


Machine Learning for Graphs and Sequential Data

Sequential Data – Neural Network Approaches

lecturer: Prof. Dr. Stephan Günnemann
www.daml.in.tum.de

Summer Term 2023

Data Analytics and
Machine Learning 

Roadmap

- Chapter: Temporal Data / Sequential Data
 1. Autoregressive Models
 2. Markov Chains
 3. Hidden Markov Models
 - 4. Neural Network Approaches**
 - a) **Word Vectors**
 - b) RNNs
 - c) Non-Recurrent Models (ConvNets, Transformer)
 5. Temporal Point Processes

Introduction

- Text is everywhere
- Applying machine learning to textual data to solve machine translation, question answering, sentiment analysis etc.
- Example:

It's a brilliant, honest performance by Nicholson, but the film is an agonizing bore except when the fantastic Kathy Bates turns up.
- Goal: given text predict whether it is positive or negative
- Problem: how to represent words to input them into a subsequent model
- One solution: one-hot encoding
 - High dimensional
 - Too sparse
 - Assumes the words are **independent** of each other

Introduction

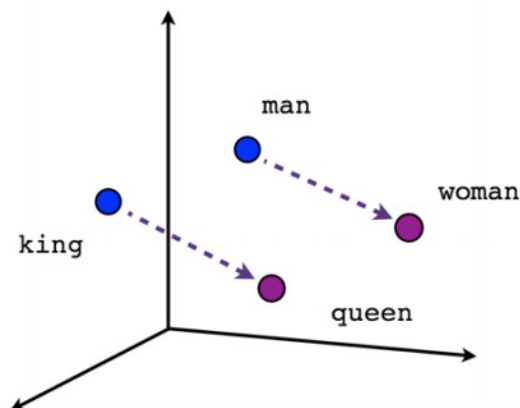
- Words as vectors while keeping the underlying language properties
- E.g. similar words should have vectors near each other
- **Distributional hypothesis** – words that appear in similar contexts have similar meanings

You shall know a word by the company it keeps.

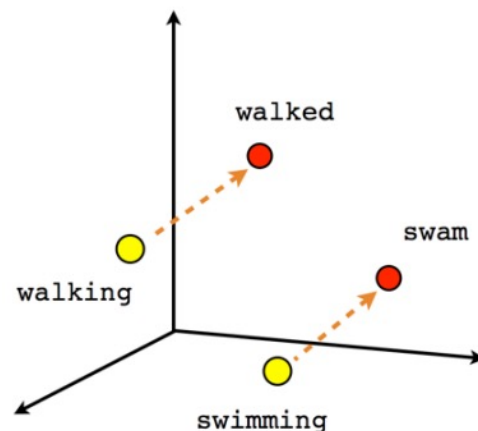
J. R. Firth

- Example: *hotel* and *motel*
 - Can be used interchangeably in many sentences while remaining meaningful
- However: *duck* – an animal vs. *duck* – to lower head quickly

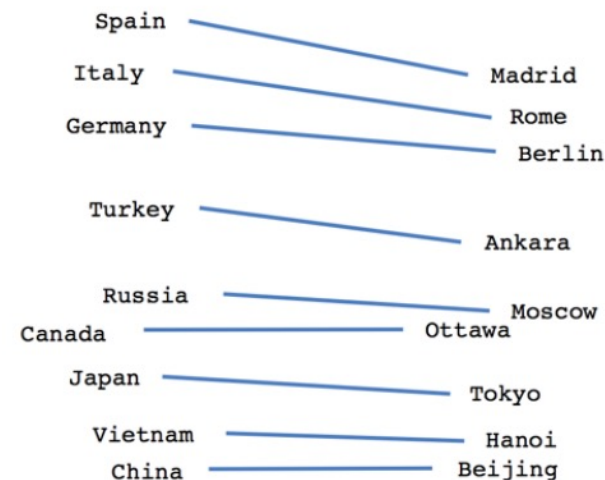
Introduction



Male-Female



Verb tense



Country-Capital

Illustration of how vectors can represent linguistic concepts

Figure from <https://www.tensorflow.org/tutorials/representation/word2vec>

Co-occurrence Matrix

- To be aware of **context** we can simply count how many times each word appeared beside other words
- If the text is given with words $\{x_1, \dots, x_N\}$, then a window of size l around a word x_i is $\{x_{i-l}, \dots, x_{i-1}, x_{i+1}, \dots, x_{i+l}\}$
- We slide this window over sentences and count the co-occurrences
- Example:

I like dogs. I like cats too. They hate each other.

- For the first sentence the windows ($l = 1$) are:
 - (\emptyset, like) (I, dogs) $(\text{like}, .)$ (dogs, \emptyset)

Co-occurrence Matrix

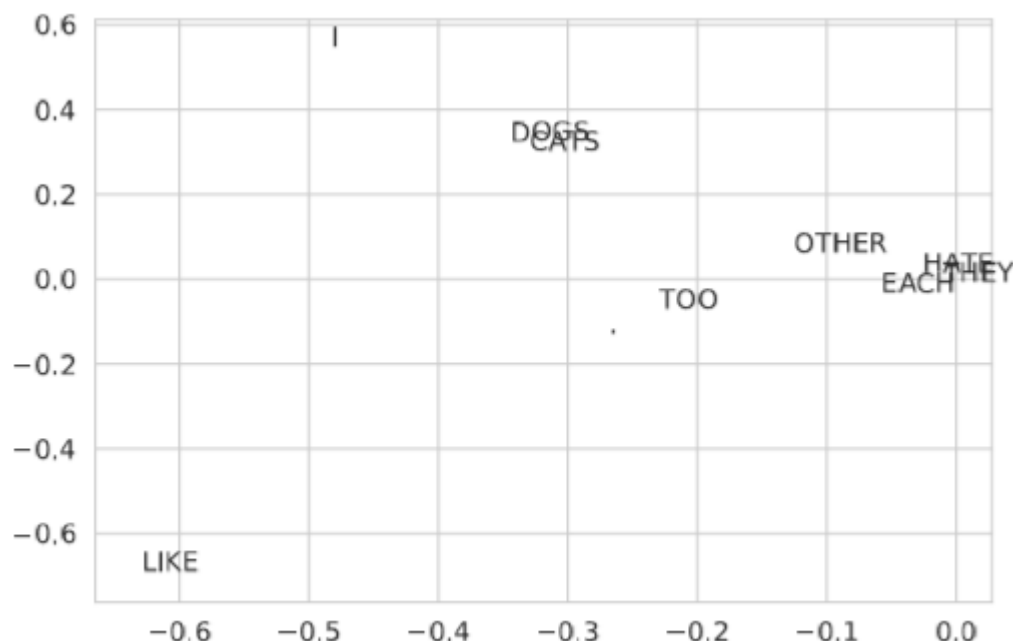
- After counting all the pairs we get a co-occurrence matrix M :

