Transfer Learning in NLP
NLPL Winter School

Thomas Wolf - HuggingFace Inc.

Hugging Face: Democratizing NLP

- Core **research** goals:
  - For most: intelligence as **making sense** of data
  - For us: intelligence as **creativity, interaction, adaptability**
- Started with **Conversational AI** (text/image/sound interaction):
  - Neural Language Generation in a Conversational AI game
  - Product used by more than 3M users, 600M+ messages exchanged
- **Develop** & **open-source** tools for **Transfer Learning** in NLP
- We want to **accelerate**, **catalyse** and **democratize** research-level work in Natural Language **Understanding** as well as Natural Language **Generation**

Democratizing NLP – sharing knowledge, code, data

- **Knowledge** sharing
  - NAACL 2019 / EMNLP 2020 Tutorial (Transfer Learning / Neural Lang Generation)
  - Workshop NeuralGen 2019 (Language Generation with Neural Networks)
  - Workshop SustaiNLP 2020 (Environmental/computational friendly NLP)
  - EurNLP Summit 2020 (European NLP summit in Paris in Nov. 2020)
- **Code** & **model** sharing: Open-sourcing the “right way”
  - **Two extremes**: 1000–commands research-code ⇔ 1–command production code
    To target the widest community our goal is to be 👆 right in the middle
  - **Breaking barriers**
    - Researchers / Practitioners
    - PyTorch / TensorFlow
  - **Speeding up** and **fueling** research in Natural Language Processing
    - Make people **stand on the shoulders of giants**

We've built an opi… …t **general-purpose** tools for Natural L…

Features:

- Super **easy** t…
- For **everyone**…
- **State-of-the-**… …sks
- **Reduce costs**… …**anguages**
- Deep interop… …n

1. **BERT** (from Google) released with the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova.
2. **GPT** (from OpenAI) released with the paper Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever.
3. **GPT-2** (from OpenAI) released with the paper Language Models are Unsupervised Multitask Learners by Alec Radford*, Jeffrey Wu*, Rewon Child, David Luan, Dario Amodei** and Ilya Sutskever**.
4. **Transformer-XL** (from Google/CMU) released with the paper Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context by Zihang Dai*, Zhilin Yang*, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov.
5. **XLNet** (from Google/CMU) released with the paper XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang*, Zihang Dai*, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le.
6. **XLM** (from Facebook) released together with the paper Cross-lingual Language Model Pretraining by Guillaume Lample and Alexis Conneau.
7. **RoBERTa** (from Facebook), released together with the paper a Robustly Optimized BERT Pretraining Approach by Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov.
8. **DistilBERT** (from HuggingFace), released together with the paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Victor Sanh, Lysandre Debut and Thomas Wolf. The same method has been applied to compress GPT2 into DistilGPT2, RoBERTa into DistilRoBERTa, Multilingual BERT into DistilmBERT and a German version of DistilBERT.
9. **CTRL** (from Salesforce) released with the paper CTRL: A Conditional Transformer Language Model for Controllable Generation by Nitish Shirish Keskar*, Bryan McCann*, Lav R. Varshney, Caiming Xiong and Richard Socher.
10. **CamemBERT** (from Inria/Facebook/Sorbonne) released with the paper CamemBERT: a Tasty French Language Model by Louis Martin*, Benjamin Muller*, Pedro Javier Ortiz Suárez*, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah and Benoît Sagot.
11. **ALBERT** (from Google Research and the Toyota Technological Institute at Chicago) released with the paper ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, by Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut.
12. **T5** (from Google AI) released with the paper Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer by Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu.
13. **XLM-RoBERTa** (from Facebook AI), released together with the paper Unsupervised Cross-lingual Representation Learning at Scale by Alexis Conneau*, Kartikay Khandelwal*, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer and Veselin Stoyanov.
14. **MMBT** (from Facebook), released together with the paper a Supervised Multimodal Bitransformers for Classifying Images and Text by Douwe Kiela, Suvrat Bhooshan, Hamed Firooz, Davide Testuggine.
15. **Other community models**, contributed by the community.

Transformers library: code example

```
import torch
from transformers import *

# Transformers has a unified API
# for 8 transformer architectures and 30 pretrained weights.
#          Model          | Tokenizer          | Pretrained weights shortcut
MODELS = [(BertModel,       BertTokenizer,       'bert-base-uncased'),
          (OpenAIGPTModel,  OpenAIGPTTokenizer,  'openai-gpt'),
          (GPT2Model,       GPT2Tokenizer,       'gpt2'),
```

💥 Check it out at 💥

https://github.com/huggingface/transformers

```
    tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
    model = model_class.from_pretrained(pretrained_weights)

    # Encode text
    input_ids = torch.tensor([tokenizer.encode("Here is some text to encode", add_special_tokens=True)])
    with torch.no_grad():
        last_hidden_states = model(input_ids)[0]  # Models outputs are now tuples

# Each architecture is provided with several class for fine-tuning on down-stream tasks, e.g.
BERT_MODEL_CLASSES = [BertModel, BertForPreTraining, BertForMaskedLM, BertForNextSentencePrediction,
                      BertForSequenceClassification, BertForMultipleChoice, BertForTokenClassification,
                      BertForQuestionAnswering]
```

Tokenizers library

Now that neu … Deep-Learning
based NLP pi … model inputs.

We have just … **zation**

Features:

- Encode **1** …
- BPE/byte …
- Bindings …
- Link: https://github.com/huggingface/tokenizers

```
pip install tokenizers
```

```
npm install tokenizers
```

```
crates.io/crates/tokenizers
```

Overview

- **Session 1: Transfer Learning - Pretraining and representations**
- Session 2: Transfer Learning - Adaptation and downstream tasks
- Session 3: Transfer Learning - Limitations, open-questions, future directions

Sebastian Ruder

Matthew Peters

Swabha Swayamdipta

Many slides are adapted from **a Tutorial on Transfer Learning in NLP** I gave at NAACL 2019 with my amazing collaborators

