NLP and the Web – WS 2024/2025

Lecture 4
Information Retrieval II



Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

Syllabus (tentative)

Nr.	Lecture
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	**IR – Word Representation, Transformer/BERT**
05	IR – Re-Ranking Methods
06	IR – Language Domain Shifts, Dense / Sparse Retrieval
07	LLM – Language Modeling Foundations
08	LLM – Neural LLM, Tokenization
09	LLM – Transformers, Self-Attention
10	LLM – Adaption, LoRa, Prompting
11	LLM – Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	LLM – Scaling, Computation Cost
14	Review & Preparation for the Exam

Today

IR – Word Representation / Neural IR

1 **Word Embeddings**
- Byte-Pair-Encoding

2 Simple Neural Techniques
- Convolutional NN
- Recurrent NN
- Encoder-Decoder Architecture

3 Transformer Architecture
- BERT Pre-Training

Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

Disclaimer

This lecture focusses on the application of these neural network architectures, not the fine details

We will only give an overview of most techniques

We will not cover mathematics of Deep Learning

If you want to go deeper:
Visit “DeepLearning in NLP” in the next summer semester

Going Neural: Word Representations

- Gradient descent based neural networks operate only in continuous spaces: Tensors of floating point numbers and continuous transformation functions
  - Without continuous values and functions -> no gradient
- This means we can't just input the character-values of words in a linear algebra "network" and expect it to work
- We need to map chars or word-pieces or words to some sort of vector
  - A lot of options have been developed to do that – we'll look at some of them

If you haven't, follow https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Simplest Option: One-Hot Encoding

- One Vector dimension for each word in our vocabulary
- Very sparse: Each word vector has exactly one “1”, rest is “0”

	cat	mat	on	sat	the
the =>	0	0	0	0	1
cat =>	1	0	0	0	0
sat =>	0	0	0	1	0
…			…		

Better: Word Embeddings

- Provide a dense vector representation for words
  - Typically 100-300 dimension
  - The dimensions are abstract
  - The vector space allows for math operations
    - For example nearest neighbors: semantic related words are close together in the space
- Can be unsupervised pre-trained on huge text data sets
  - Wikipedia, CommonCrawl, or more domain specific
  - And fine-tuned inside a model (end-to-end trained)
- Super simple data structure: `Dictionary<string, float[]>`
  - A major factor for their success: ease of use

Word Embeddings

- There are many unsupervised (creation) methods
  - 1-Word-1Vector:
    - *Word2Vec (Skip-Gram & CBOW)*
    - *Glove*
    - *A lot of specialized variants of the two*
  - *1-Word-1-Vector+Char-n-grams*
    - *FastText (based on Word2Vec)*
  - *Contextualized / context dependent / complex structure (char or word piece based)*
    - *ELMo*
    - *Transformers a la BERT and its variants*

Recommended Library for simple word embeddings: https://github.com/RaRe-Technologies/gensim

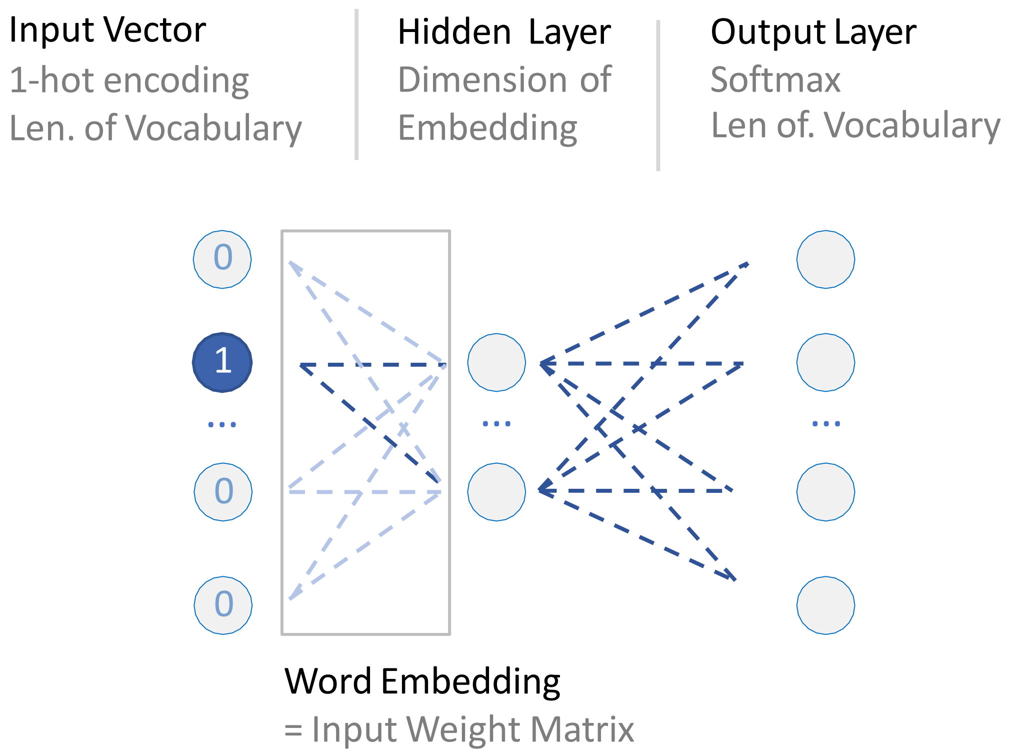
Unsupervised Training: Language Modelling

- We don’t have explicit labels = unsupervised
- But we have real text, how people use language – we model that
- Task: Predict next word given a sequence of words
  - Allows us to compute loss based on probability over a vocabulary
- Main technique for text pre-training
- Many variants exist:
  - Predict context words -2,-1,+1,+2 ..
  - Predict masked words in sequence (Masked Language model)

More about Masked Language Modelling in the Transformers lecture

Word2Vec

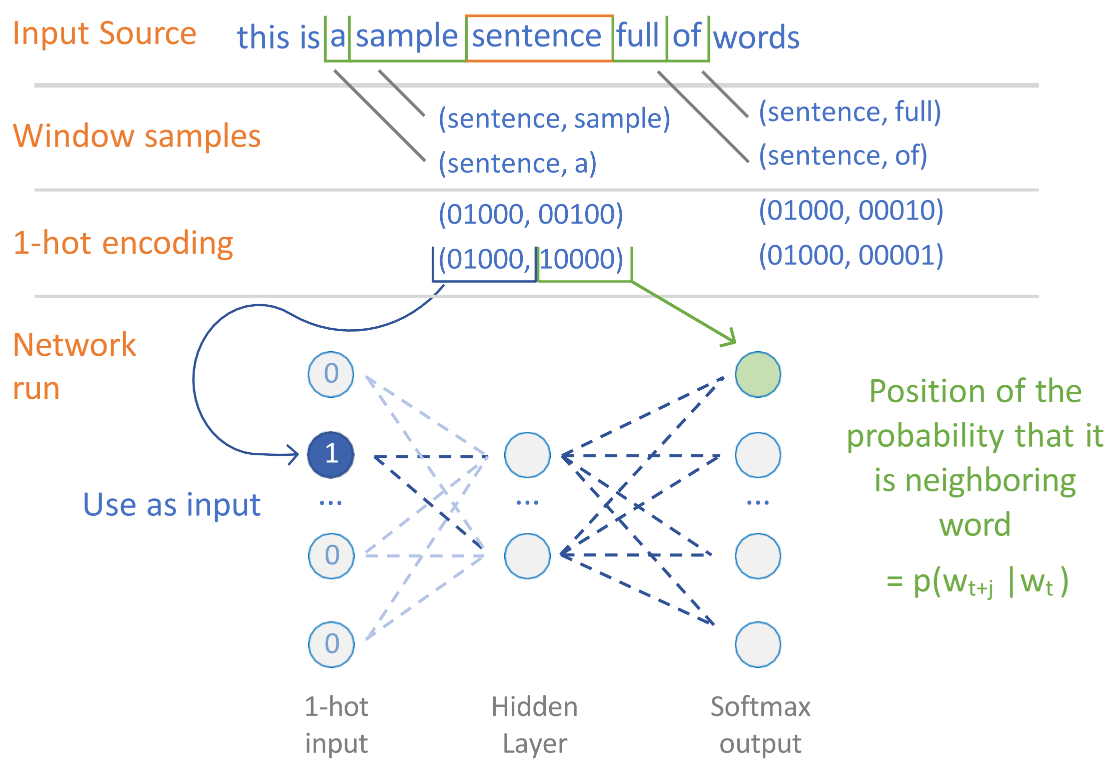
- Train a 1 hidden layer network to predict context words
  - Language Modelling
- Target words via 1-hot encoding
- Harvest the word vectors from the network = take the matrix
  - Each row is now corresponding to the 1-hot position of a word
- Output matrix is ignored (mostly)



Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Proc. Of NeurIPS. 2013.