	.	I	cats	dogs	each	hate	like	other	they	too
.	0	0	0	1	0	0	0	1	0	1
I	0	0	0	0	0	0	2	0	0	0
cats	0	0	0	0	0	0	1	0	0	1
dogs	1	0	0	0	0	0	1	0	0	0
each	0	0	0	0	0	1	0	1	0	0
hate	0	0	0	0	1	0	0	0	1	0
like	0	2	1	1	0	0	0	0	0	0
other	1	0	0	0	1	0	0	0	0	0
they	0	0	0	0	0	1	0	0	0	0
too	1	0	1	0	0	0	0	0	0	0

- Pros: similar words have similar vectors
- Cons: still high dimensional and sparse
- Solution: reduce the dimension to get dense vector of fixed dimension

SVD

- We can reduce the dimension with an SVD decomposition: $M = U\Sigma V^T$
- If we take the first D columns of U we get D -dimensional word vectors
- Applied to the previous example ($D = 2$):



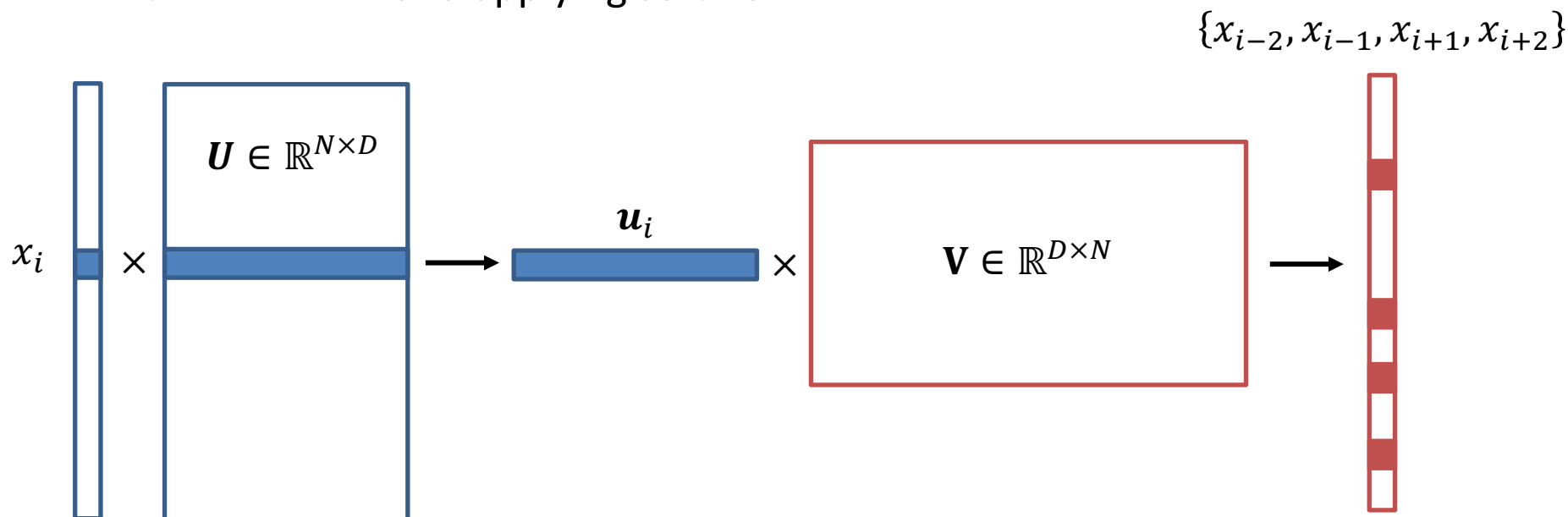
- Problems: slow computation and hard to add new words

Word2Vec

- A different way to get word vectors is with a neural network
- Task: prediction of words based on context
- Two approaches:
 - **Continuous bag-of-words (CBOW)**
 - Predicts a word from the words surrounding it (window)
 - Not good for rare words because the model might not predict them from context
 - **Skip-gram [1]**
 - Predicts the surrounding context from the current word
 - Given a rare word it must *understand* it to predict the context
 - Slower to train but can work well with smaller amounts of data and with rare words

Skip-gram

- Input: one-hot vector with dimension N
- Embedding: project the word to D -dimensional space with $\mathbf{U} \in \mathbb{R}^{N \times D}$
 - Since input has zeros everywhere except on i th position, multiplication is equivalent to taking i th row of \mathbf{U}
- Prediction: get probabilities of context words by multiplying embedding with $\mathbf{V} \in \mathbb{R}^{D \times N}$ and applying softmax



Skip-gram

- Formally: if $S = \{x_{i-l}, \dots, x_{i-1}, x_{i+1}, \dots, x_{i+l}\}$ is a window of size l around the word x_i , and $\boldsymbol{\theta}$ denotes model parameters, the objective is

$$\max_{\boldsymbol{\theta}} \mathbb{E}[P(S|x_i, \boldsymbol{\theta})] = \min_{\boldsymbol{\theta}} (-\mathbb{E}[P(S|x_i, \boldsymbol{\theta})])$$

$$\text{where } P(S|x_i, \boldsymbol{\theta}) = \prod_{x_k \in S} P(x_k|x_i, \boldsymbol{\theta})$$

$$\text{and } P(x_k|x_i, \boldsymbol{\theta}) = \text{softmax}(\mathbf{u}_i \mathbf{V})_k$$

- The vector \mathbf{u}_i is the corresponding embedding
- We can choose to set $\mathbf{U} = \mathbf{V}$, giving less parameters to optimize but also less expressiveness

Training

- Each forward pass computes normalized probabilities over the entire vocabulary

$$P(x_k|x_i, \boldsymbol{\theta}) = \text{softmax}(\mathbf{u}_i \mathbf{V})_k = \exp(\mathbf{u}_i \mathbf{v}_k^T) / \sum_{l=1}^N \exp(\mathbf{u}_i \mathbf{v}_l^T)$$

- Inefficient for large vocabularies
- Alternative: **Negative Sampling** [3]:
 - In each iteration, sample word p in the context of word i and word(s) n not in this context
 - Binary classification problem: Distinguish positive pair (i, p) from the negative pair(s) (i, n)

$$L = \log(P(x_p|x_i, \boldsymbol{\theta})) + \log(1 - P(x_n|x_i, \boldsymbol{\theta}))$$

$$P(x_k|x_i, \boldsymbol{\theta}) = \text{sigmoid}(\mathbf{u}_i \mathbf{v}_k^T)$$

References

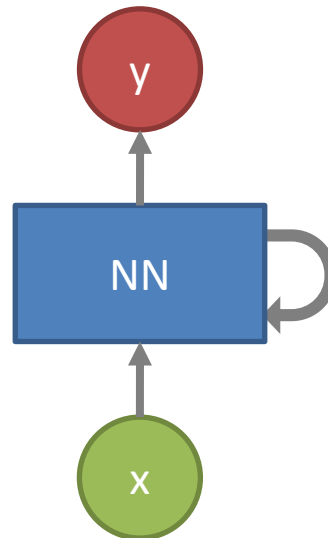
- [1] Mikolov, Tomas et al. (2013). “Efficient estimation of word representations in vector space”. In: arXiv preprint arXiv:1301.3781.
- [2] Morin, Frederic and Yoshua Bengio (2005). “Hierarchical probabilistic neural network language model.” In: Aistats. Vol. 5. Citeseer, pp. 246–252.
- [3] Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: Advances in Neural Information Processing Systems.