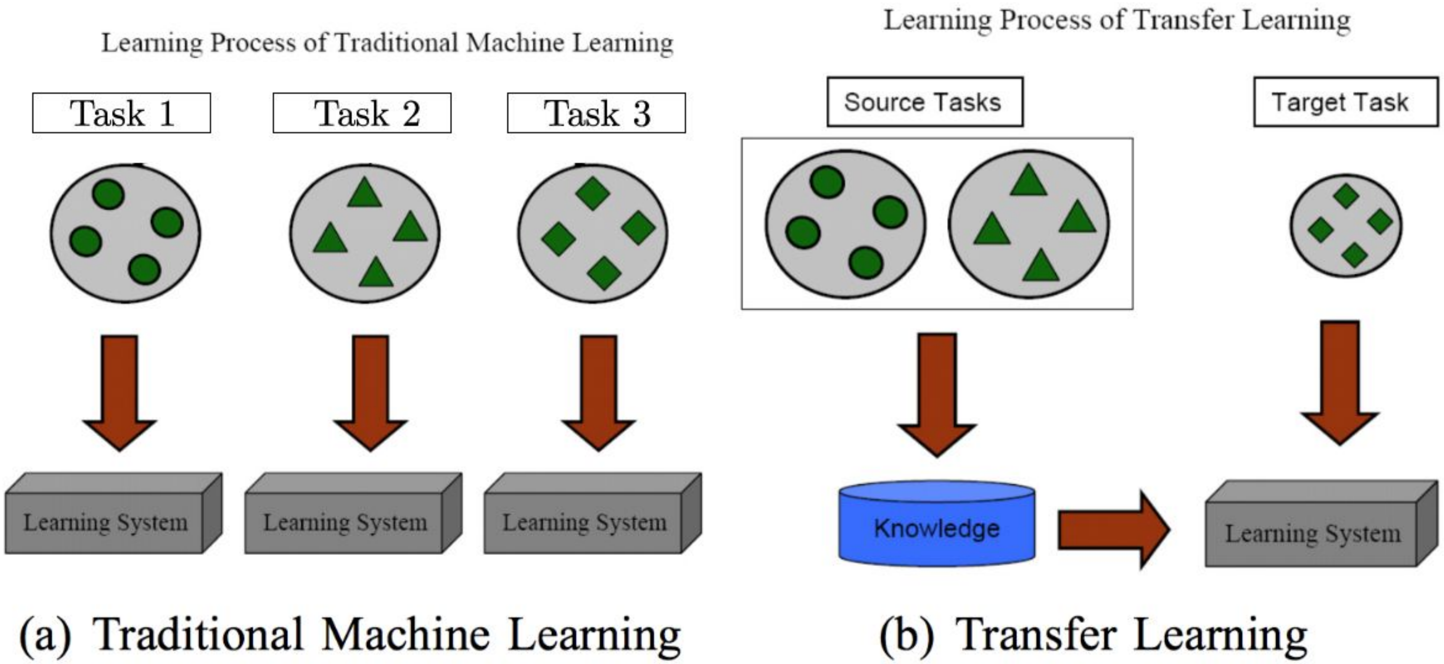
Transfer Learning in NLP
NLPL Winter School
Session 1

Transfer Learning in NLP

Follow along with the tutorial:

- ❏ Colab: https://tinyurl.com/NAACLTransferColab
- ❏ Code: https://tinyurl.com/NAACLTransferCode

What is transfer learning?



Pan and Yang (2010)

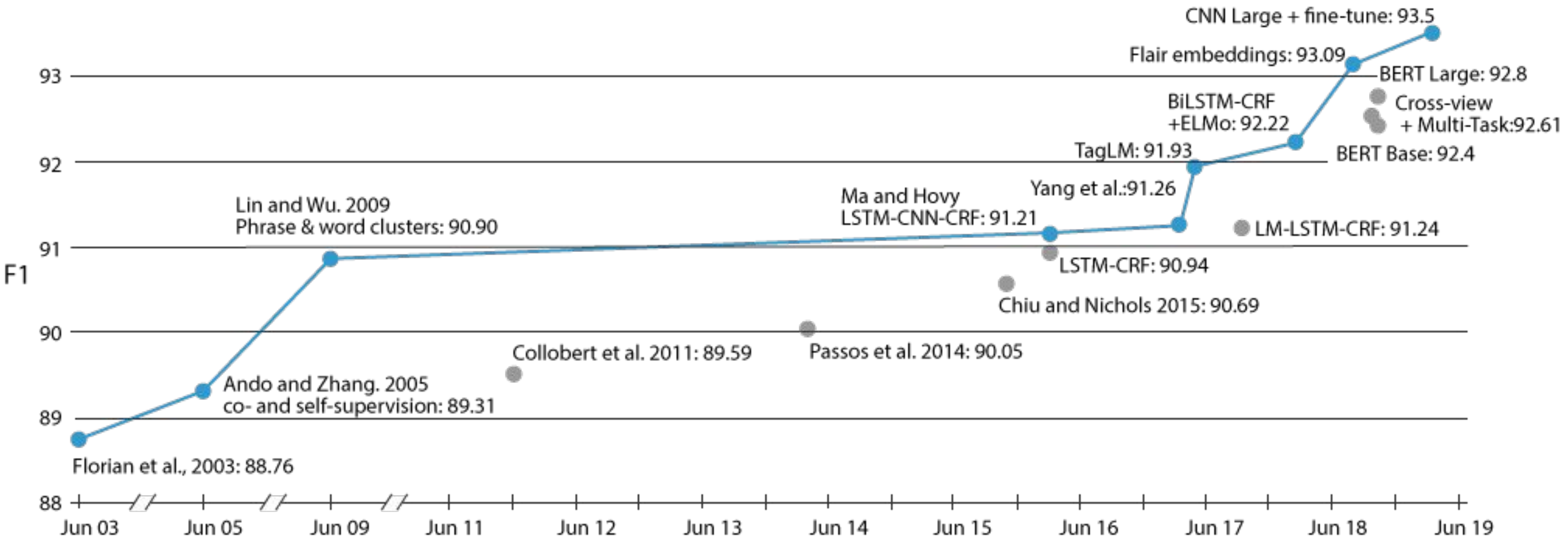
Why transfer learning in NLP?

- ❑ Many NLP tasks share common knowledge about language (e.g. linguistic representations, structural similarities)
- ❑ Tasks can inform each other—e.g. syntax and semantics
- ❑ Annotated data is rare, make use of as much supervision as available.

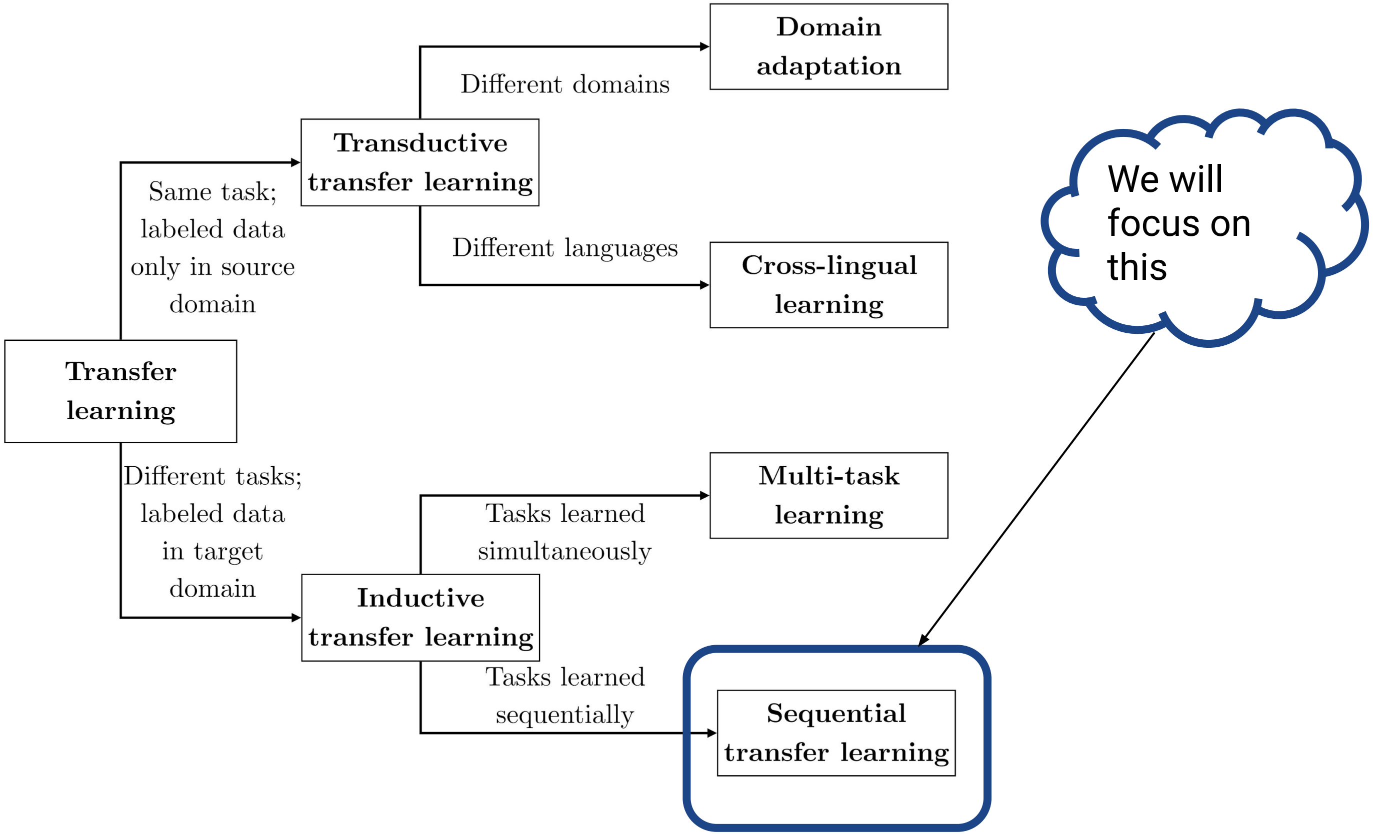
- ❑ Empirically, transfer learning has resulted in SOTA for many supervised NLP tasks (e.g. classification, information extraction, Q&A, etc).