Word2Vec - Training

- Train with a sliding window across our input text
  - That text sausage does not know about sentence or document boundaries (it does not matter)
- Compute negative log likelihood loss
  - But not over all terms in the vocabulary (too costly)
  - Do negative sampling of random terms



For detailed formulations & derivatives: Stanford NLP course https://www.youtube.com/watch?v=ERibwqs9p38

Limitations: Word Ordering or N-Grams

“it was not good, it was actually quite bad”

== or !=

“it was not bad, it was actually quite good”

- The ordering & local context is important: “not good” vs. “not bad”
- Looking at N words at a time is called N-gram
- Creating bi-gram (2) or tri-gram (3) embeddings is not feasible
  - Sparsity problem
  - Not enough training data: no connection between “quite good” and “very good”

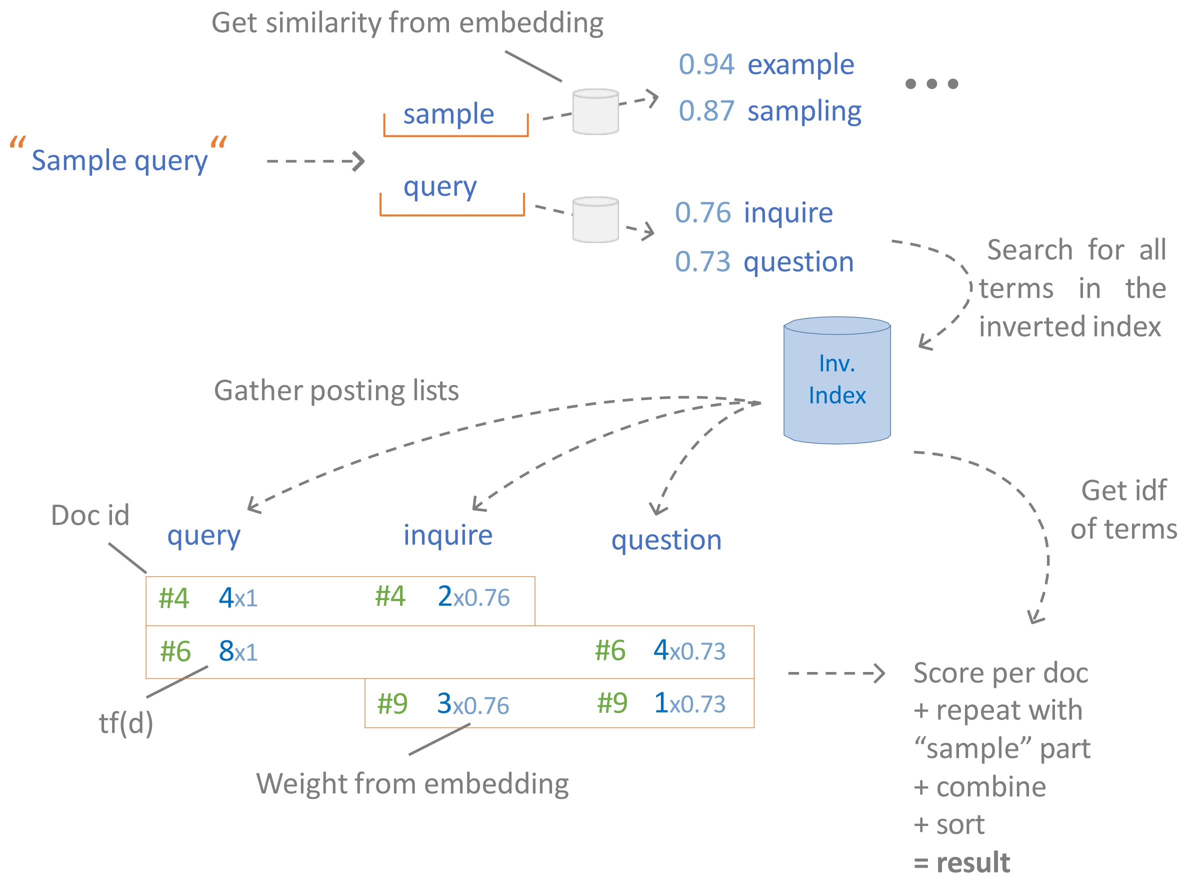
Limitations: Multiple Senses per Word

- Word2Vec, Glove & FastText always map 1 word to 1 vector
  - This is good for analysis purposes and constrained resource environments
- There is no contextualization in the vector after training
  - The vector of 1 word does not change based on other words in the sentence around it
  - Words with multiple senses depending on context are squashed together in an average or most common sense in the training data
- But many words do have many senses based on the context
  - Missed opportunity for improved effectiveness

Query Expansion with Word Embeddings

- Expand the search space of a search query with similar words
- Update collection statistics
- Adapted relevance model to score 1 document with multiple similar words together

N. Rekabsaz, M. Lupu, A. Hanbury, and G. Zuccon, "Generalizing Translation Models in the Probabilistic Relevance Framework," CIKM 2016
https://dl.acm.org/citation.cfm?id=2983833



Today

IR – Word Representation / Neural IR

1. Word Embeddings
   - **Byte-Pair-Encoding**
2. Simple Neural Techniques
   - Convolutional NN
   - Recurrent NN
   - Encoder-Decoder Architecture
3. Transformer Architecture
   - BERT Pre-Training

How to deal with unknown words?

We do not use words as our tokens, but we use sub-words!

-> Basic algorithm: Byte-Pair Encoding

Byte Pair Encoding (BPE)

- Subword tokenization technique
- Used for data compression and dealing with unknown words

- Initialization:
- Vocabulary = set of all individual characters
- V = {A, B, C, … a, b, c, … 1, 2, 3, … !, $, %, …}

- Repeat:
- - Choose two symbols that appear as a pair most frequently (say “a” and “t”)
- - Add new merged symbol (“at”)
- - Replace each occurrence with the new symbol (“t”,”h”,”a”,”t” -> “t”,”h”,”at”)

- Until k merges have been done

Byte Pair Encoding (BPE)

```
Segments:                                    Vocabulary:
5       l o w </w>                           </w>, d, e, I, l, n, o, r, s, t, w
2       l o w e s t </w>
6       n e w e r </w>
3       w i d e r </w>
2       n e w </w>

Most frequent symbol pair: er (9 times)

Segments:                                    Vocabulary:
5       l o w </w>                           </w>, d, e, I, l, n, o, r, s, t, w, er
2       l o w e s t </w>
6       n e w er </w>
3       w i d er </w>
2       n e w </w>
```

Byte Pair Encoding (BPE)

Segments:

5 l o w </w>
2 l o w e s t </w>
6 n e w er </w>
3 w i d er </w>
2 n e w </w>

Vocabulary:

</w>, d, e, I, l, n, o, r, s, t, w

Most frequent symbol pair: er</w> (9 times)

Segments:

5 l o w </w>
2 l o w e s t </w>
6 n e w er</w>
3 w i d er</w>
2 n e w </w>

Vocabulary:

</w>, d, e, I, l, n, o, r, s, t, w, er, er</w>

Byte Pair Encoding (BPE)

Segments:
5 l o w </w>
2 l o w e s t </w>
6 n e w er</w>
3 w i d er</w>
2 n e w </w>

Vocabulary:
</w>, d, e, I, l, n, o, r, s, t, w, er, er</w>

Most frequent symbol pair: ne (8 times)

Segments:
5 l o w </w>
2 l o w e s t </w>
6 ne w er</w>
3 w i d er</w>
2 ne w </w>

Vocabulary:
</w>, d, e, I, l, n, o, r, s, t, w, er, er</w>, ne

Byte Pair Encoding (BPE)

Main pros:

Compression efficiency – common sequences are treated as single tokens
Adaptability – Encoding can optimize for different types of text
Flexibility – Can handle out-of-vocabulary text (can still use basic chars)

Summary: Word Embeddings

1. Represent words as vectors instead of characters
2. Many potential applications in IR, such as query expansion
3. Sub-word embeddings to deal with Out-Of-Vocabulary terms

Today

IR – Word Representation / Neural IR

1. Word Embeddings
   - Byte-Pair-Encoding
2. **Simple Neural Techniques**
   - **Convolutional NN**
   - Recurrent NN
   - Encoder-Decoder Architecture
3. Transformer Architecture
   - BERT Pre-Training

Representation Learning: Word N-Grams

“it was not good, it was actually quite bad”