Roadmap

- Chapter: Temporal Data / Sequential Data
 1. Autoregressive Models
 2. Markov Chains
 3. Hidden Markov Models
 - 4. Neural Network Approaches**
 - a) Word Vectors
 - b) RNNs**
 - c) Non-Recurrent Models (ConvNets, Transformer)
 5. Temporal Point Processes

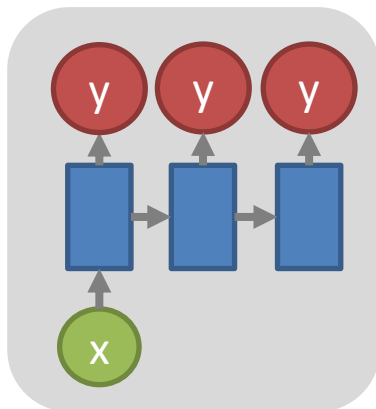
Introduction

- In word embeddings, we learn a representation for every individual word
- How to process an entire sequence with neural networks?
 - In particular if the sequences have varying length?
- We can use **Recurrent Neural Networks (RNNs)**



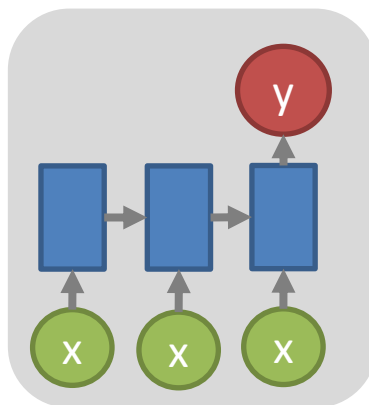
RNN Tasks

- Different problems can be solved using RNNs:



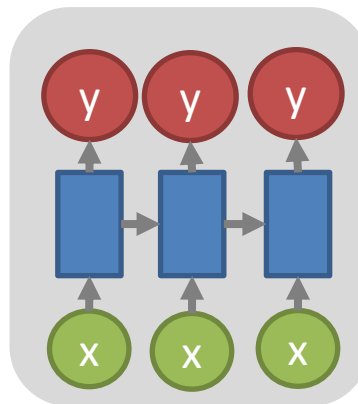
One to Many:

- Image Captioning



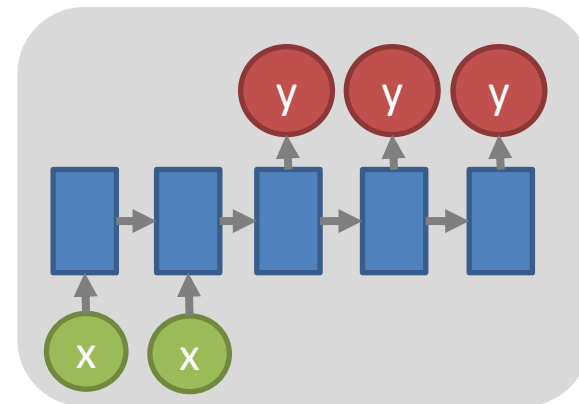
Many to One:

- Sentiment Analysis
- Text Classification



Many to Many

- Machine Translation
- Video Captioning
- Part of Speech Tagging



Definition

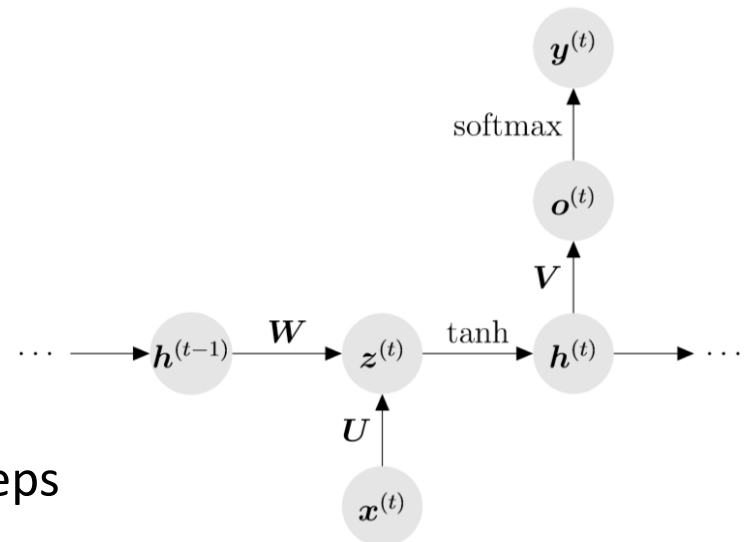
- Given a sequence of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ and outputs $\{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(N)}\}$ we want to know the probability $P(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)})$
- Represent a sequence $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t-1)}\}$ with a hidden state $\mathbf{h}^{(t-1)}$
- Neural network takes $\mathbf{h}^{(t-1)}$ and current input and maps them to a new hidden state $\mathbf{h}^{(t)}$ from which we can predict the output at step t
 - Also use $\mathbf{h}^{(t)}$ in the next step
- The update equations are

$$\mathbf{z}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{z}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)}$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$