Why transfer learning in NLP? (Empirically)

Performance on Named Entity Recognition (NER) on CoNLL-2003 (English) over time



Types of transfer learning in NLP



Ruder (2019)

What this tutorial is about and what it's not about

- Goal: provide broad overview of transfer methods in NLP, focusing on the most empirically successful methods *as of mid 2019*
- Provide practical, hands on advice → by end of tutorial, everyone has ability to apply recent advances to text classification task

- What this is not: **Comprehensive** (it's impossible to cover all related papers in one tutorial!)
- (Bender Rule: This tutorial is mostly for work done in English, extensibility to other languages depends on availability of data and resources.)

Agenda

[1] Introduction

[2] Pretraining

[3] What's in a representation?

[4] Adaptation

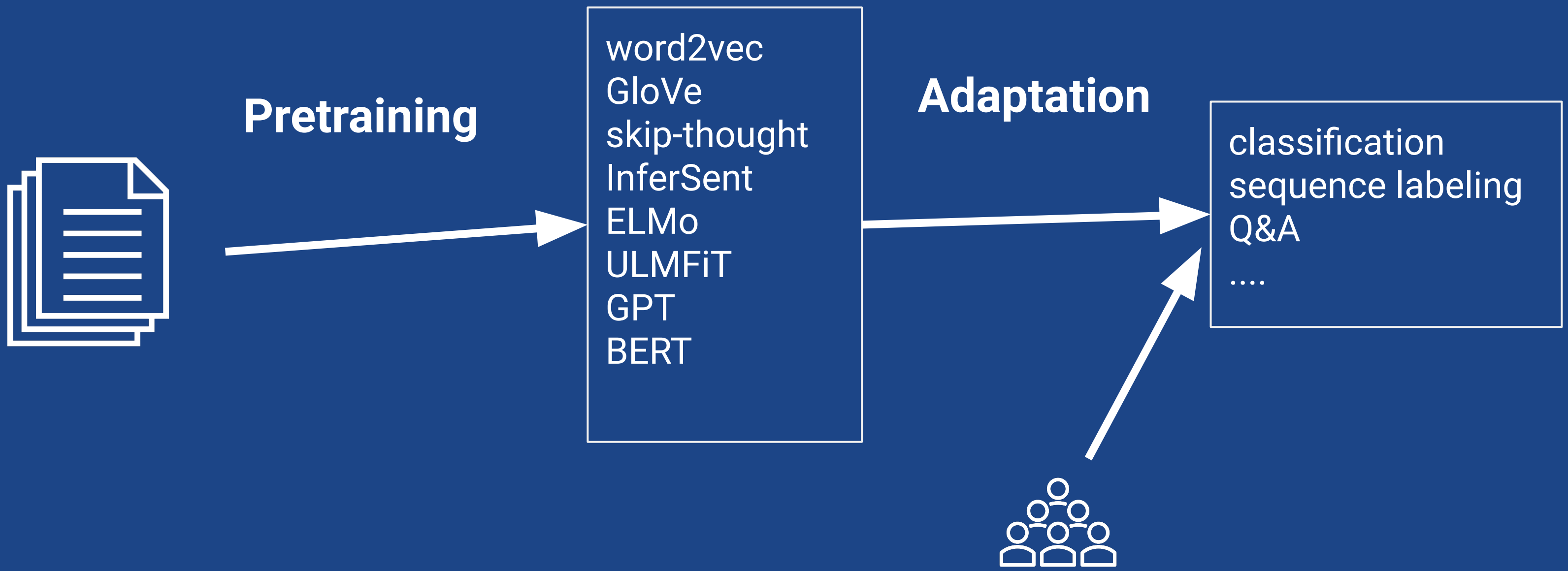
[5] Downstream

[6] Open Problems

1. Introduction

Sequential transfer learning

Learn on one task / dataset, then transfer to another task / dataset



Pretraining tasks and datasets

- Unlabeled data and self-supervision
  - Easy to gather very large corpora: Wikipedia, news, web crawl, social media, etc.
  - Training takes advantage of distributional hypothesis: “You shall know a word by the company it keeps” (Firth, 1957), often formalized as training some variant of language model
  - Focus on efficient algorithms to make use of plentiful data
- Supervised pretraining
  - Very common in vision, less in NLP due to lack of large supervised datasets
  - Machine translation
  - NLI for sentence representations
  - Task-specific—transfer from one Q&A dataset to another

Target tasks and datasets

Target tasks are typically supervised and span a range of common NLP tasks:

- ❑ Sentence or document classification (e.g. sentiment)
- ❑ Sentence pair classification (e.g. NLI, paraphrase)
- ❑ Word level (e.g. sequence labeling, extractive Q&A)
- ❑ Structured prediction (e.g. parsing)
- ❑ Generation (e.g. dialogue, summarization)

Concrete example—word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

cat = [0.1, -0.2, 0.4, …]

dog = [0.2, -0.1, 0.7, …]

Concrete example—word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

cat = [0.1, -0.2, 0.4, …]

dog = [0.2, -0.1, 0.7, …]

PRP	VBP	PRP	NN	CC	NN	.
I	love	my	cat	and	dog	.

Concrete example—word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

cat = [0.1, -0.2, 0.4, ...]

dog = [0.2, -0.1, 0.7, ...]

PRP	VBP	PRP	NN	CC	NN	.
I	love	my	cat	and	dog	.

I love my cat and dog . }-> “positive"

Major Themes

Major themes: From words to words-in-context

Word vectors

cats = [0.2, -0.3, …]

dogs = [0.4, -0.5, …]

Sentence / doc vectors

We have two cats. } [-1.2, 0.0, …]

It's raining cats and dogs. } [0.8, 0.9, …]

Word-in-context vectors

We have two $\overbrace{\text{cats}}^{[1.2,\ -0.3,\ \ldots]}$.

It's raining $\overbrace{\text{cats}}^{[-0.4,\ 0.9,\ \ldots]}$ and dogs.