== or !=

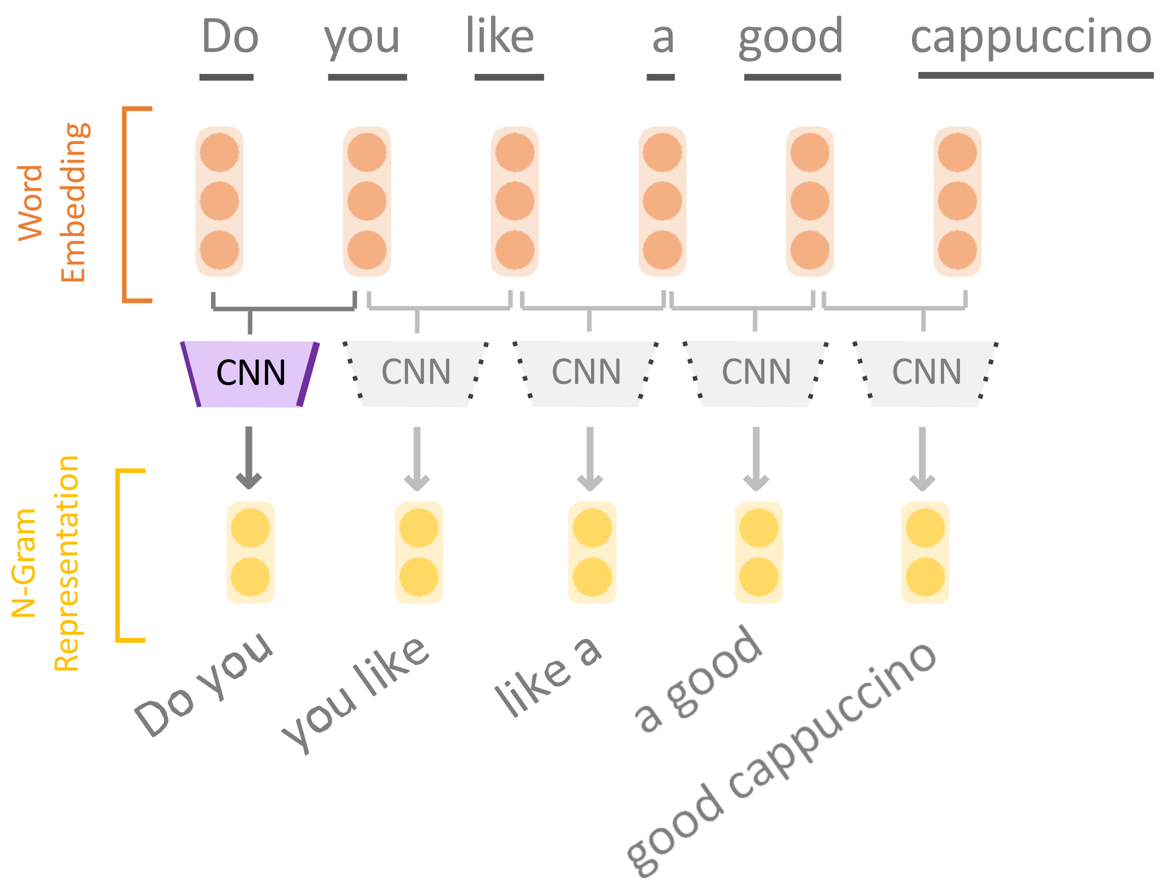
“it was not bad, it was actually quite good”

- The ordering & local context is important: “not good” vs. “not bad”
- Looking at N words at a time is called N-gram
- Creating bi-gram (2) or tri-gram (3) embeddings is not feasible
  - Sparsity problem
  - Not enough training data: no connection between “quite good” and “very good”

1D CNN

- 2D CNNs are ubiquitous in computer vision
- What are CNNs doing?
  - Applying a filter with a sliding window over the input data
  - Values in the filter region are merged into an output value -> guided by the filter
  - The filter parameters are learned during the training
  - Typically multiple filters are applied in parallel (made for GPU multiprocessing)
- 1D CNNs have a 1 dimensional filter and operate on 1 dimensional input

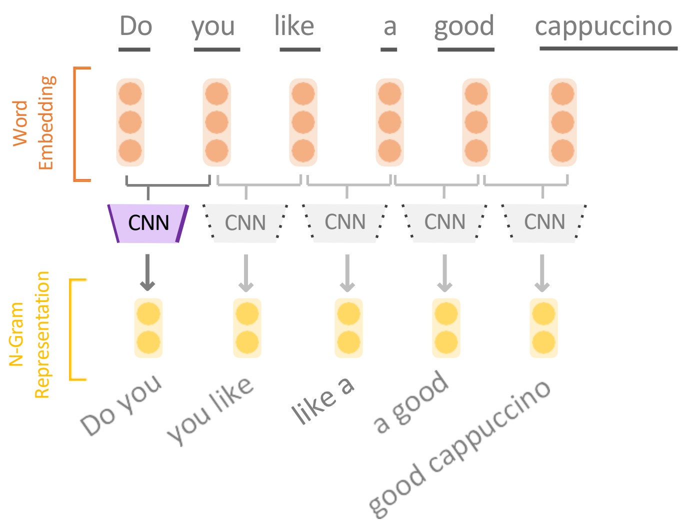
Modelling Word N-Grams with 1D CNNs



- Apply a 1D CNN on a sequence of word vectors
- N of N-grams = filter size
  - In this example N=2
- Output is a sequence of N-gram representations
  - Further used in other network components
- WE & CNN can be trained end-to-end

Kalchbrenner, N., Grefenstette, E. and Blunsom, P., 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*

1D CNNs in PyTorch



- Definition:

```
conv = nn.Sequential(
          nn.ConstantPad1d((0,2 - 1), 0),
          nn.Conv1d(kernel_size=2,
                    in_channels=300,
                    out_channels=200),
          nn.ReLU())
```

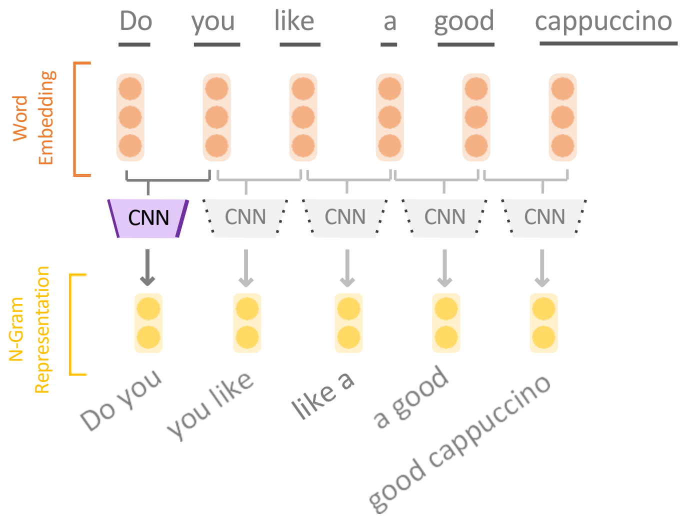
PyTorch Helper
Pad for same out dim.
The size of the window
Word embedding dim.
N-gram output dim
Activation function

- Forward:

```
embeddings_t = embeddings.transpose(1, 2)
n_grams = conv(embeddings_t).transpose(1, 2)
```

Documentation: https://pytorch.org/docs/stable/nn.html#convolution-layers

1D CNNs in PyTorch



- Forward:

Embeddings shape:
[batch, sequence_length, emb_dim]

```
embeddings_t = embeddings.transpose(1, 2)
n_grams = conv(embeddings_t).transpose(1, 2)
```

Transpose to nn.Conv1d required shape

nn.Conv1d requires tensor in shape:
[batch, emb_dim, sequence_length]

Optional transpose back to original layout:
[batch, sequence_length, conv1d_out_channels]

Documentation: https://pytorch.org/docs/stable/nn.html#convolution-layers

Today

IR – Word Representation / Neural IR

1. Word Embeddings
   - Byte-Pair-Encoding
2. **Simple Neural Techniques**
   - Convolutional NN
   - **Recurrent NN**
   - Encoder-Decoder Architecture
3. Transformer Architecture
   - BERT Pre-Training

Recurrent Neural Networks

- Model global patterns in sequence data
  - Allow for arbitrarily sized inputs
- Trainable with backpropagation & gradient descent
  - Recursion gets unrolled to form a standard computation graph
- Ability to condition the next output on an entire sentence history
  - Allows for conditional generation models
- Of course not only useable with text data – but now it is our focus

Neural Network Methods in NLP, Chapter 14 (source of notations presented here)

Simple RNN

- The simplest RNN, which takes ordering of elements into account is:

$$s_i = R_{SRNN}(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$$

- s_i is the state of the RNN at position *i*
  - And it depends on the previous state s_{i-1} (recursive)
- W^s, W^x , b are trainable parameters

