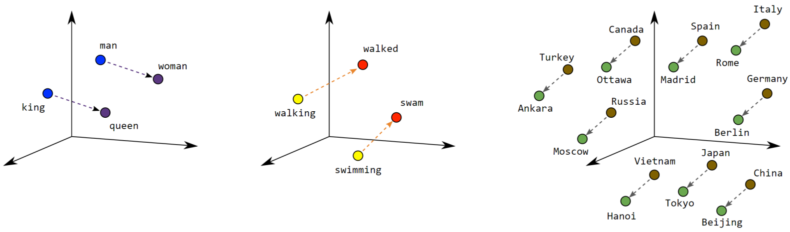
Chapter 9: Transfer Learning & Tokenization

Matthias Aßenmacher

January 13, 2021

Recap: Word vectors



Male-Female | Verb Tense | Country-Capital

Source: *google*

- Information is encoded in (pre-)trained word embeddings
- Embeddings are used for tasks external to the training corpus

What is Transfer Learning?

Wikipedia says:
"Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem."

How it works with word2vec

- Train word2vec on some "fake task" (DBOW or Skip-gram)
- Extract the stored knowledge (a.k.a. embedding)
  or: Directly download embeddings from the web
- Perform a different (supervised) task using the embeddings

A remark on vocabulary & tokenization

Challenges:

- If no embedding for a word exists, it cannot be represented.
- *Workaround:*
  - Train subword (character n-gram) embeddings
  - Represent OOV word as combination of them
- This is already a special case of **Tokenization**

Tokenization examples:

- Whitespace tokenization
- N-grams
- Character n-grams
- Characters

Fine-grained tokenization

Difficulties:

- When using embeddings (or other models/methods) for transferring knowledge, one has to stick to this method's tokenization scheme.
- Using words as tokens leads to vocabulary sizes of easily $> 100k$, which is undesirable.
- Characters as tokens lead to a very small vocabulary size but aggravate capturing meaning.
- Using (sets of) n-grams is kind of heuristic.

Smart alternatives:

- BytePair encoding ▸ Gage (1994) ▸ Sennrich et al. (2016)
- WordPiece ▸ Schuster & Nakajima (2012) ▸ Wu et al. (2016)
- SentencePiece ▸ Kudo et al. (2018)

BytePair encoding (BPE)

Data compression algorithm Gage (1994)

- Considering data on a *byte*-level
- Looking at pairs of bytes:
    1. Count the occurrences of all byte pairs
    2. Find the most frequent byte pair
    3. Replace it with an unused byte
- Repeat this process until no further compression is possible

Open-vocabulary neural machine translation Sennrich et al. (2016)

- Translation as an open-vocabulary problem
- Word-level NMT models:
    - Handling out-of-vocabulary word by using back-off dictionaries
    - Unable to translate or generate previously unseen words
- Subword-level models alleviate this problem

BytePair encoding (BPE)

Adapt BPE for word segmentation ▸ Sennrich et al. (2016)

- *Goal:* Represent an open vocabulary by a vocabulary of fixed size
  $\rightarrow$ Use variable-length character sequences
- Looking at pairs of characters:
  1. Initialize the the vocabulary with all characters plus end-of-word token
  2. Count occurrences and find the most frequent character pair, e.g. "A" and "B" (⚠ Word boundaries are **not** crossed)
  3. Replace it with the new token "AB"
- Only one hyperparameter: Vocabulary size
  (Initial vocabulary + Specified no. of merge operations)
  $\rightarrow$ Repeat this process until given $|V|$ is reached

WordPiece

Voice Search for Japanese and Korean Schuster & Nakajima (2012)

- *Specific Problems:*
  - Asian languages have larger basic character inventories compared to Western languages
  - Concept of spaces between words does (partly) not exist
  - Many different pronounciations for each character
- *WordPieceModel:* Data-dependent + do not produce OOVs
  1. Initialize the the vocabulary with basic Unicode characters (22k for Japanese, 11k for Korean)
     ⚠ Spaces are indicated by an underscore attached before (of after) the respective basic unit or word (increases initial $|V|$ by up to factor 4)
  2. Build a language model using this vocabulary
  3. Merge word units that increase the likelihood on the training data the most, when added to the model
- Two possible stopping criteria:
  Vocabulary size *or* incremental increase of the likelihood

WordPiece

Use for neural machine translation ‣ Wu et al. (2016)