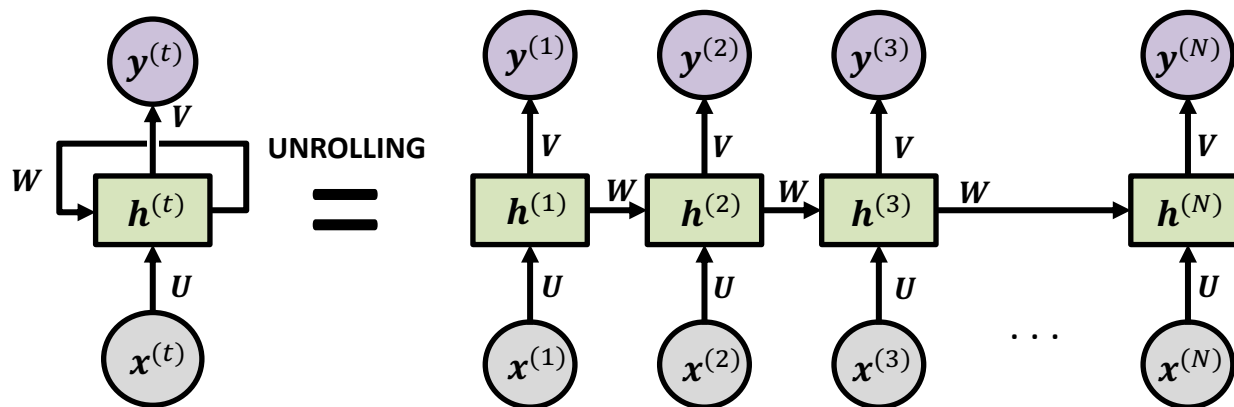
- The weights are shared over all time steps

Objective

- The negative log-likelihood is

$$\begin{aligned}
 L &= -\log \prod_t p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}) \\
 &= -\sum_t \log p_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}) = -\sum_t L^{(t)}
 \end{aligned}$$

- Network fully differentiable – train with a gradient based method
- Unrolling of the RNN graph



Backpropagation through time

- All functions used in the update equations are differentiable (linear, tanh, softmax) → We can compute the derivative w.r.t the parameters:

$$\frac{\partial L}{\partial \mathbf{V}} = \sum_t (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) (\mathbf{h}^{(t)})^T$$

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_t \text{diag} \left(1 - (\mathbf{h}^{(t)})^2 \right) \frac{\partial L}{\partial \mathbf{h}^{(t)}} (\mathbf{h}^{(t-1)})^T$$

$$\frac{\partial L}{\partial \mathbf{U}} = \sum_t \text{diag} \left(1 - (\mathbf{h}^{(t)})^2 \right) \frac{\partial L}{\partial \mathbf{h}^{(t)}} (\mathbf{x}^{(t)})^T$$

$$\mathbf{z}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{z}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)}$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}) \text{ or } \mathbf{o}^{(t)}$$

- Since parameters are shared over the steps, final derivative are a sum of all the contributions at every step t .

Backpropagation through time

- The hidden state $\mathbf{h}^{(t)}$ recursively depends on all previous hidden states $\mathbf{h}^{(t-1)}, \dots, \mathbf{h}^{(0)}$ i.e.

$$\mathbf{h}^{(t)} = \tanh(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b})$$

- The gradient $\frac{\partial L}{\partial \mathbf{h}^{(t)}}$ depends on future times

$$\frac{\partial L}{\partial \mathbf{h}^{(t)}} = \mathbf{v}^T (\hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) + \mathbf{W}^T \text{diag} \left(1 - (\mathbf{h}^{(t+1)})^2 \right) \frac{\partial L}{\partial \mathbf{h}^{(t+1)}}$$

- The impact of future times might **vanish** or **explode** (e.g. 1-D example: $W > 1$ or $W < 1$) → RNN cannot retain information for many steps.

$$\frac{\partial L}{\partial \mathbf{h}^{(t)}} = \sum_{s=t}^N \frac{\partial L}{\partial \mathbf{h}^{(s)}} \frac{\partial \mathbf{h}^{(s)}}{\partial \mathbf{h}^{(t)}} = \sum_{s=t}^N \frac{\partial L}{\partial \mathbf{h}^{(s)}} \prod_{t+1 \leq k \leq s} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{h}^{(k-1)}} = \sum_{s=t}^N \frac{\partial L}{\partial \mathbf{h}^{(s)}} \prod_{t \leq k \leq s} W (1 - (h^{(k)})^2)$$

GRU

- Solution: change the RNN architecture so it can keep information longer
- Main idea: not every input should be fully taken into account when updating the hidden state – update partially with a *gating* mechanism
- **Gated Recurrent Unit (GRU) [2]**

$$\mathbf{z}^{(t)} = \sigma(\mathbf{W}_z[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}])$$

$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}])$$

Gates

$$\tilde{\mathbf{h}}^{(t)} = \tanh\left(\mathbf{W}[\mathbf{r}^{(t)} \odot \mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}]\right)$$

Simple RNN update – gives candidate state

$$\mathbf{h}^{(t)} = (1 - \mathbf{z}^{(t)}) \odot \mathbf{h}^{(t-1)} + \mathbf{z}^{(t)} \odot \tilde{\mathbf{h}}^{(t)}$$

How much to take from previous state vs. candidate state

LSTM

- More powerful architecture: **Long Short-Term Memory (LSTM)** [3]
- Introduces a cell state $\mathbf{c}^{(t)}$ in addition to $\mathbf{h}^{(t)}$ – we have two states

Previous
cell state

$$\mathbf{f}^{(t)} = \sigma \left(\mathbf{W}_f \left[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)} \right] \right)$$

Forget gate

$$\mathbf{i}^{(t)} = \sigma \left(\mathbf{W}_i \left[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)} \right] \right)$$

Input gate

$$\mathbf{o}^{(t)} = \sigma \left(\mathbf{W}_o \left[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)} \right] \right)$$

Output gate

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tanh \left(\mathbf{W} \left[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)} \right] \right)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)})$$

Simple RNN update
– LSTM treats it as
an input

Update hidden state (now the output)
using a cell state

Summary

- LSTM and GRU are two examples of improvements to the basic RNN
- Gating enables *skipping* some inputs to capture long-term dependencies
 - Actually, since it uses an element-wise product, it can *remember* or *forget* per individual dimension of a hidden state
 - Avoids gradient problems that RNN has
- It is fully differentiable so we can derive gradients for all the parameters as in the RNN and train it with, e.g., gradient descent
- Many variations on LSTM architecture
 - E.g. *peephole LSTM* – replaces $\mathbf{h}^{(t)}$ with $\mathbf{c}^{(t)}$ in all the equations

References

- [1] Cho, Kyunghyun et al. (2014). “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: arXiv preprint arXiv:1406.1078.
- [2] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). “On the difficulty of training recurrent neural networks”. In: International conference on machine learning, pp. 1310–1318.
- [3] Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: Neural computation 9.8, pp. 1735–1780.
- [4] Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." *arXiv preprint arXiv:1802.05365* (2018).