Elman, J.L., 1990. Finding structure in time. Cognitive science

s_i	RNN state / output at *i*
x_i	Input vector at *i* (from $x_{1:n}$)
b	Bias vector
W^s, W^x	Weight matrices
g	Nonlinear activation function (tanh, relu)

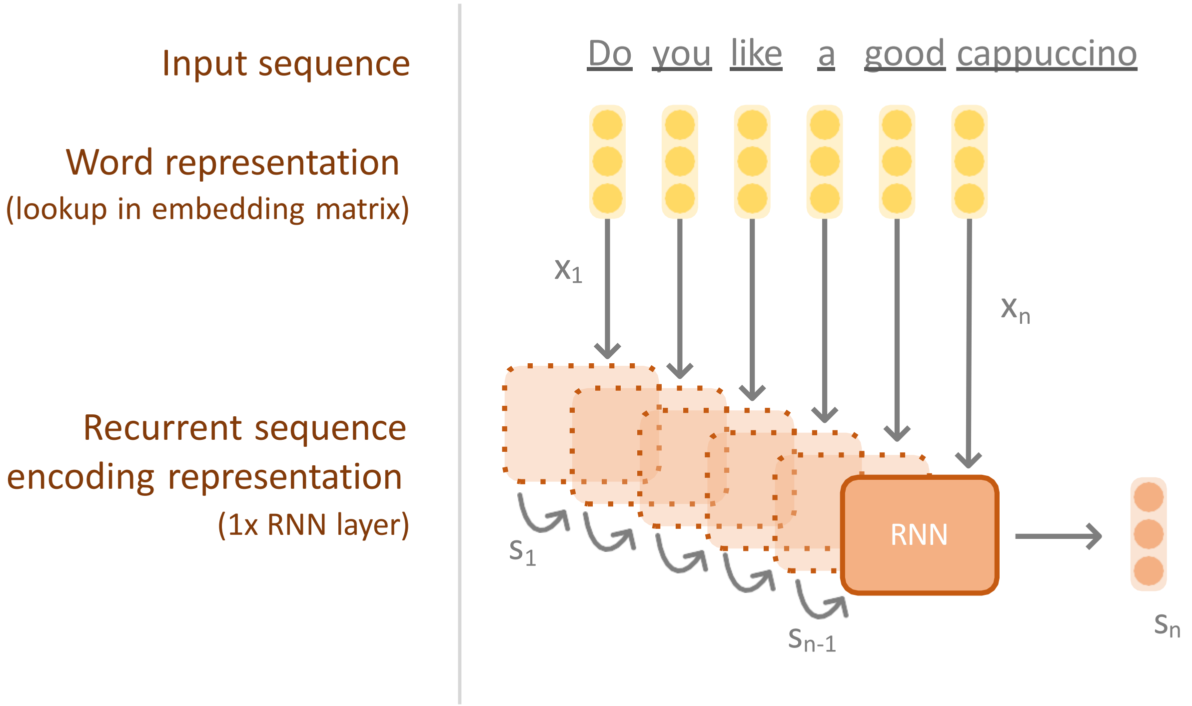
$s_i, b \in \mathbb{R}^{d_s}$

$x_i \in \mathbb{R}^{d_x}$

$W^s \in \mathbb{R}^{d_s \times d_s}$

$W^x \in \mathbb{R}^{d_x \times d_s}$

RNN as Encoder



- Sequence as the input single vector (last state) as output
  - Part of a larger network
- Unrolled computation graph visualized by shifted overlapping dotted boxes
  - The RNN is still one set of learnable parameters
- Optimally, s_n represents the full sequence

Sequence In + Out Tasks

- Translation
- Question Answering
  - SQuAD - https://rajpurkar.github.io/SQuAD-explorer/
- Summarization
- Email auto response
- Chatbots describing how much they like cappuccino 

- Solutions for the tasks influence each other

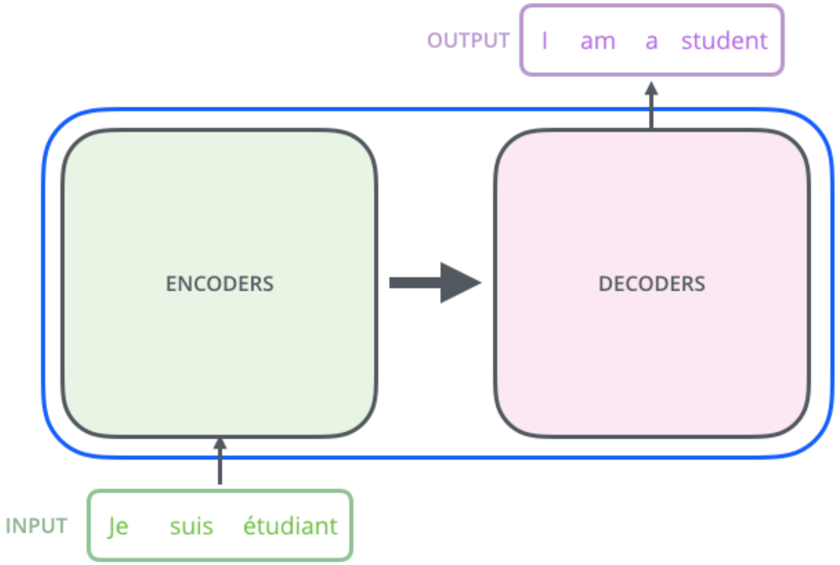
Today

IR – Word Representation / Neural IR

1. Word Embeddings
   - Byte-Pair-Encoding
2. **Simple Neural Techniques**
   - Convolutional NN
   - Recurrent NN
   - **Encoder-Decoder Architecture**
3. Transformer Architecture
   - BERT Pre-Training

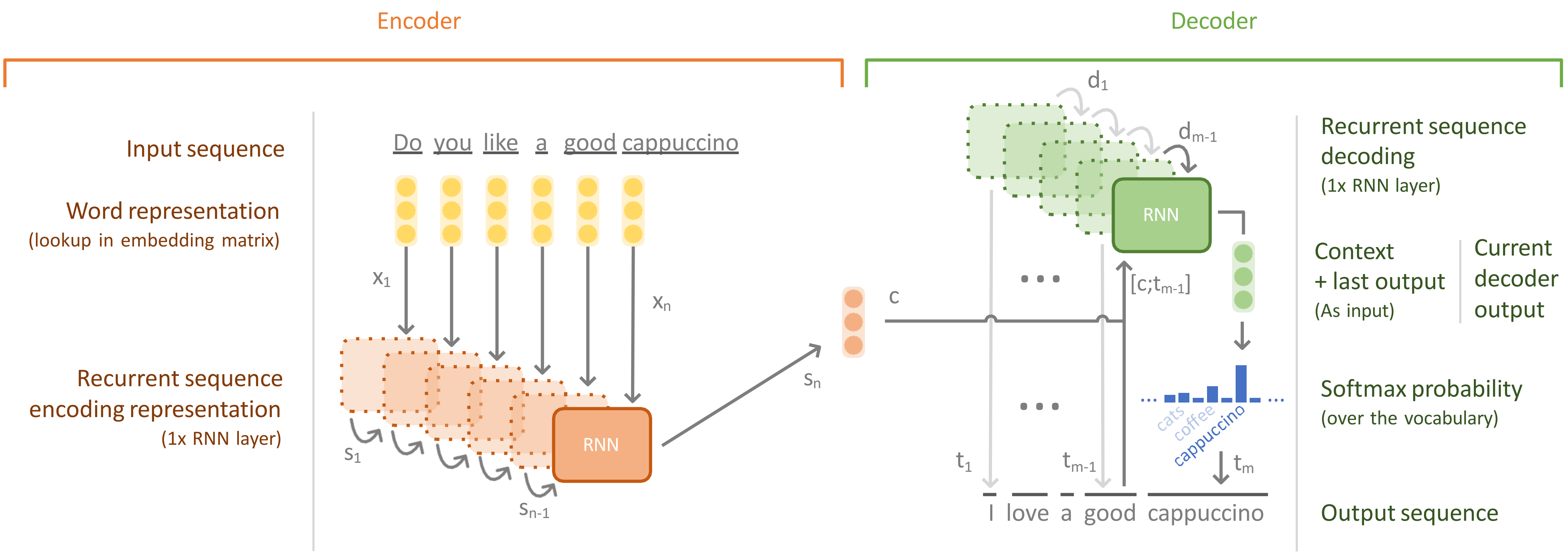
Encoder – Decoder Architecture

- Versatile architecture supporting sequence input & output
  - Based on the training data (and some tweaks) useable for different tasks