- *Adaptions:*
  - Application to Western languages leads to a lower number of basic units ($\sim$ 500)
  - Add space markers (underscores) *only* at the beginning of words
  - Final vocabulary sizes between 8k and 32k yield a good balance between accuracy and fast decoding speed (compared to around 200k from ‣ Schuster & Nakajima (2012))

Independent **vs.** *joint* **encodings for source & target language**

- Sennrich et al. (2016) report better results for joint BPE
- Wu et al. (2016) use shared WordPieceModel to guarantee identical segmentation in source & target language in order to facilitate copying rare entity names or numbers

SentencePiece ‣ Kudo et al. (2018b)

No need for Pre-Tokenization

- BPE & WordPiece require a sequence of words as inputs
  $\rightarrow$ Some sort of (whitespace) tokenization has to be performed before their application
- SentencePiece (as the name already reveals) doesn't need that
  $\rightarrow$ Can be applied to "raw" sentences
  $\rightarrow$ Consists of *Normalizer*, *Trainer*, *Encoder* & *Decoder*
  $\rightarrow$ Under the hood, two different algorithms are implemented
  - byte-pair encoding ‣ Sennrich et al. (2016)
  - unigram language model ‣ Kudo et al. (2018a)
- No language-specific pre-processing

$\Rightarrow$ Basically a nice, end-to-end usable system/pipeline

Back to Transfer Learning

Embedding Idea + more complex architectures:

- *Naive approach:*
  Standard embeddings (like word2vec) are "*context-free*"
- *Better:*
  - More complex networks, where the embeddings are trainable parameters of the model
  - Model learns *context sensitive* embeddings
  - We already encountered this in the Transformer
- *More complex networks:*
  - Whole network just to learn the embeddings, *or*
  - Additional embedding-layer as the lowest layer

Contextuality

1st Generation of neural embeddings are "context-free":

- Breakthrough paper by Mikolov et al, 2013 (Word2Vec)
- Followed by Pennington et al, 2014 (GloVe)
- Extension of Word2Vec by Bojanowski et al, 2016 (FastText)

Why "Context-free"?

- Models learn *one single* embedding for each word
- Why could this possibly be problematic?
  - "The *default* setting of the function is xyz."
  - "The probability of *default* is rather high."
- Would be nice to have different embeddings for these two occurrences

Transfer Learning – Further Motivation

Questions/Problems:

- Where in this (deep) model do we achieve contextuality?
  (For sure *not* in the lowest layer!)
  → Not straightforward to extract them
- The deeper the network ..
  - .. the more expensive to train
  - .. the more data we need

  → You cannot just train them at home

Transfer Learning

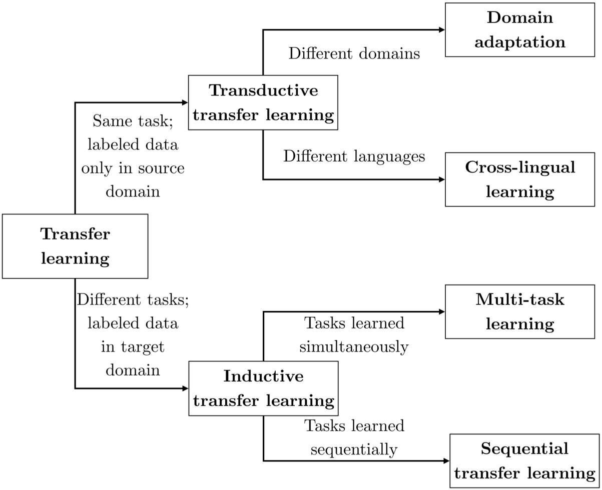
TRANSFER LEARNING

- Train such an architecture on ..
  - .. a fairly *general* task
  - .. which does not require any labels ("*self-supervised*")
  - .. using *large* amounts of data

- Do not extract static embeddings, but use the whole **pre-trained architecture**

- Replace the final layer used for the *general* task by a different layer for a *specific* task at hand

Taxonomy of transfer learning ‣ Ruder, 2019



Source: *Sebastian Ruder*

Taxonomy of transfer learning ▸ Ruder, 2019

Transductive Transfer learning

- Domain adaptation:
  → "*Transfer knowledge learned from performing task A on labeled data from domain X to performing task A in domain Y.*"

- Cross-lingual learning:
  → "*Transfer knowledge learned from performing task A on labeled data from language X to performing task A in language Y.*"

- *Important:* No labeled data in target domain/language Y.