Roadmap

- Chapter: Temporal Data / Sequential Data
 1. Autoregressive Models
 2. Markov Chains
 3. Hidden Markov Models
 - 4. Neural Network Approaches**
 - a) Word Vectors
 - b) RNNs
 - c) Non-Recurrent Models (ConvNets, Transformer)**
 5. Temporal Point Processes

Introduction

- Sometimes when modeling a sequence we do not need the complete history to produce the output
- Example: **generating speech**
 - Raw audio has many data points (16000 per second)
 - Important relations on many time scales

- Recall an autoregressive model:

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t$$

- Uses a fixed window of p previous inputs and performs regression
- Can we use neural networks to capture more complex behavior?
 - RNNs share the parameters across time steps, but **depend on full history**
 - We can instead use **Convolutional Neural Networks (ConvNets)**

Recap: Definition

- The convolution $f * g$ of functions f and g is

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau$$

- In image processing, given an image I and a kernel K , both 2-D matrices, the convolution can be written as:

$$(K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

- Output is again a 2-D matrix (transformed image), where an element (pixel) is a sum of its neighbors, weighted by a kernel

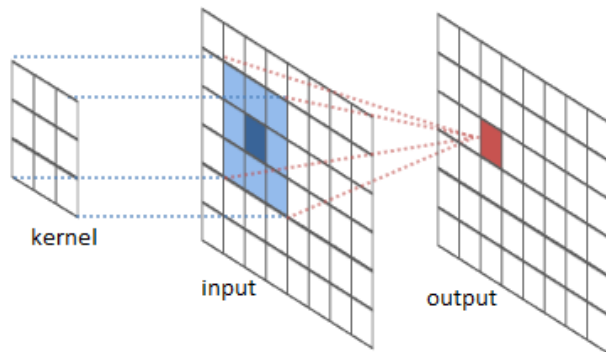
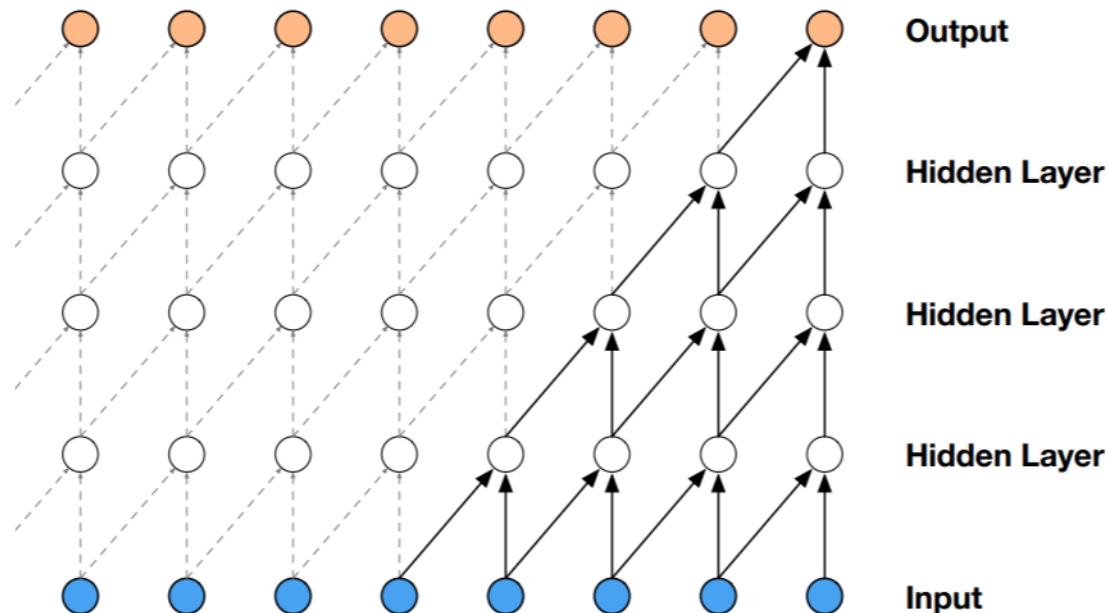


Figure from
<https://intellabs.github.io/RiverTrail/tutorial/>

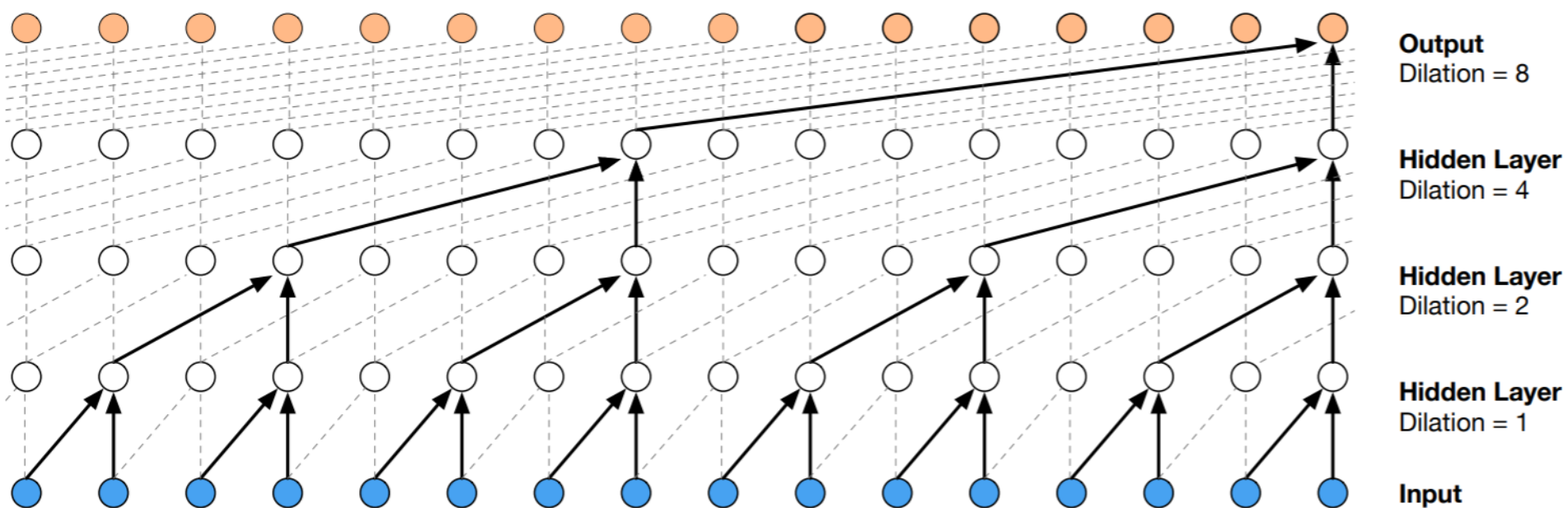
WaveNet

- Sequences are 1-D so we can use a 1-D version of ConvNets
- WaveNet** [2] is an architecture that uses 1-D ConvNets to model speech
 - In addition, it uses special convolutions to ensure causality and increase receptive field
- Causal convolutions** – ensure that the output only depends on the past



