot h_i$, for $i = 1, ..., m$
  - Value $v_i = W_V \cdot h_i$, for $i = 1, ..., m$
- There exist different types of attention: general, self, masked, and multi-head
- The Transformer model is based on the concept of self-attention and uses an encoder-decoder architecture
- The model is highly stable, parallelizable, and improved NLP but also CV tasks
- Transformers are going to replace RNNs (LSTMs, GRUs) and “maybe” also convolutions to a certain extend

Further Questions?

https://www.oth-aw.de/hochschule/ueber-uns/personen/bergler-christian/
https://www.oth-aw.de/hochschule/ueber-uns/personen/levi-patrick/