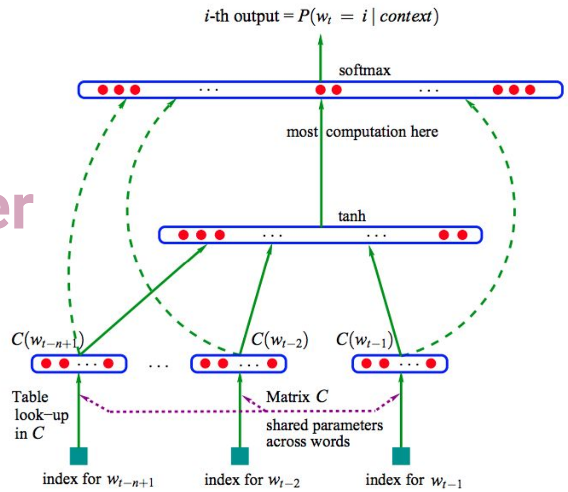
Major themes: LM pretraining

- Many successful pretraining approaches are based on language modeling
- Informally, a LM learns P_θ*(text)* or P_θ*(text | some other text)*

- Doesn't require human annotation
- Many languages have enough text to learn high capacity model
- Versatile—can learn both sentence and word representations with a variety of objective functions

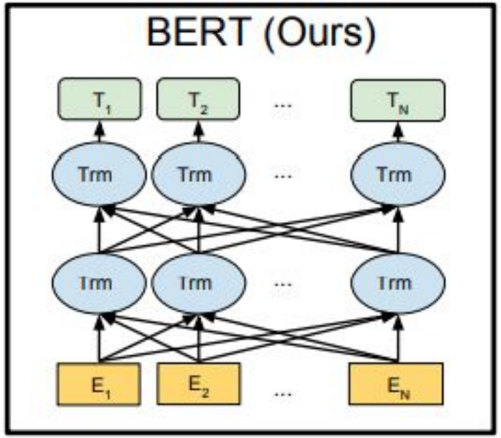
Major themes: From shallow to deep

1 layer



[Bengio et al 2003: A Neural Probabilistic Language Model]()

24 layers



[Devlin et al 2019: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding]()

Major themes: pretraining vs target task

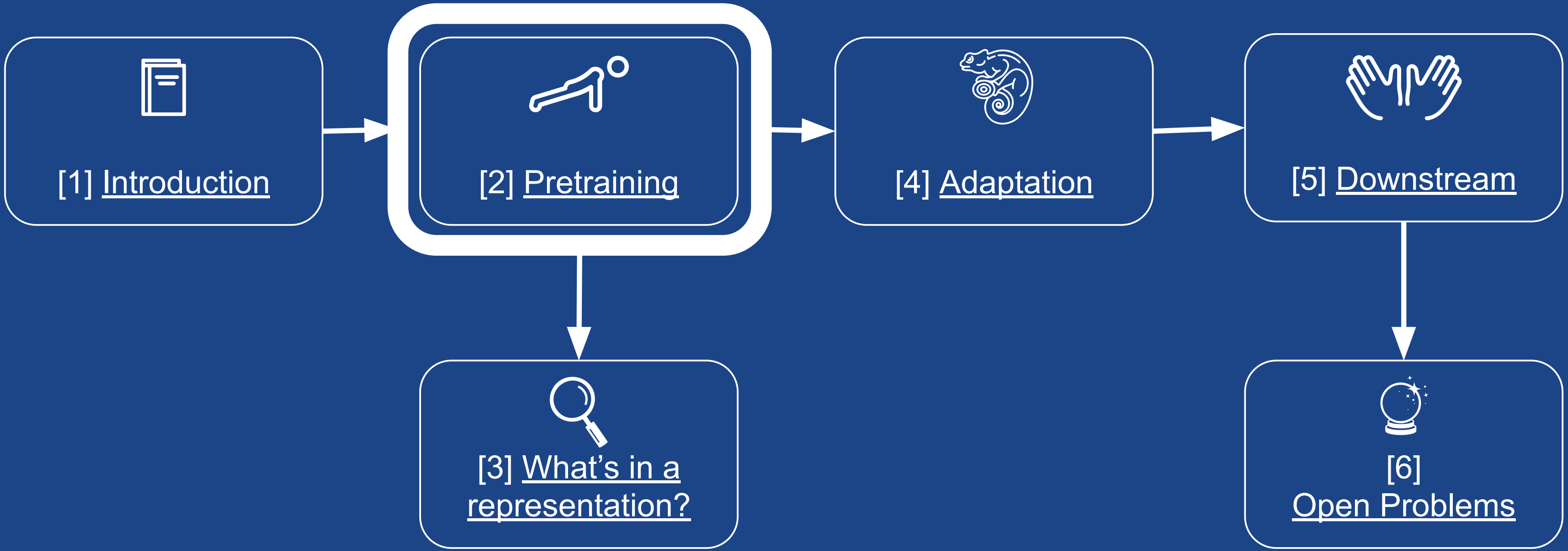
Choice of pretraining and target tasks are coupled

- ❏ Sentence / document representations not useful for word level predictions
- ❏ Word vectors can be pooled across contexts, but often outperformed by other methods
- ❏ In contextual word vectors, bidirectional context important

In general:

- ❏ Similar pretraining and target tasks → best results

Agenda


[1] Introduction
[2] Pretraining
[3] What’s in a representation?
[4] Adaptation
[5] Downstream
[6] Open Problems

2. Pretraining

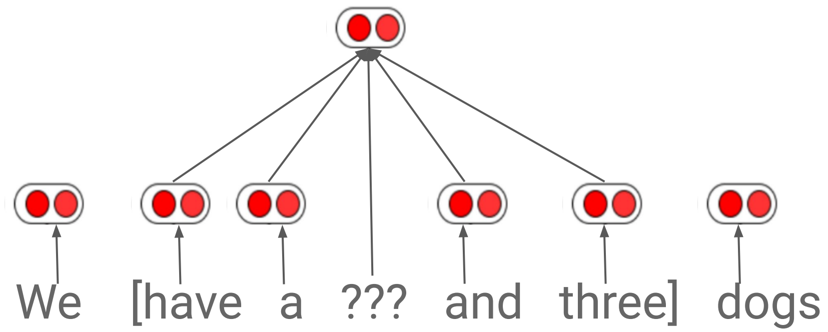
Image credit: Creative Stall

Overview

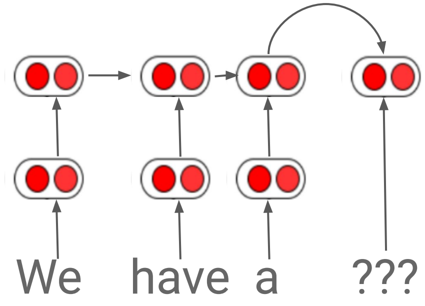
- ❏ Language model pretraining
- ❏ Word vectors
- ❏ Sentence and document vectors
- ❏ Contextual word vectors
- ❏ Interesting properties of pretraining
- ❏ Cross-lingual pretraining

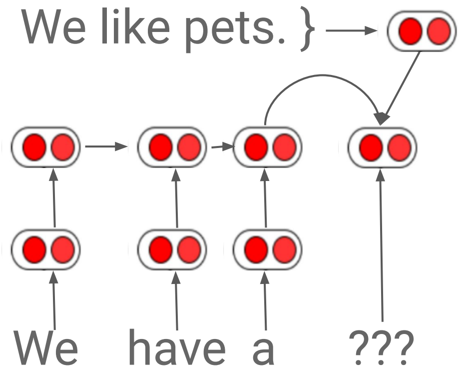
LM pretraining

word2vec, Mikolov et al (2013)