WaveNet

- **Dilated convolutions** – skip some inputs to increase the receptive field
 - Dilation of 1 gives standard convolution
 - If we start with dilation of 1 in the first layer and double it with every layer (2,4,8...) the receptive field will be the exponential of the number of layers
 - E.g. with 4 layers we use 16 inputs in the first layer



Transformers & Attention

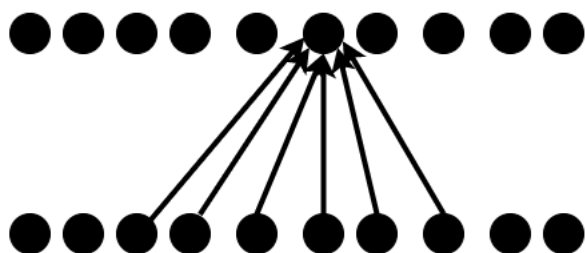
Transformers:

- Transformers [3] are fast models using attention mechanisms
 - Like WaveNet, it is not a recurrent neural network → parallelizable and fast

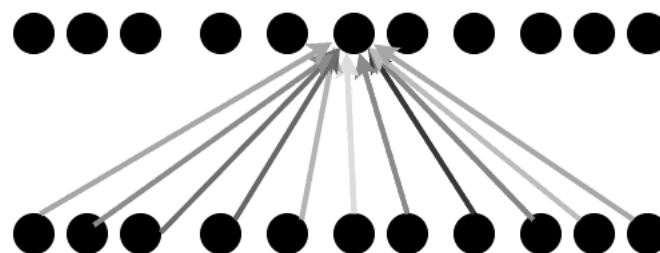
Attention:

- Attention is a learned weighting over the elements x_j (given element x_i)

Convolution

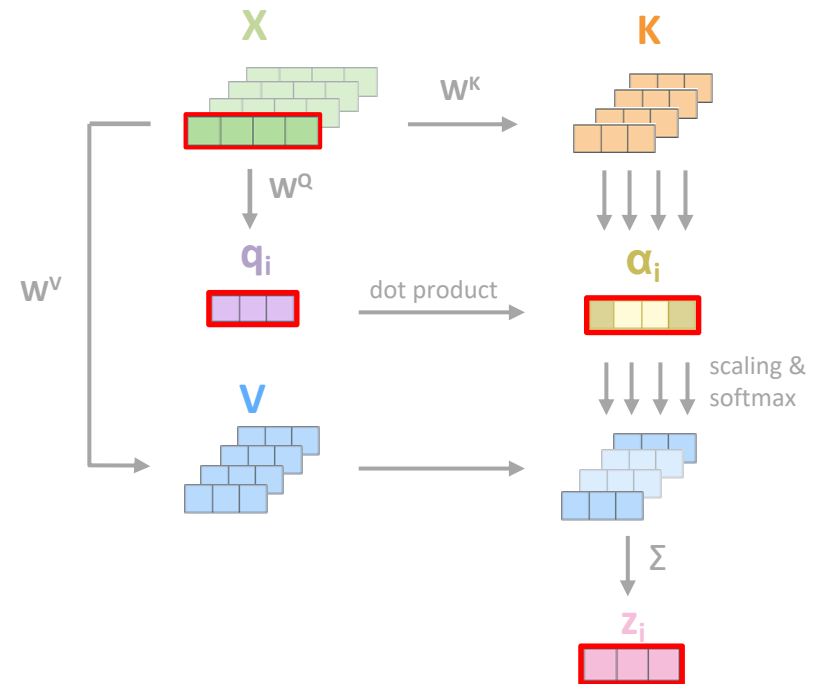


Attention



(Self-)Attention

- Weighting mechanism:
 - The weighting is computed by applying softmax to query/key scores
 - Query depends on x_i ; key on x_j
 - The weight indicates how much of v_j we use (the “value” of x_j)
- Self-attention: the attention is on the input signal itself
- It is easy computable in a matrix formulation.



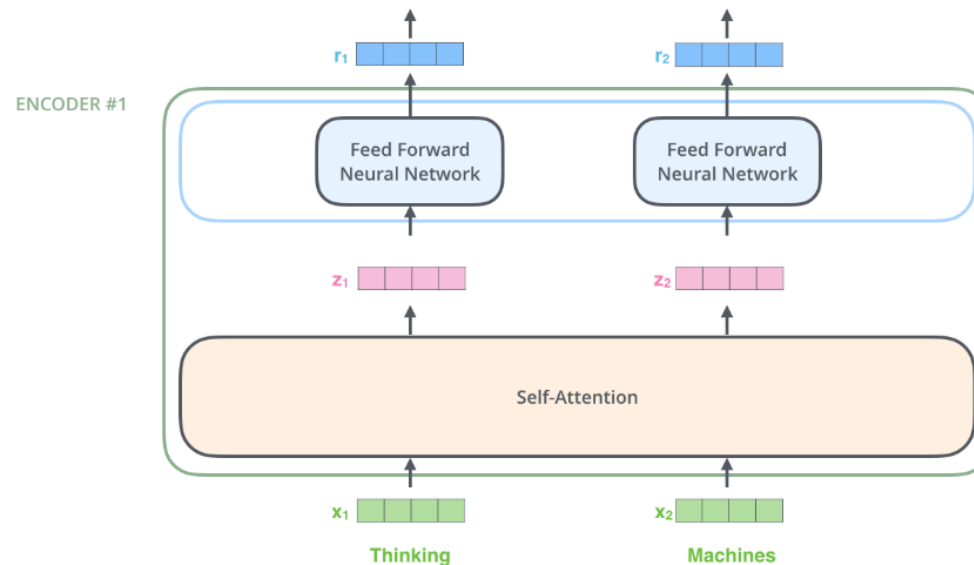
$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

$$= Z$$

“Self-attention allows the model to look at other positions in the input sequence for clues that can help lead to a better encoding for this word” [4]