The Illustrated Transformer https://jalammar.github.io/illustrated-transformer/

Encoder – Decoder Architecture

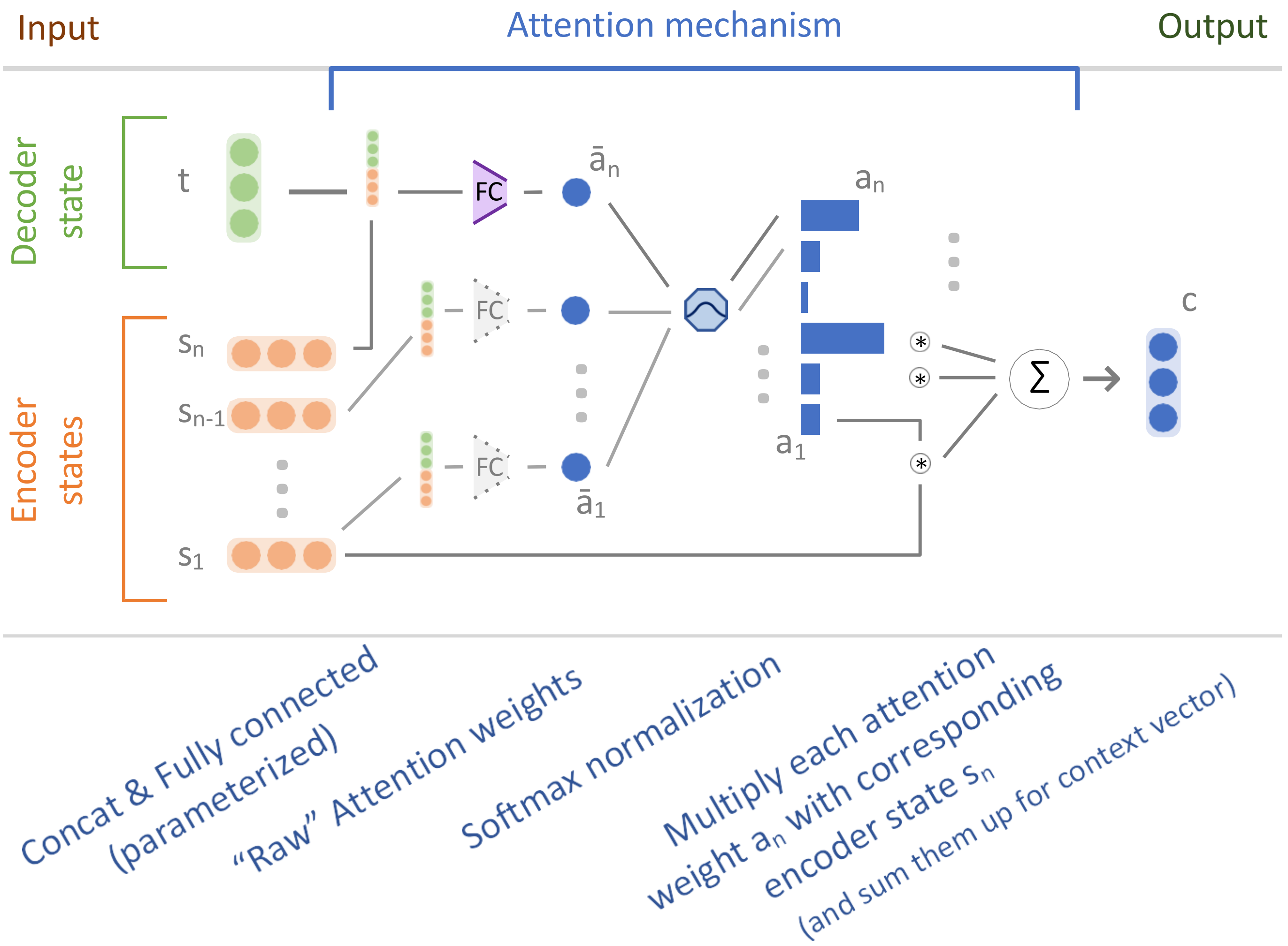


Attention

- Attention mechanism allows to search for relevant parts of the input
  - It creates a weighted average context vector
  - Weights are based on a softmax -> sum up to 1
  - Attention is parameterized & trained end-to-end with the model
- Attention is very effective and versatile
  - Now, there is a jungle of different versions and purposes
- Attention provides some interpretability
  - At least one can show which words have more impact for which output

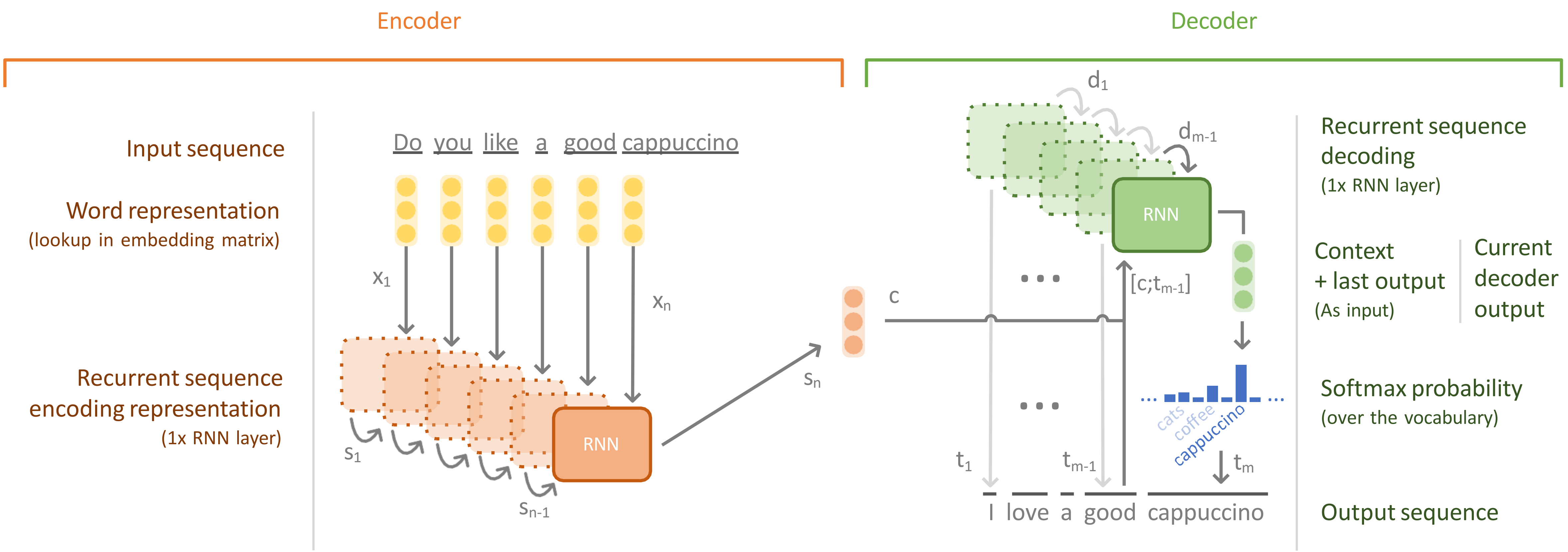
Bahdanau, D., Cho, K. and Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In ICLR