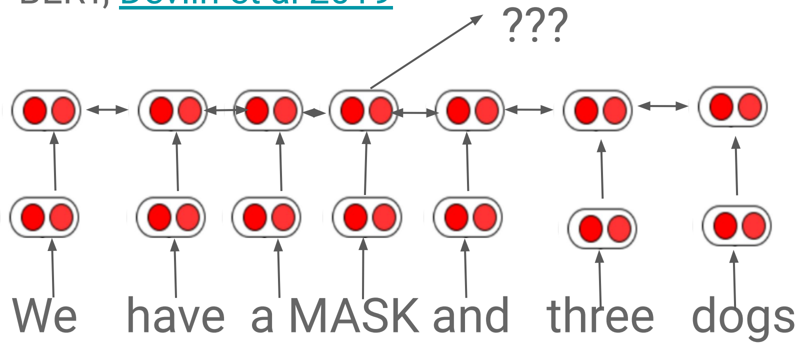
ELMo, Peters et al. 2018, ULMFiT (Howard & Ruder 2018), GPT (Radford et al. 2018)





Skip-Thought (Kiros et al., 2015)

BERT, Devlin et al 2019



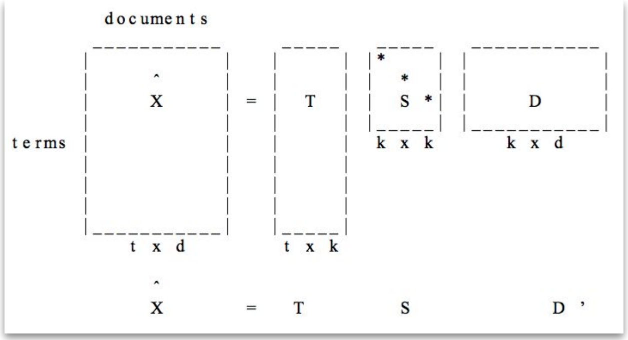
Word vectors

Why embed words?

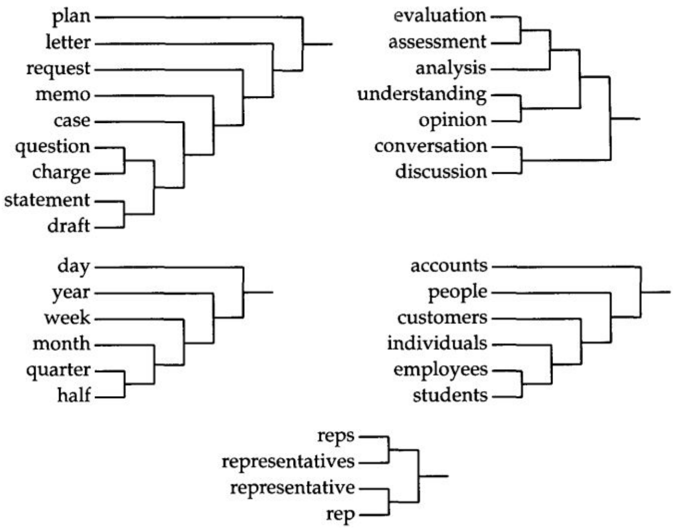
- Embeddings are themselves parameters—can be learned
- Sharing representations across tasks
- Lower dimensional space
  - Better for computation—difficult to handle sparse vectors.

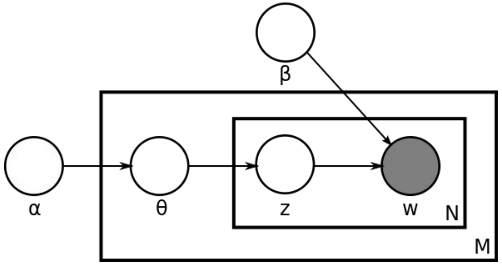
Unsupervised pretraining : Pre-Neural

Latent Semantic Analysis (LSA)—SVD of term-document matrix, (Deerwester et al., 1990)



Brown clusters, hard hierarchical clustering based on n-gram LMs, (Brown et al. 1992)

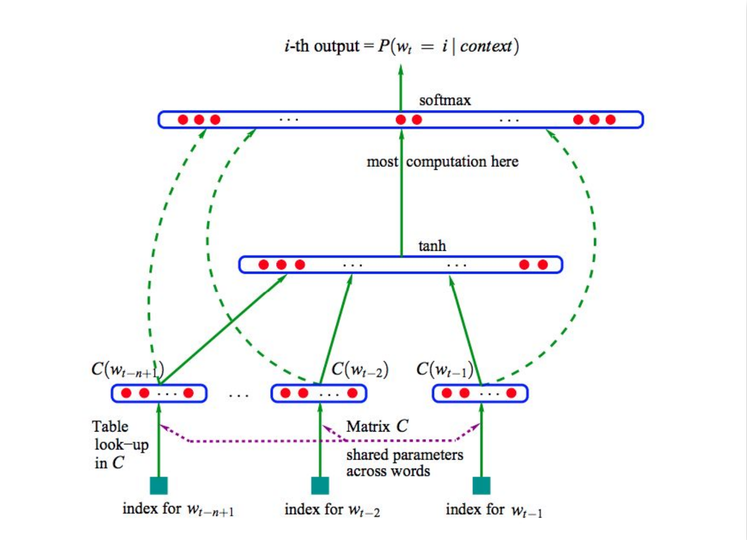




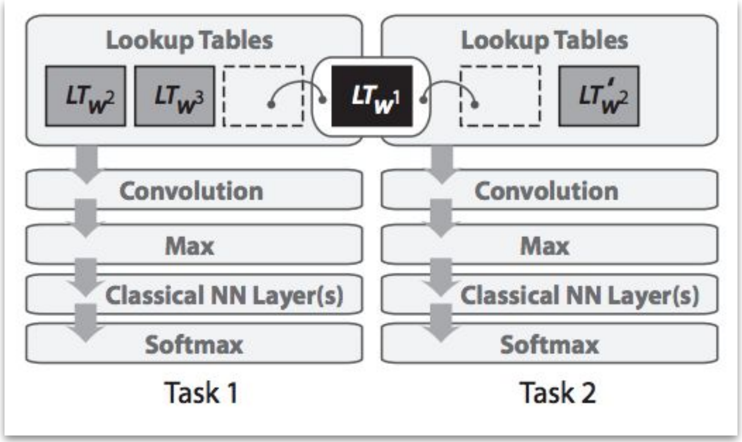
Latent Dirichlet Allocation (LDA)—Documents are mixtures of topics and topics are mixtures of words (Blei et al., 2003)

Word vector pretraining

n-gram neural language model (Bengio et al. 2003)