Encoder Block

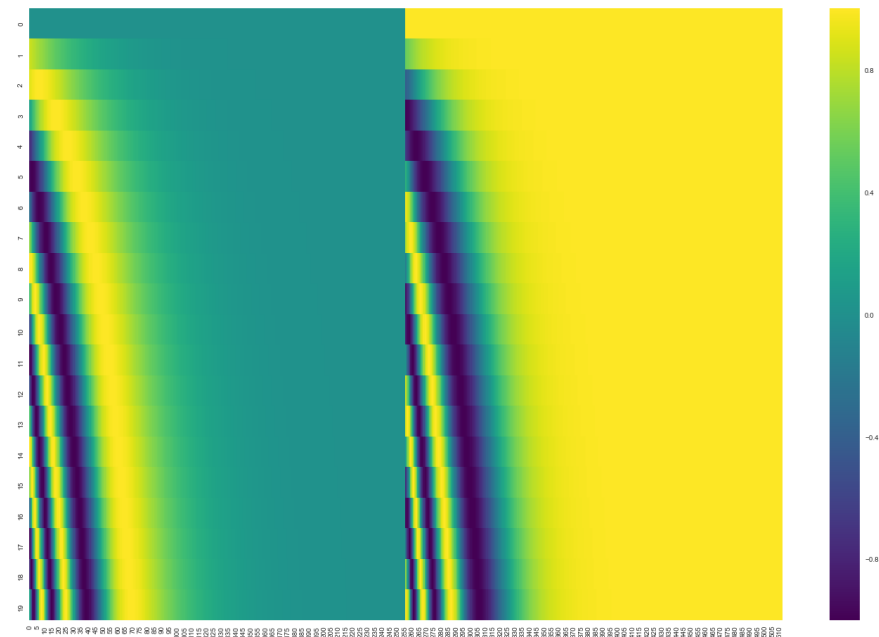
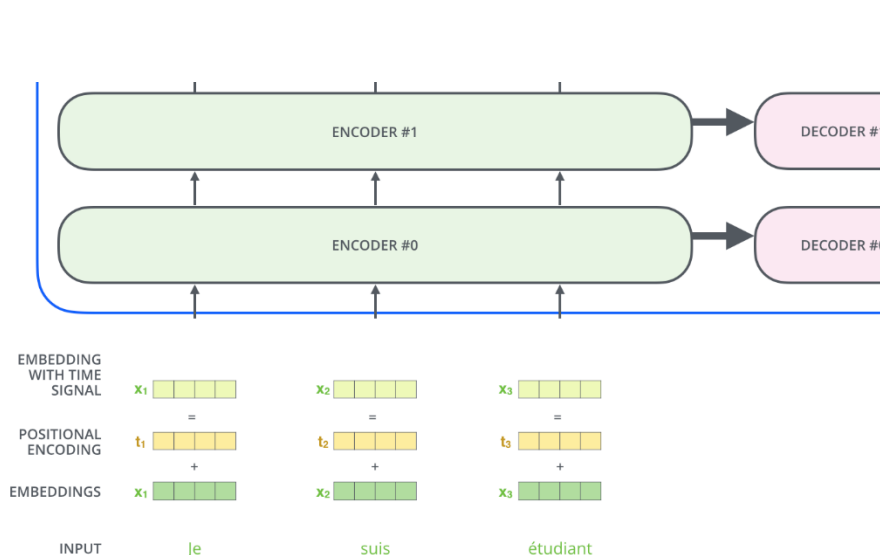
- Tokens (e.g. words) are represented with **embeddings**
- The self-attention layer “couples” the embeddings
- The rest handles the embeddings independently



The following images are taken from [4] Jalammar blog

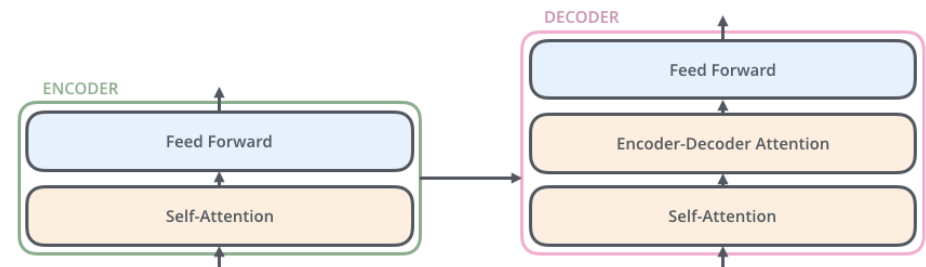
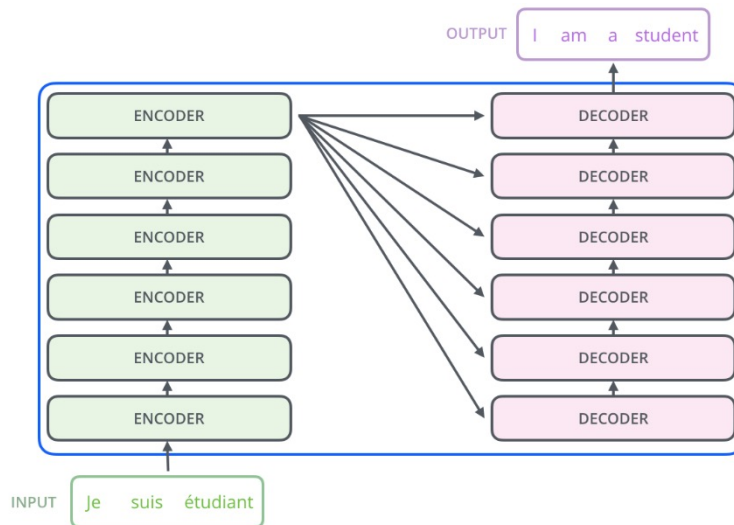
Positional Encoding

- **Note:** Attention mechanisms do not care about the order of tokens / words, i.e. the architecture itself is not aware of the non-i.i.d. nature
- Standard workaround: **positional encoding** to represent the token order. Positional encoding: meaningful static vectors which are concatenated with the token embeddings.



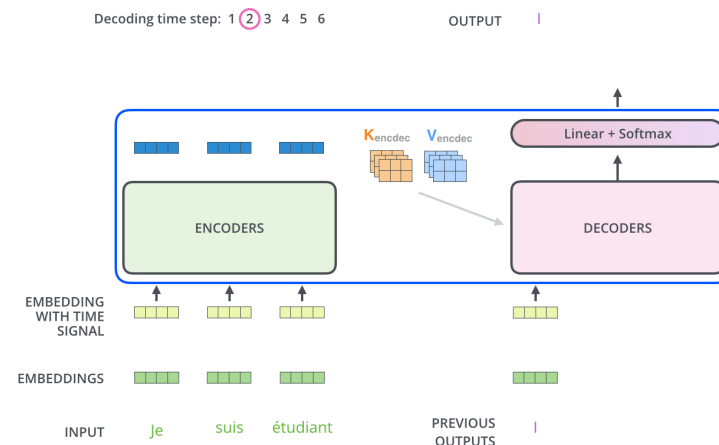
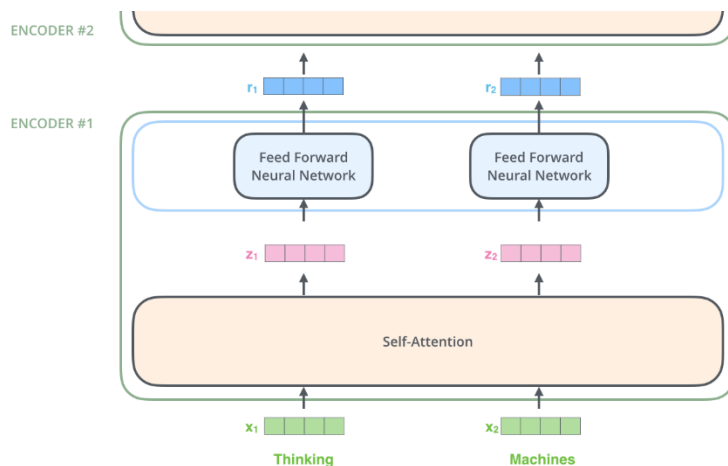
Transformers