Taxonomy of transfer learning ▸ Ruder, 2019

Inductive Transfer learning

- Multi-task learning:
  → *"Transfer knowledge learned from performing task A on data from domain X to performing multiple (simultaneous) tasks B, C, D, .. in domain Y."*

- Sequential transfer learning:
  → *"Transfer knowledge learned from performing task A on data from domain X to performing multiple (sequential) tasks B, C, D, .. in domain Y."*

- *Important:* Labeled data only for task(s) from target domain Y.

Feature-based transfer learning

Again: Word Embeddings

- The stored knowledge from the pre-trained model is extracted **as is** and is not further adapted to the actual domain/task of interest.
- *Difficulties:*
  - Source & target domain/task might be pretty different
  - No representations for domain-/task-specific words
  - No contextualization

Enhancement: E*mbeddings from* **L***anguage* **Mo***dels (ELMo)*

ELMo ▸ Peters et al., 2018

- Bidirectional language model (LM)
- Combines a forward LM

$$p\left(t_1, t_2, \ldots, t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_1, t_2, \ldots, t_{k-1}\right)$$

and a backward LM

$$p\left(t_1, t_2, \ldots, t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_{k+1}, t_{k+2}, \ldots, t_N\right)$$

to arrive at the following loglikelihood:

$$\sum_{k=1}^{N} \Big(\log p\left(t_k | t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s\right) + \log p\left(t_k | t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s\right)\Big)$$

ELMo embeddings

- Character-based (context-independent) token representations

$$x_k^{LM}$$

- Two-layer biLSTM as main architecture:
  - Two context-dependent token representations *per layer*, i.e.

    $\overrightarrow{\mathbf{h}}_{k,j}^{LM}$ & $\overleftarrow{\mathbf{h}}_{k,j}^{LM}$ for the k-th token in the j-th layer.

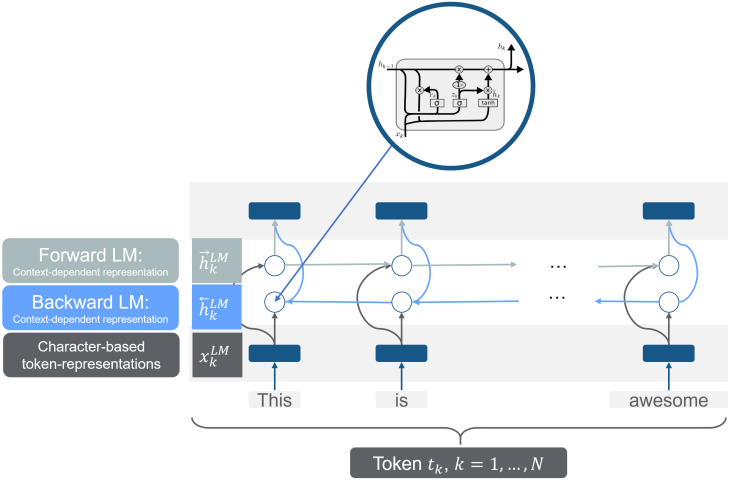
  - Four context-dependent token representations in total:

$$\left\{\overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, 2\right\}$$

- Five representations per token in total:

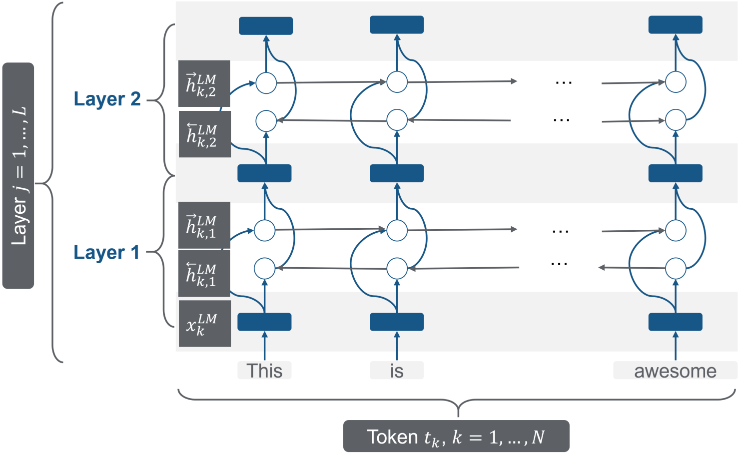
$$\begin{aligned} R_k &= \left\{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \ldots, L\right\} \\ &= \left\{\mathbf{h}_{k,j}^{LM} | j = 0, 1, 2\right\} \end{aligned}$$