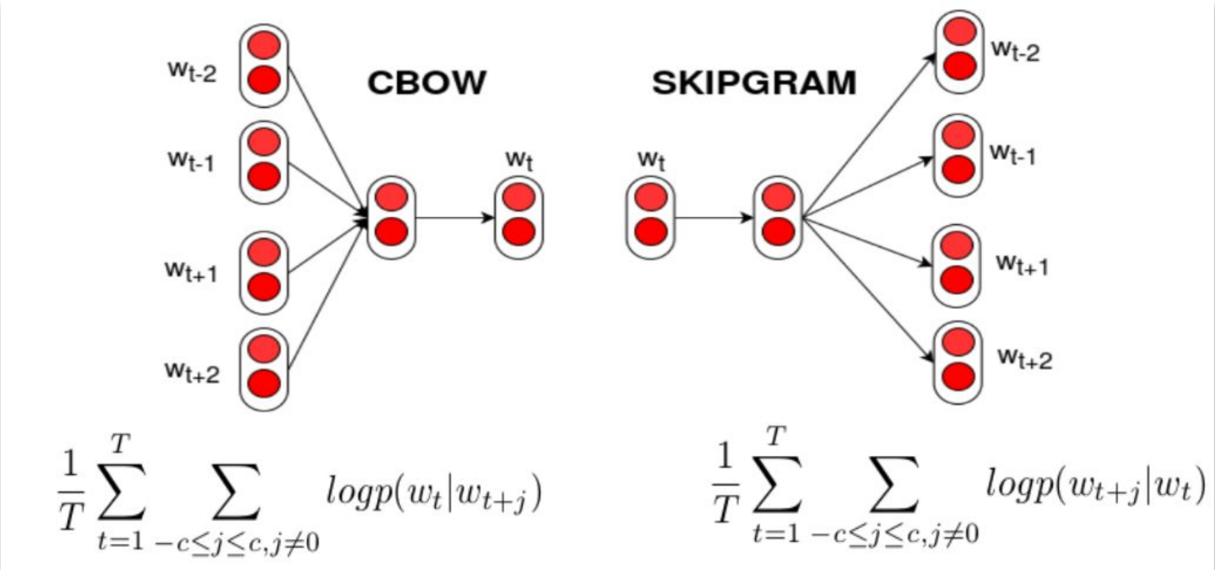
Supervised multitask word embeddings (Collobert and Weston, 2008)



word2vec

Efficient algorithm + large scale training → high quality word vectors

(Mikolov et al., 2013)



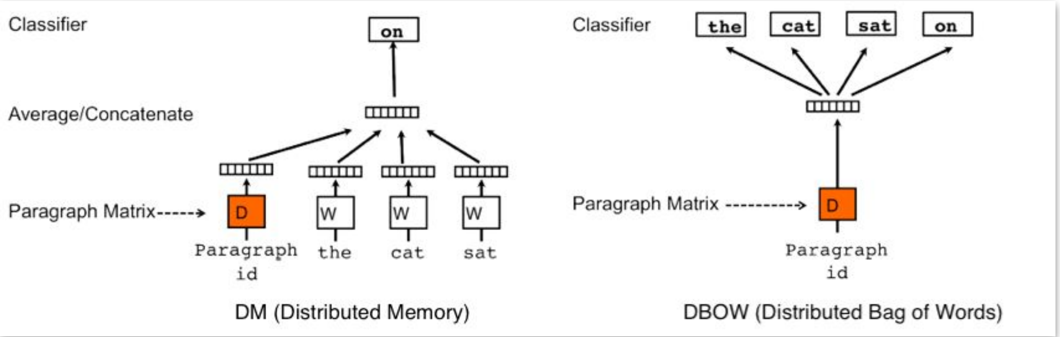
See also:

- Pennington et al. (2014): GloVe
- Bojanowski et al. (2017): fastText

Sentence and document vectors

Paragraph vector

Unsupervised paragraph embeddings (Le & Mikolov, 2014)

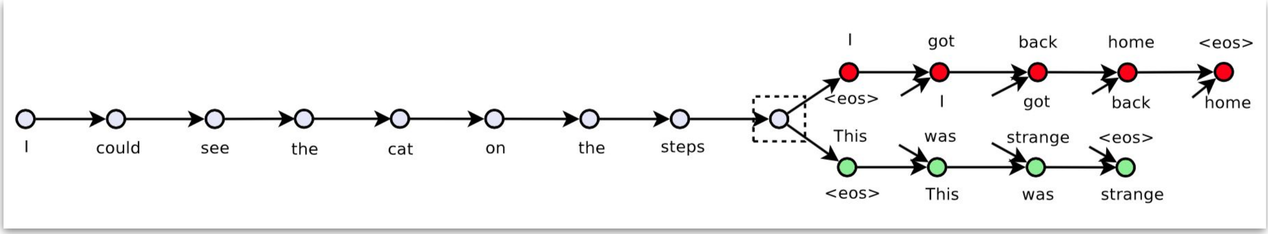


SOTA classification (IMDB, SST)

Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

Skip-Thought Vectors

Predict previous / next sentence with seq2seq model (Kiros et al., 2015)

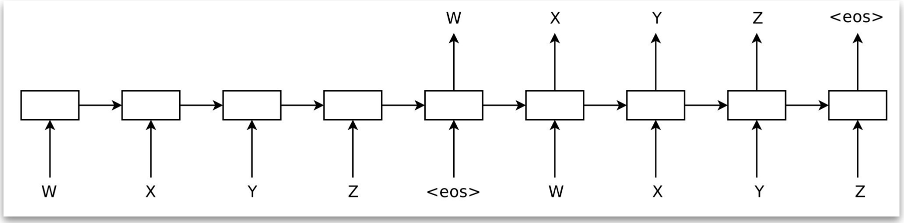


Method	MR	CR	SUBJ	MPQA	TREC
NB-SVM [41]	79.4	81.8	93.2	86.3	
MNB [41]	79.0	80.0	93.6	86.3	
cBoW [6]	77.2	79.9	91.3	86.4	87.3
GrConv [6]	76.3	81.3	89.5	84.5	88.4
RNN [6]	77.2	82.3	93.7	90.1	90.2
BRNN [6]	82.3	82.6	94.2	90.3	91.0
CNN [4]	81.5	85.0	93.4	89.6	**93.6**
AdaSent [6]	**83.1**	**86.3**	**95.5**	**93.3**	92.4
Paragraph-vector [7]	74.8	78.1	90.5	74.2	91.8
uni-skip	75.5	79.3	92.1	86.9	91.4
bi-skip	73.9	77.9	92.5	83.3	89.4
combine-skip	76.5	80.1	93.6	87.1	92.2
combine-skip + NB	80.4	81.3	93.6	87.5	

Hidden state of encoder transfers to sentence tasks (classification, semantic similarity)

Autoencoder pretraining

Dai & Le (2015): Pretrain a sequence autoencoder (SA) and generative LM



SOTA classification (IMDB)

Model	Test error rate
LSTM with tuning and dropout	13.50%
LSTM initialized with word2vec embeddings	10.00%
LM-LSTM (see Section 2)	7.64%
SA-LSTM (see Figure 1)	7.24%
SA-LSTM with linear gain (see Section 3)	9.17%
SA-LSTM with joint training (see Section 3)	14.70%
Full+Unlabeled+BoW [21]	11.11%
WRRBM + BoW (bnc) [21]	10.77%
NBSVM-bi (Naïve Bayes SVM with bigrams) [35]	8.78%
seq2-bown-CNN (ConvNet with dynamic pooling) [11]	7.67%
Paragraph Vectors [18]	7.42%

See also:

- Socher et. al (2011): Semi-supervised recursive auto encoder
- Bowman et al. (2016): Variational autoencoder (VAE)
- Hill et al. (2016): Denoising autoencoder

Supervised sentence embeddings

Also possible to train sentence embeddings with supervised objective

- Paragram-phrase: uses paraphrase database for supervision, best for paraphrase and semantic similarity (Wieting et al. 2016)
- InferSent: bi-LSTM trained on SNLI + MNLI (Conneau et al. 2017)
- GenSen: multitask training (skip-thought, machine translation, NLI, parsing) (Subramanian et al. 2018)

Contextual word vectors

Contextual word vectors - Motivation

Word vectors compress all contexts into a *single vector*

Nearest neighbor GloVe vectors to “**play**”

VERB	NOUN	ADJ	??
playing	game	multiplayer	plays
played	games		Play
	players		
	football		

Contextual word vectors - Key Idea

Instead of learning one vector per word, learn a vector that depends on context

f(play | The kids play a game in the park.)