- Transformers are composed of a stack of encoders and decoders using (self-)attention



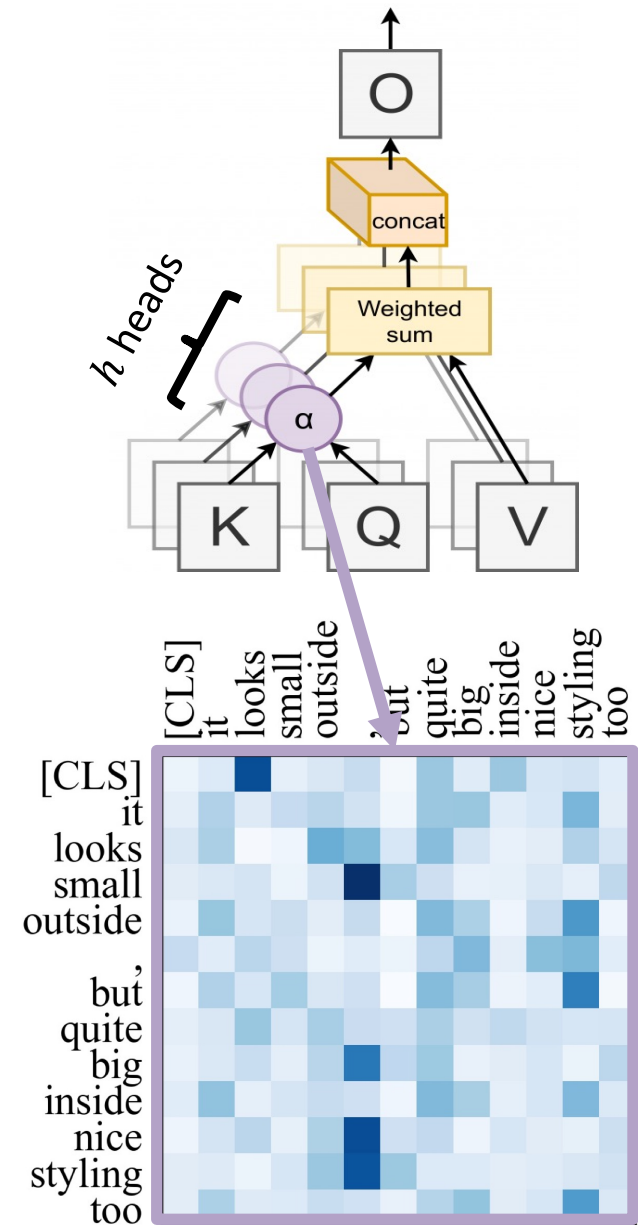
Transformers: Notes on Training and Inference

- During training, the embeddings flow all through the transformer at the same time/in parallel (attention coefficients are masked for future tokens)
 - This enables efficient training on very large datasets; crucial for the success of recent models
- At inference time, decoding is done one step after the other until the end of sentence symbol is reached.



Transformers: Complexity

- Multi-Head Attention: Combine the output of h self-attention blocks
- Each self-attention block computes n^2 attention weights
- Intractable for long sequences
- Many possible solutions:
 - Fix structure of attention weights
 - Low-Rank approximations
 - Downsampling the sequence length n



Images taken from [5, 6]

Transformers: GPT Models

- **Generative Pre-Trained Transformers:**
 - First, unsupervised pre-training on predicting the next token in a sequence

$$L = \sum_i \log P_{\theta}(\mathbf{x}^{(i)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(i-k)})$$

- Second, task-specific fine-tuning (classification, chatbots, ...)
- GPT-n models use large text corpora (Data crawled from the internet, books, ...)
- Strong performance because of large models and datasets: Loss and datasets compute follow a power-law
 - e.g. GPT-3 has **~175B** parameters

OpenAI codebase next word prediction

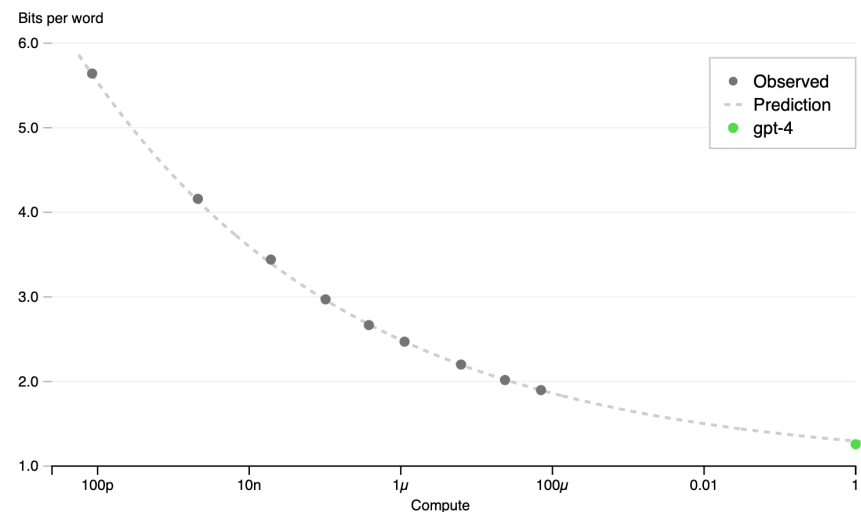


Image taken from [7]

Questions – NN

1. In an RNN, the hidden state at a given time influences all hidden states into the future. However, an RNN cannot model long-term dependencies. Why?
2. What is the receptive field of a causal convolution and dilated convolution with n layers ?

References

- [1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in NIPS*, pp. 1097-1105. 2012.
- [2] Van Den Oord, Aäron, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. "WaveNet: A generative model for raw audio." *SSW* 125 (2016).
- [3] Ashish Vaswani et al., "Attention Is All You Need" NIPS 2017
- [4] Jalammar blog, <http://jalammar.github.io/illustrated-transformer/>
- [5] Cui, Baiyun, et. al. "Fine-tune BERT with sparse self-attention mechanism." *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 2019.
- [6] Paul Michel, <https://blog.ml.cmu.edu/2020/03/20/are-sixteen-heads-really-better-than-one/>
- [7] OpenAI. GPT-4 Technical Report. arXiv preprint