ELMo – Graphical representation



Source: *Carolin Becker*

ELMo – Graphical representation



Source: *Carolin Becker*

ELMo – Task Adaption

Including ELMo in downstream tasks:

- Calculate task-specific weights of all five representations:

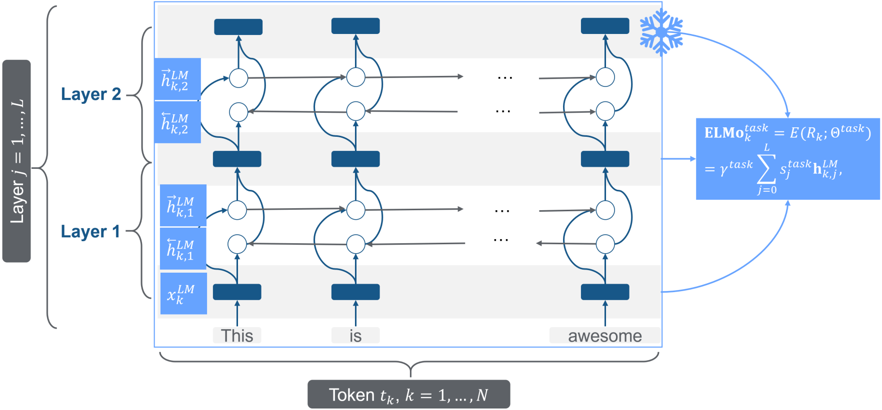
$$\mathbf{ELMo}_k^{task} = E\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM},$$

  where the $\mathbf{h}_{k,j}^{LM}$ are **not trainable** anymore.
- Trainable parameters during the adaption:
  - s_j^{task} are trainable (softmax-normalized) weights
  - γ^{task} is a trainable scaling parameter

Advantages over context free-embeddings:

- Task-specific model has access to *multiple* representations of each token
- Model learns to which degree to use the different representations depending on the task at hand

ELMo – Task Adaption



Source: *Carolin Becker*

Fine-tuning approach

Shortcomings of ELMo:

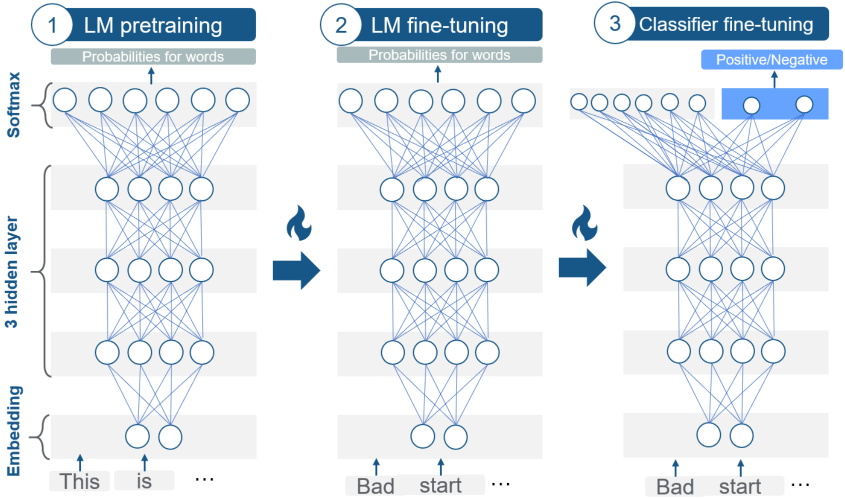
- Pre-trained on a general domain corpus, embeddings are not adapted to the domain/task at hand
- Sequential nature of LSTMs:
  - Not fully parallelizable (compared to Transformers)
  - Fail to capture long-range dependency during contextualization

Alleviations/Alternatives:

- ULMFiT ▸ Howard and Ruder, 2018 is a uni-directional LSTM which is fine-tuned as a whole model on data from the target domain/task.
- GPT ▸ Radford et al., 2018 is a Transformer (decoder) which is fine-tuned as a whole model on data from the target domain/task.