!=

f(play | The Broadway play premiered yesterday.)

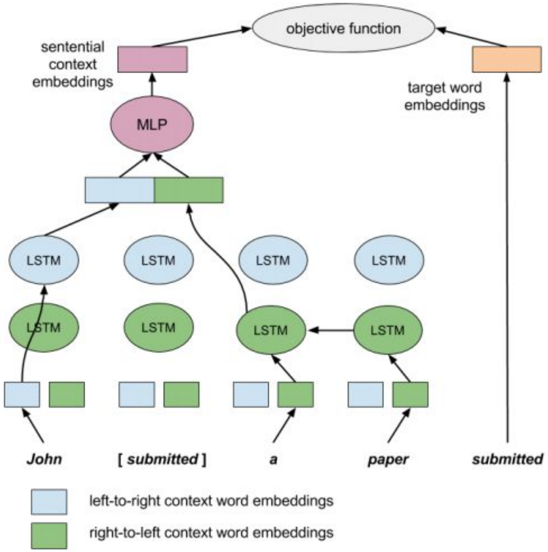
Many approaches based on language models

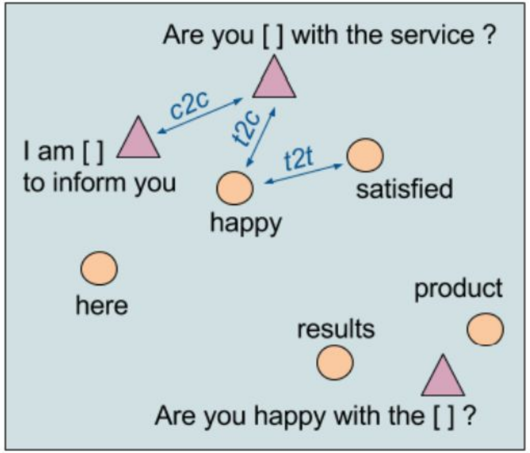
context2vec

Use bidirectional LSTM and cloze prediction objective (a 1 layer masked LM)

Learn representations for both words and contexts (minus word)

Sentence completion
Lexical substitution
WSD



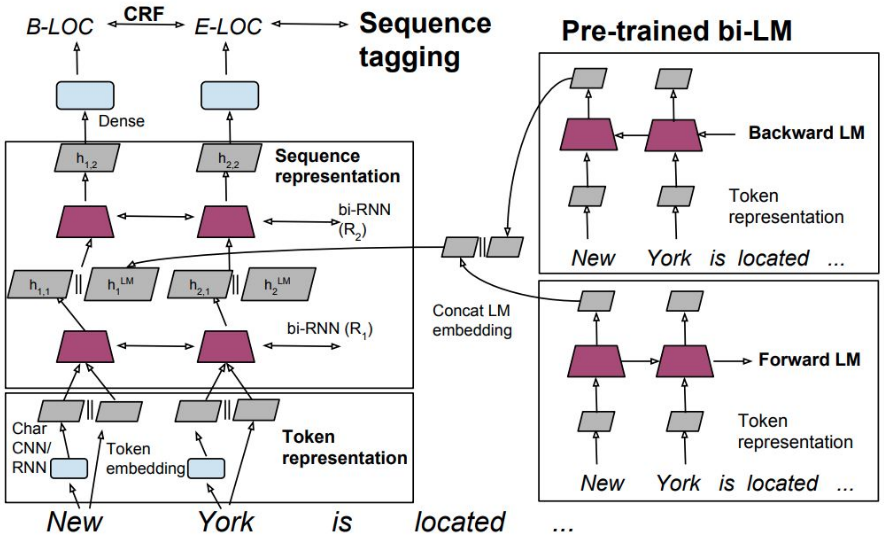


	c2v iters+	c2v	AWE	S-1	S-2
MCSS					
test	**64.0**	62.7	48.4	-	-
all	**65.1**	61.3	49.7	58.9	56.2
LST-07					
test	**56.1**	54.8	41.9	55.2	-
all	**56.0**	54.6	42.5	55.1	53.6
LST-14					
test	47.7	47.3	38.1	**50.0**	-
all	47.9	47.5	38.9	**50.2**	48.3
SE-3					
test	72.8	71.2	61.4	**74.1**	73.6

(Melamud et al., CoNLL 2016)

TagLM

Pretrain two LMs (forward and backward) and add to sequence tagger.
SOTA NER and chunking results

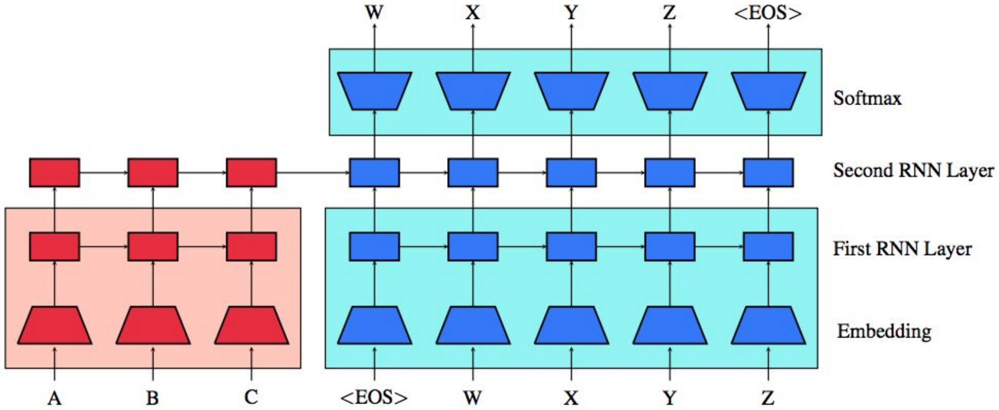


Model	$F_1 \pm$ std
Chiu and Nichols (2016)	90.91 ± 0.20
Lample et al. (2016)	90.94
Ma and Hovy (2016)	91.37
Our baseline without LM	90.87 ± 0.13
TagLM	$\mathbf{91.93 \pm 0.19}$

Table 1: Test set F_1 comparison on CoNLL 2003 NER task, using only CoNLL 2003 data and unlabeled text.

(Peters et al. ACL 2017)

Unsupervised Pretraining for Seq2Seq



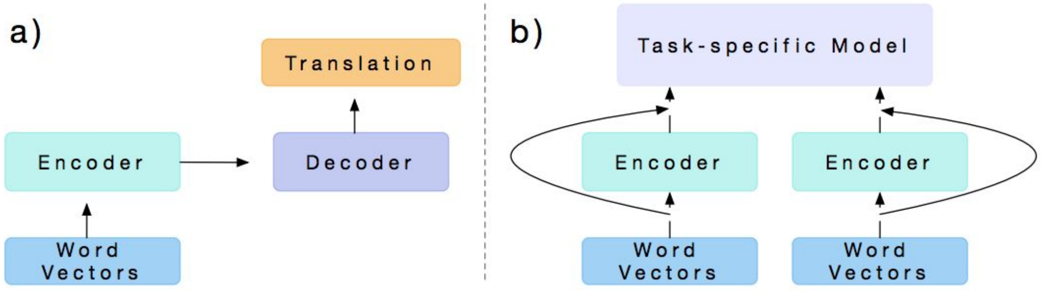
Pretrain encoder and decoder with LMs (everything shaded is pretrained).