Attention mechanism

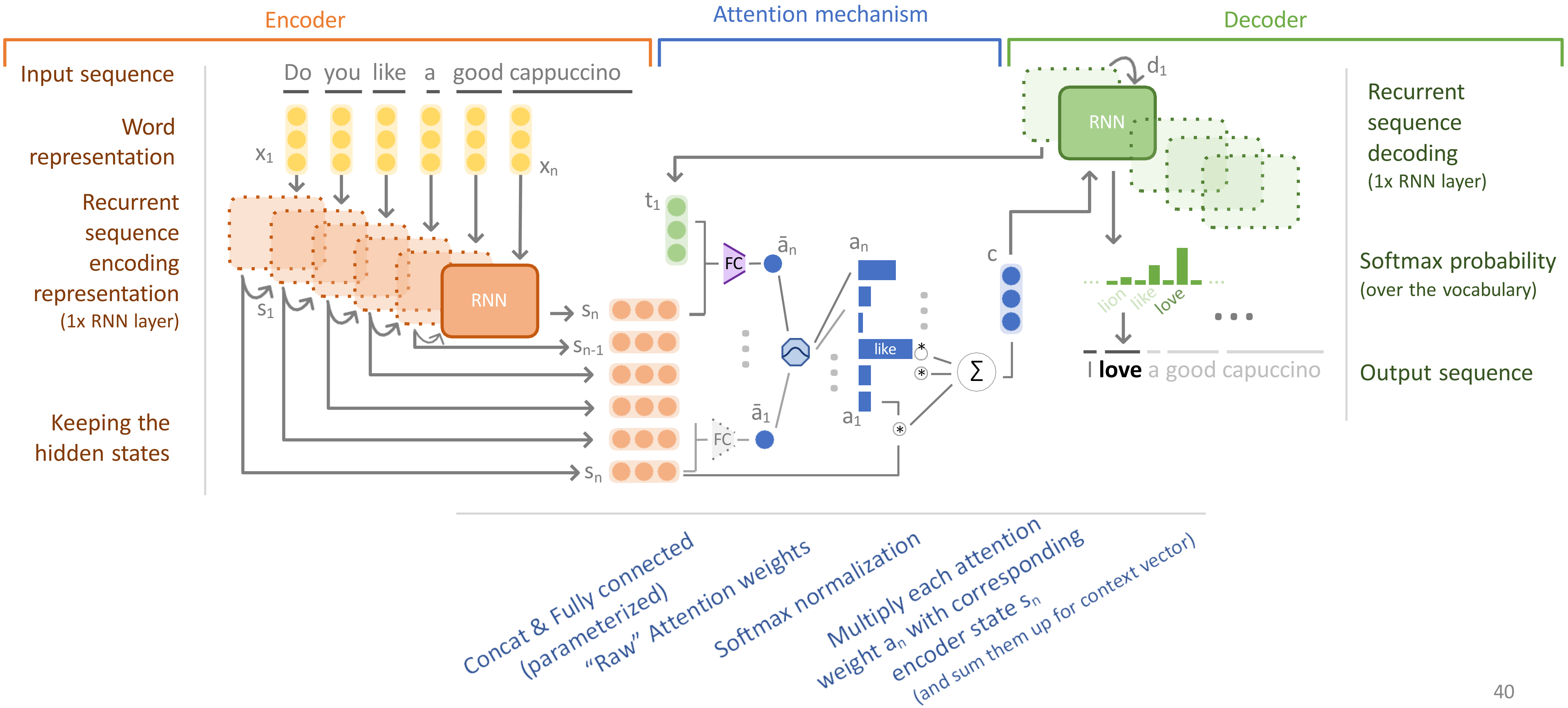


- Each new decoder state produces a new context vector
  - Here, we show only one
- The encoder states are read-only memory
- Fully connected layer contains learned parameters & non-linear activation

Recall the Encoder – Decoder Architecture

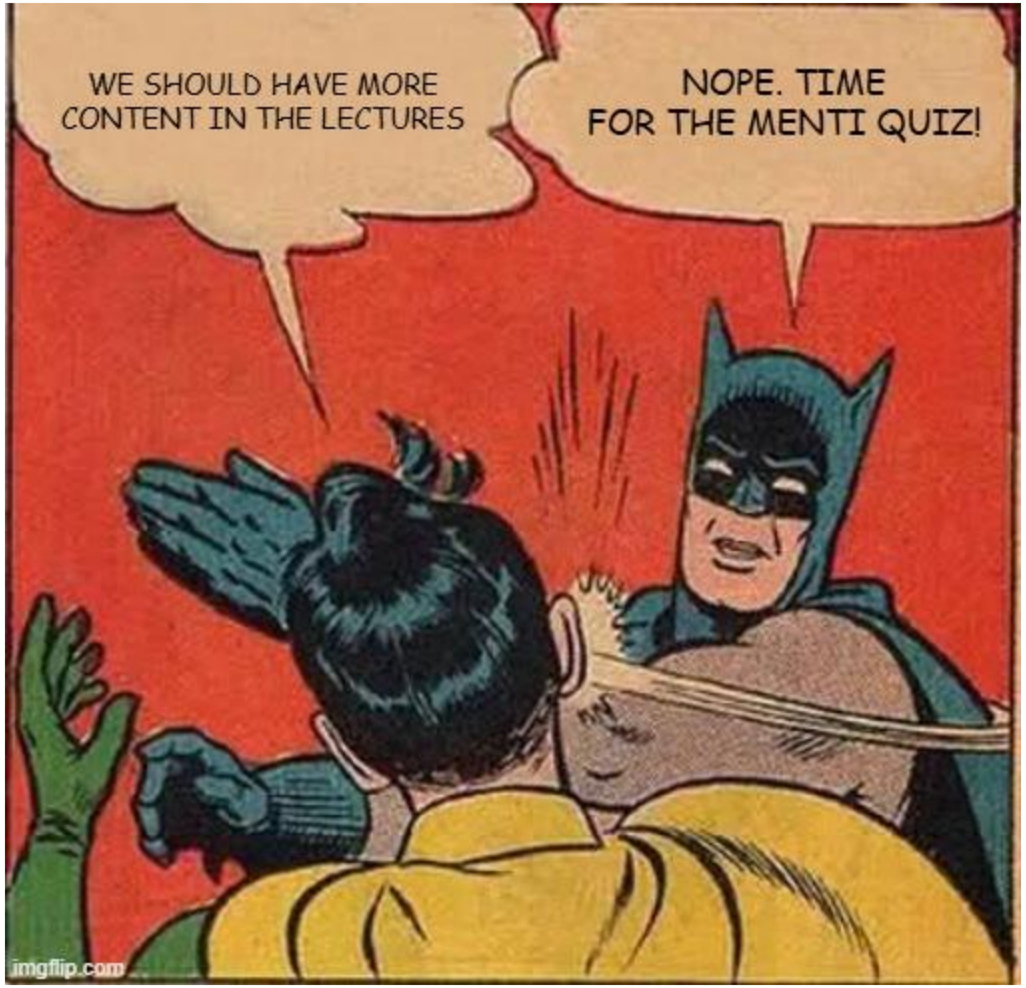


Encoder – Decoder & Attention



Summary: Neural Networks for NLP

1. Neural Networks are versatile – techniques are shared between tasks
2. CNNs can be utilized for n-gram representation learning
3. RNNs model sequences, for input and output


WE SHOULD HAVE MORE CONTENT IN THE LECTURES
NOPE. TIME FOR THE MENTI QUIZ!
imgflip.com

Today

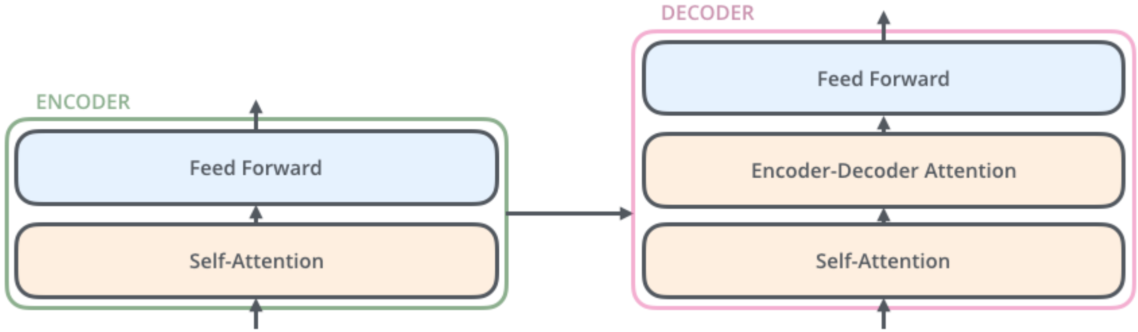
IR – Word Representation / Neural IR

1. Word Embeddings
   - Byte-Pair-Encoding
2. Simple Neural Techniques
   - Convolutional NN
   - Recurrent NN
   - Encoder-Decoder Architecture
3. **Transformer Architecture**
   - BERT Pre-Training