ULMFiT

‣ Howard and Ruder, 2018

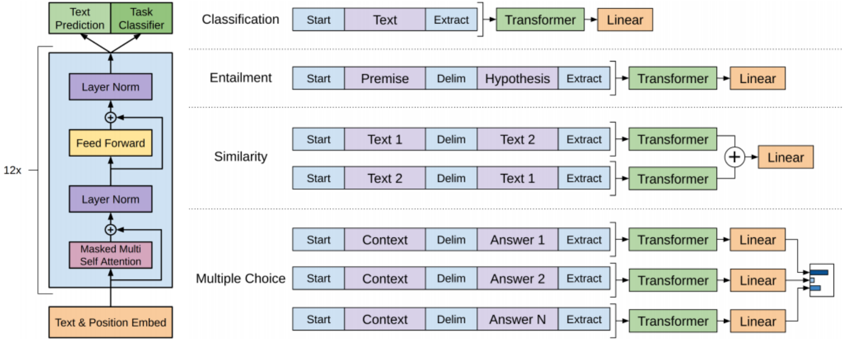


Source: *Carolin Becker*

ULMFiT – Architectural Details

- AWD-LSTMs ▸ Merity et al., 2017 as backbone of the architecture
  - DropConnect ▸ Wan et al., 2013
  - Averaged stochastic gradient descent (ASGD) for optimiziation
- Embedding layer + three LSTM layers + Softmax Layer
- **LM fine-tuning:**
  - Discriminative fine-tuning
- **Classifier fine-tuning:**
  - Concat Pooling
  - Gradual unfreezing

GPT ▸ Radford et al., 2018



Source: *Radford et al., 2018*

GPT – Architectural Details

- Transformer decoder as backbone of the architecture
  - 12-layer-decoder with masked attention heads
  - 40k BPE vocabulary
  - Learned positional embeddings (compared to sinusoidal versions)
- **Fine-tuning:**
  - Linear output layer with softmax activation on top
  - Auxiliary language modeling objective during fine-tuning
    $\rightarrow$ Improves generalization
    $\rightarrow$ Accelerates convegence
  - Task-specific input transformations (see previous slide)

GPT – SOTA results

Performance on different benchmarks:

Table 2: Experimental results on natural language inference tasks, comparing our model with current state-of-the-art methods. 5x indicates an ensemble of 5 models. All datasets use accuracy as the evaluation metric.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Table 3: Results on question answering and commonsense reasoning, comparing our model with current state-of-the-art methods.. 9x means an ensemble of 9 models.

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	76.5	-	-	-
Hidden Coherence Model [7]	77.6	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM (ours)	**86.5**	**62.9**	**57.4**	**59.0**

Source: Radford et al. (2018)

Pre-training objectives

Self-Supervision:

- Special case of unsupervised learning
- Labels are generated from the data itself

Self-supervised objectives:

- Skip-gram objective (cf. word2vec Mikolov et al. (2013a))
- Language modeling objective (cf. Bengio et al. (2003))
- *Masked language modeling (MLM)* objective (cf. chapter 10)
  → Replace words by a `[MASK]` token and train the model to predict
- *Permutation language modeling (PLM)* objective (cf. chapter 11)
  → Autoregressive objective of XLNet
- *Replaced token detection* objective (cf. chapter 11)
  → Requires two models: One performing MLM & the second model to discriminate between actual and the predicted tokens

Pre-training resources

Commonly used (large-scale) data sets for pre-training

- English Wikipedia
- 1B Word Benchmark ▸ Chelba et al. (2013)
- BooksCorpus ▸ Zhu et al. (2015)
- Wikitext-103 ▸ Merity et al. (2016)
- CommonCrawl ▸ https://commoncrawl.org/

Non-exhaustive list; tbc in the following chapters

Transfer Learning in Computer Vision

ImageNet: Deng et al., 2009

- Large-scale data set ($\approx$ 50 million labeled images)
- Hierarchical data set structured in synsets
- "*Diverse coverage of the image world.*" (Deng et al., 2009)

How it changed learning:

- Quasi-standard to use a model pre-trained on ImageNet
- Achieved SOTA results in various computer vision tasks
- Enable the use of large models to small (labeled) data sets

Summary & Outlook

Pros:

- New and deep architectures enable better representation learning
- Leverage favorable properties of language to create self-supervised tasks
- Use ubiquitous large amounts of unlabeled data available on the web

Cons:

- Pre-Training *extremely* costly
- Models will have up to over billions parameters
- Only works well for high-resource languages

Further reading

- *NLP's ImageNet moment has arrived*
- Sebastian Ruder's PhD thesis: ▸ Ruder, 2019