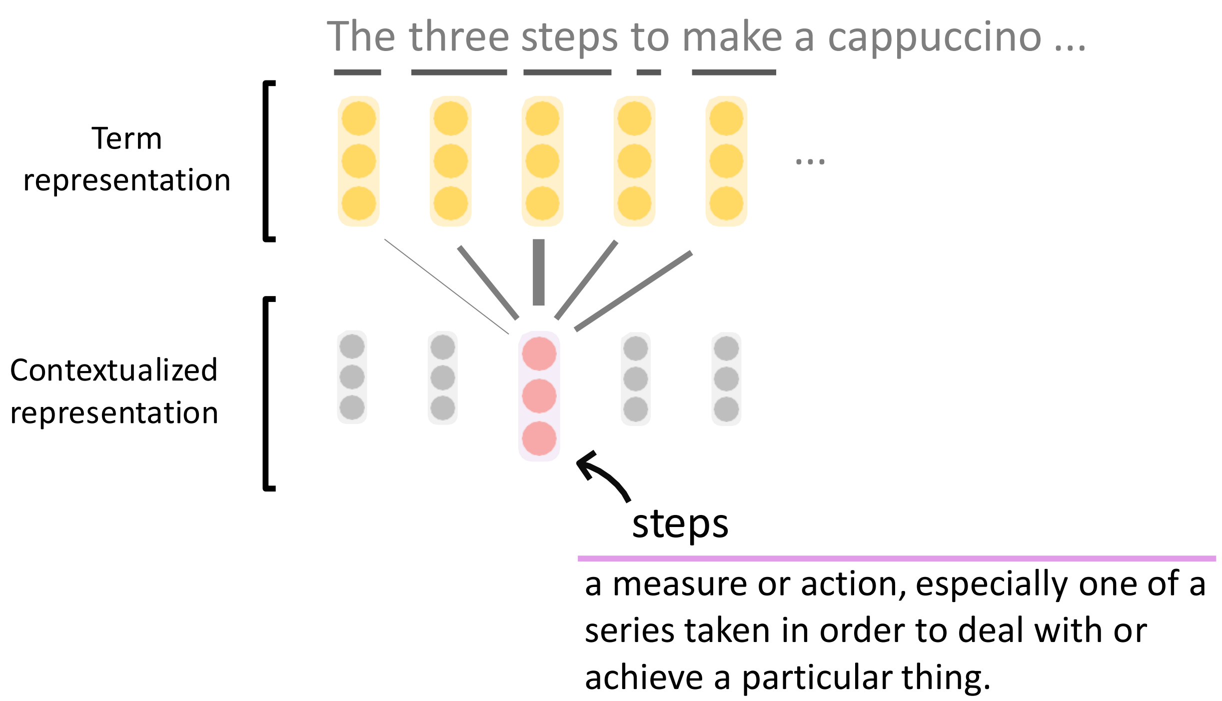
Transformers: Another Versatile Building Block

- The Transformer architecture (as CNNs and RNNs) is not task specific
  - It operates on sequences of vectors, what we do with it is our choice

- Quickly gained huge popularity
  - Pre-trained Transformers are now ubiquitous in NLP & IR research, increasingly also in production systems



Contextualization via Self-Attention



- Learn meaning based on surrounding context for every word occurrence
- This *contextualization* combines representations
- Context here is local to the sequence (not necessary a fixed window)
- Is computationally intensive $O(n^2)$
  - Every token attends to every other token

Transformer

- Transformers contextualize with multiple self-attention units (heads)
  - Every token attends to every other token $O(n^2)$ complexity
- Commonly Transformers stack many layers
- Can be utilized as encoder-only or encoder-decoder combination
- Do not require any recurrence
  - The attention breaks down to a series of matrix multiplications over the sequence
- Initially proposed in translation
  - Now the backbone of virtually every NLP advancement in the last years

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al.
Attention is all you need. In NeurIPS. 2017.

Transformer in PyTorch

- Native support in PyTorch
  - Brings many speed, stability, robustness improvements
  - Raw Transformer Encoder:

```
encoder_layer = nn.TransformerEncoderLayer(d_model=300,nhead=10,dim_feedforward=300)
transformer = nn.TransformerEncoder(encoder_layer, num_layers=2)

src = torch.rand(10, 32, 300)
out = transformer(src)
```

Documentation: https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html
Tutorial: https://pytorch.org/tutorials/beginner/transformer_tutorial.html

In-Depth Resources for Transformers

- Popularity naturally brings more educational content
  - More than we could cover today
- Here are some pointers, if you want to know more about Transformers:

  https://jalammar.github.io/illustrated-transformer/

  https://mccormickml.com/2019/11/11/bert-research-ep-1-key-concepts-and-sources/

  https://github.com/sannykim/transformers

Today

IR – Word Representation / Neural IR

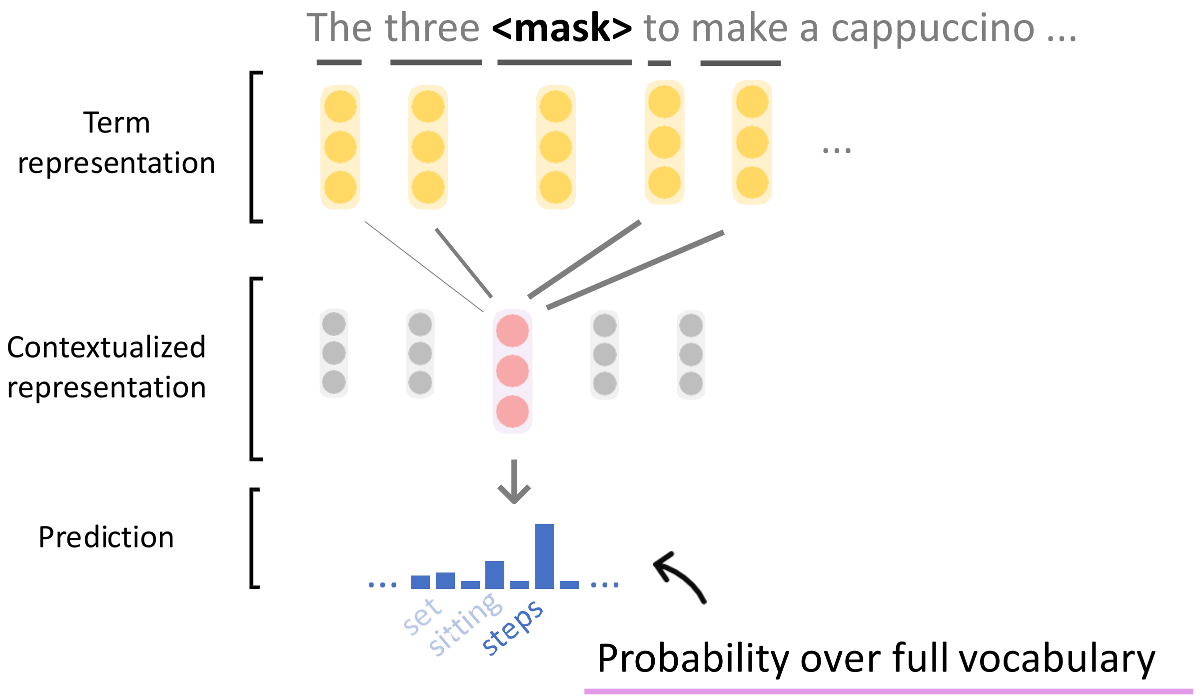
1. **Word Embeddings**
   - Byte-Pair-Encoding
2. **Simple Neural Techniques**
   - Convolutional NN
   - Recurrent NN
   - Encoder-Decoder Architecture
3. **Transformer Architecture**
   - **BERT Pre-Training**

BERT

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- Large effectiveness gains on *all* NLP tasks
- Ingredients:
  - WordPiece Tokenization & Embedding (similar to BPE)
  - Large model (many dimensions and layers – base: 12 layers and 768 dim.)
  - Special tokens (shared use between pre-training and fine-tuning)
    - **[CLS]** Classification token, used as pooling operator to get a single vector per sequence
    - **[MASK]** Used in the masked language model, to predict this word
    - **[SEP]** Used to indicate (+ sequence encodings) a second sentence
  - Long Masked Language Modelling pre-training (weeks if done on 1 GPU)

Devlin et al. 2019 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

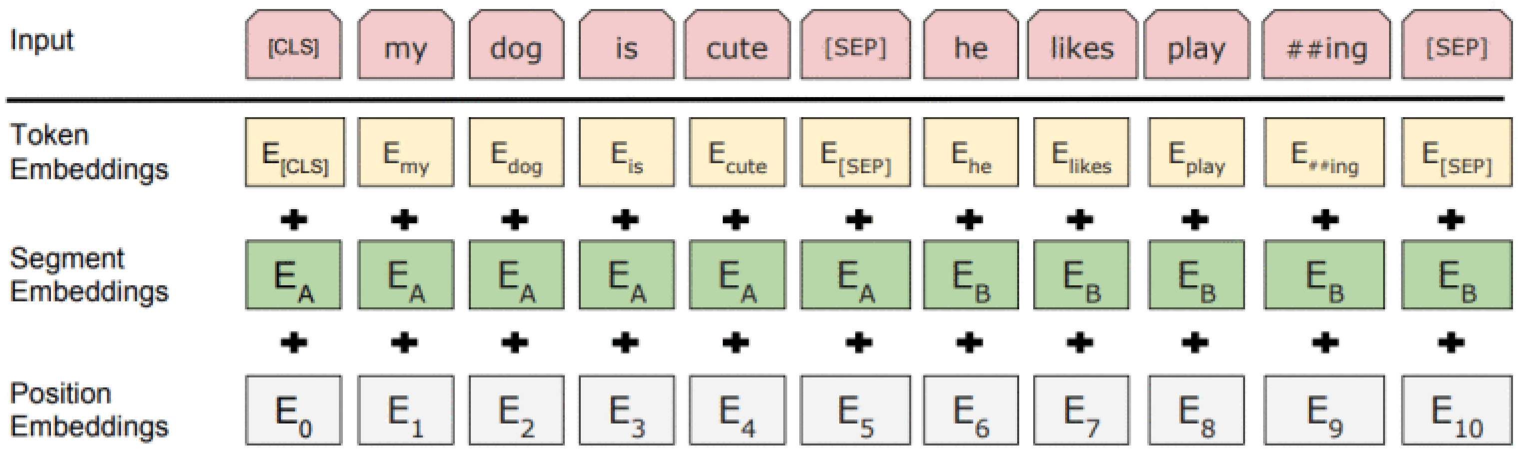
Masked Language Modelling



- Training procedure:
  - Take text and mask random words
  - Try to predict original word from context words
  - Update weights based on difference of prediction vs. actual word

BERT - Input

- Either one or two sentences, always prepended with [CLS]
  - BERT adds trained position embeddings & sequence embeddings



More info: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/

BERT - Model

- Model itself is quite simple: n Layers of stacked Transformers
  - Every Transformer layer receives as input the output of the previous one

- The [CLS] token itself is only special because we train it to be
  - No mechanism inside the model that differentiates it from other tokens

- Novel contributions center around pre-training & workflow

BERT - Workflow

- Someone with lots of compute or time pre-trains a large model
  - BERT uses Masked Language Modelling [MASK] and Next Sentence Prediction [CLS]
- We download it and fine-tune on our task

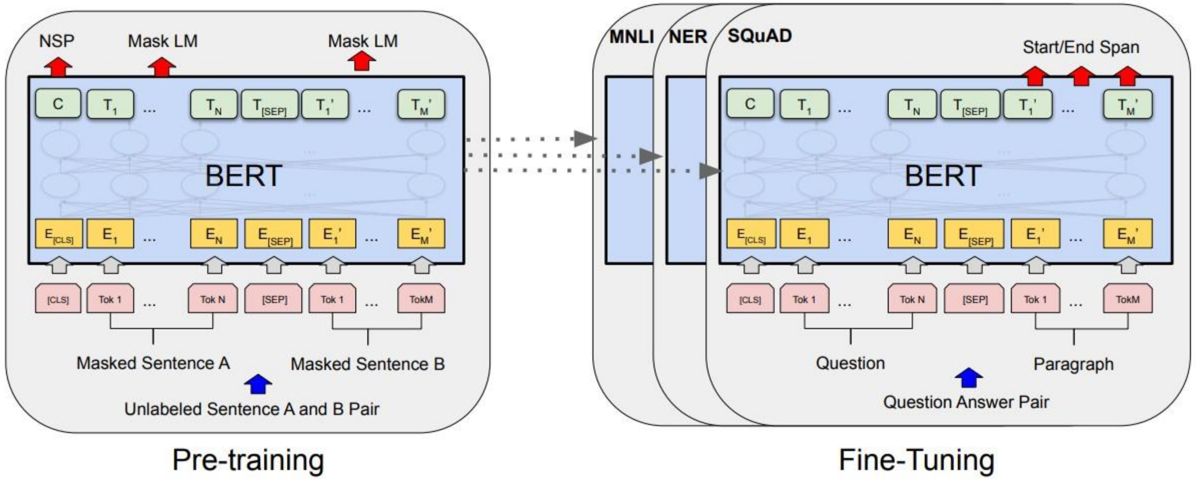


Figure taken from the BERT paper

BERT++

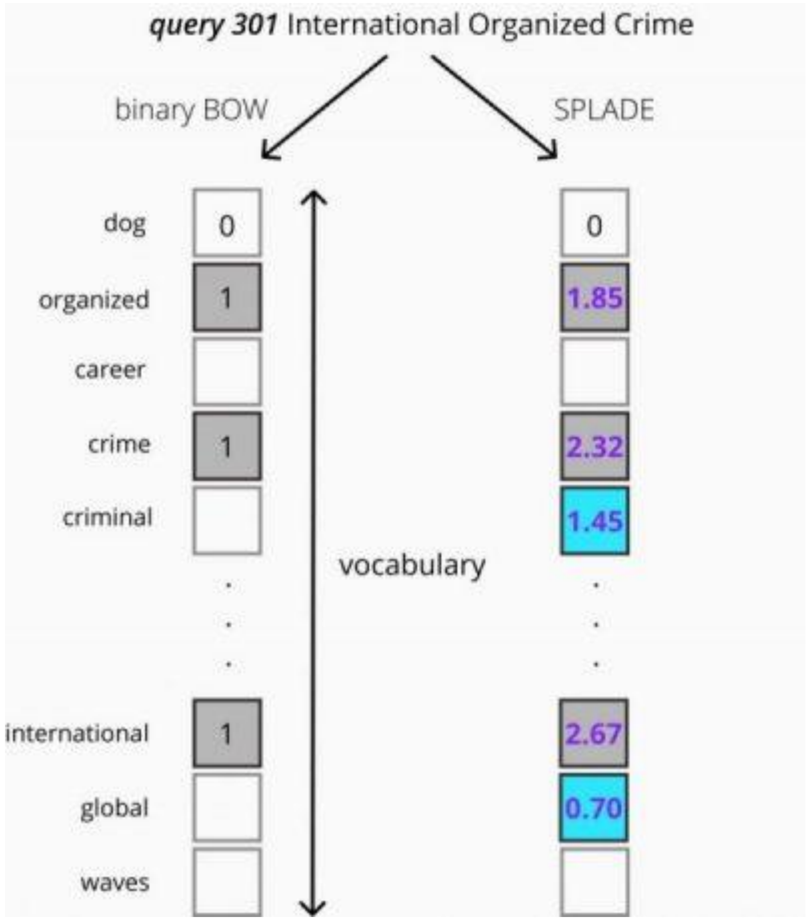
- Same as with Transformer variations, there are now many BERT variants
  - For many languages
  - Domains like biomedical publications
  - Different architectures, but similar workflow:
    Roberta, Transformer-XL, XLNet, Longformer …
- Main themes for adapted architectures:
  - Bigger
  - More efficient
  - Allowing for longer sequences (BERT is capped at 512 tokens in total)

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks
https://doi.org/10.18653/v1/D19-1410
https://sbert.net/

Preview to Practice Class: SPLADE (SParse Lexical AnD Expansion)

- combines BERT's semantic understanding with sparse, interpretable representations

- generates sparse vectors by activating only relevant terms from a large vocabulary



https://europe.naverlabs.com/blog/splade-a-sparse-bi-encoder-bert-based-model-achieves-effective-and-efficient-first-stage-ranking/

Preview to Practice Class: SPLADE (SParse Lexical AnD Expansion)

Simplified steps

- BERT embeddings for each input word (for query / doc)
  - dense, low dimensionality (e.g. 768)
- Expand dense vectors to mix of tokens
  - dense, high dimensionality (e.g. 30.000)
- Enforce sparsity of result vectors through activation function
  - sparse, high dimensionality (e.g. 30.000)
- Regularization to further control the sparsity
  - more sparse, high dimensionality (e.g. 30.000)

https://europe.naverlabs.com/blog/splade-a-sparse-bi-encoder-bert-based-model-achieves-effective-and-efficient-first-stage-ranking/

Preview to Practice Class: SPLADE (SParse Lexical AnD Expansion)

some terms are dropped (compression effect)

original document (doc ID: 7131647)
if (1.2) bow (2.56) legs (1.18) ~~is~~ caused (1.29) by (0.47) ~~the~~ bone (1.2) alignment (1.88) issue (0.87) ~~than you may be~~ able (0.29) ~~to~~ correct (1.37) through (0.43) *bow legs* correction (1.05) ~~exercises.~~ ~~read more here..~~ *if bow legs is caused by the bone alignment issue than you may be able to correct through bow legs correction exercises.*
expansion terms
(leg, 1.62) (arrow, 0.7) (exercise, 0.64) (bones, 0.63) (problem, 0.41) (treatment, 0.35) (happen, 0.29) (create, 0.22) (can, 0.14) (worse, 0.14) (effect, 0.08) (teeth, 0.06) (remove, 0.03)

stemming effect

good expansion terms

bad expansion terms !

https://europe.naverlabs.com/blog/splade-a-sparse-bi-encoder-bert-based-model-achieves-effective-and-efficient-first-stage-ranking/

Summary: Transformers & BERT

1. Transformers apply self-attention to contextualize a sequence

2. BERT pre-trains Transformers for easy downstream use