System	ensemble?	BLEU newstest2014	BLEU newstest2015
Phrase Based MT (Williams et al., 2016)	-	21.9	23.7
Supervised NMT (Jean et al., 2015)	single	-	22.4
Edit Distance Transducer NMT (Stahlberg et al., 2016)	single	21.7	24.1
Edit Distance Transducer NMT (Stahlberg et al., 2016)	ensemble 8	22.9	25.7
Backtranslation (Sennrich et al., 2015a)	single	22.7	25.7
Backtranslation (Sennrich et al., 2015a)	ensemble 4	23.8	26.5
Backtranslation (Sennrich et al., 2015a)	ensemble 12	**24.7**	27.6
No pretraining	single	21.3	24.3
Pretrained seq2seq	single	**24.0**	**27.0**
Pretrained seq2seq	ensemble 5	**24.7**	**28.1**

Large boost for MT.

(Ramachandran et al, EMNLP 2017)

CoVe



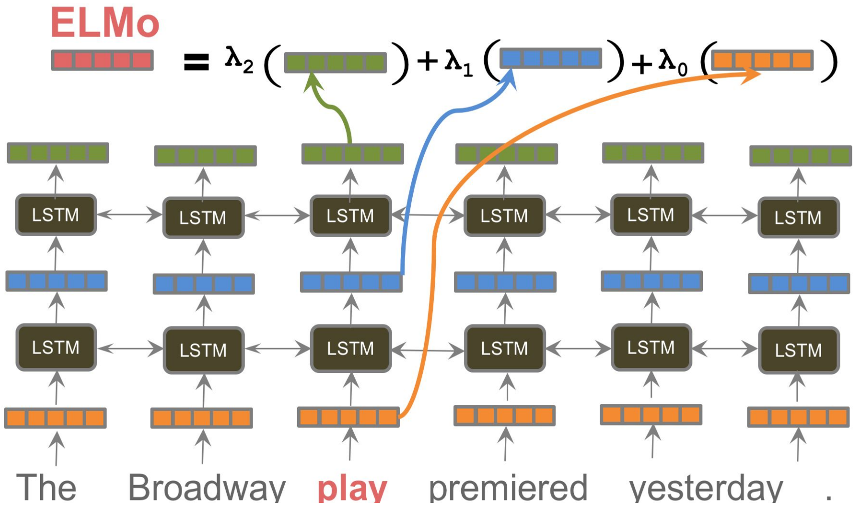
Dataset	Random	GloVe	GloVe+ Char	GloVe+ CoVe-S	GloVe+ CoVe-M	GloVe+ CoVe-L	GloVe+ Char+CoVe-L
SST-2	84.2	88.4	90.1	89.0	90.9	91.1	**91.2**
SST-5	48.6	53.5	52.2	54.0	54.7	54.5	**55.2**
IMDb	88.4	91.1	91.3	90.6	91.6	91.7	**92.1**
TREC-6	88.9	94.9	94.7	94.7	95.1	95.8	**95.8**
TREC-50	81.9	89.2	89.8	89.6	89.6	90.5	**91.2**
SNLI	82.3	87.7	87.7	87.3	87.5	87.9	**88.1**
SQuAD	65.4	76.0	78.1	76.5	77.1	79.5	**79.9**

Pretrain bidirectional encoder with MT supervision, extract LSTM states

Adding CoVe with GloVe gives improvements for classification, NLI, Q&A

(McCann et al, NeurIPS 2017)

ELMo



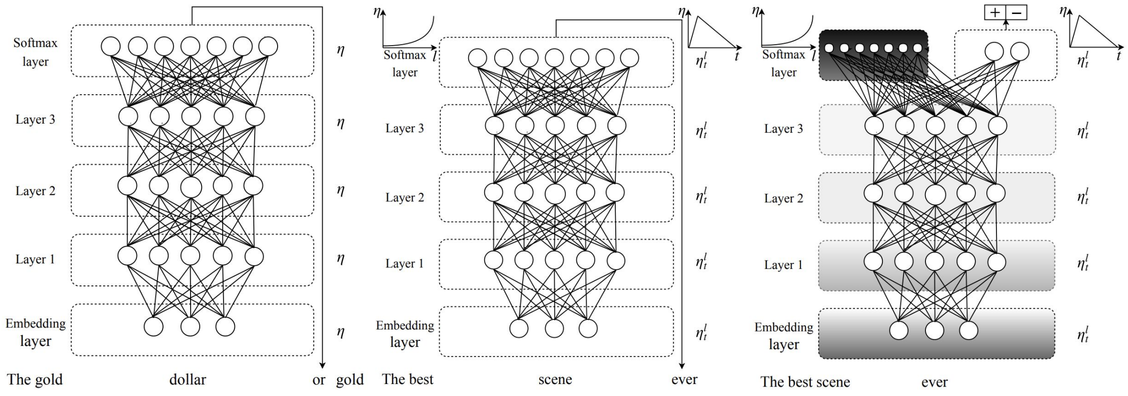
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Pretrain deep bidirectional LM, extract contextual word vectors as learned linear combination of hidden states

SOTA for 6 diverse tasks

[(Peters et al, NAACL 2018)]

ULMFiT



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

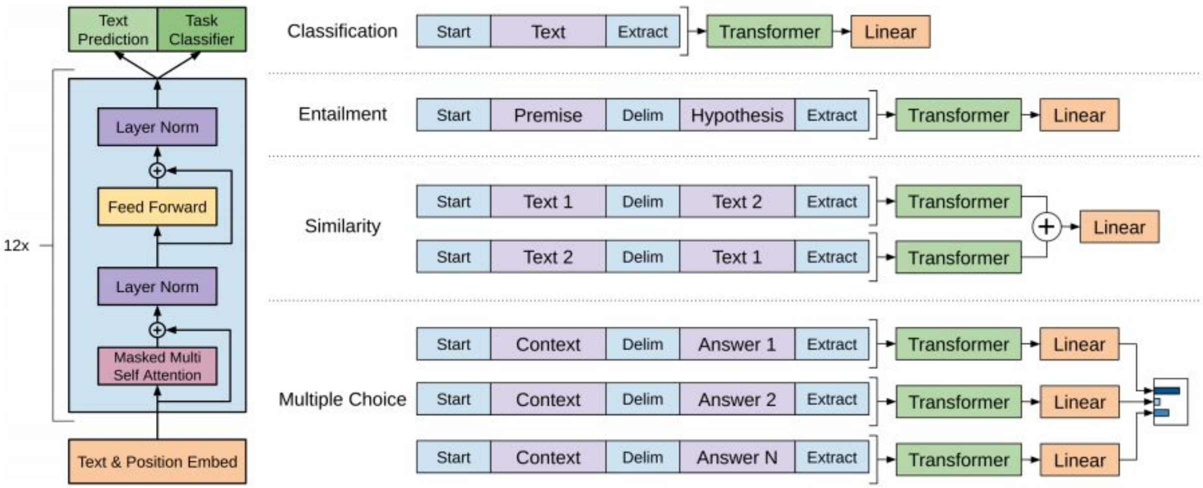
	AG	DBpedia	Yelp-bi	Yelp-full
Char-level CNN (Zhang et al., 2015)	9.51	1.55	4.88	37.95
CNN (Johnson and Zhang, 2016)	6.57	0.84	2.90	32.39
DPCNN (Johnson and Zhang, 2017)	6.87	0.88	2.64	30.58
ULMFiT (ours)	**5.01**	**0.80**	**2.16**	**29.98**

Pretrain AWD-LSTM LM,
fine-tune LM in two stages with different adaptation techniques

SOTA for six classification datasets

(Howard and Ruder, ACL 2018)

GPT



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

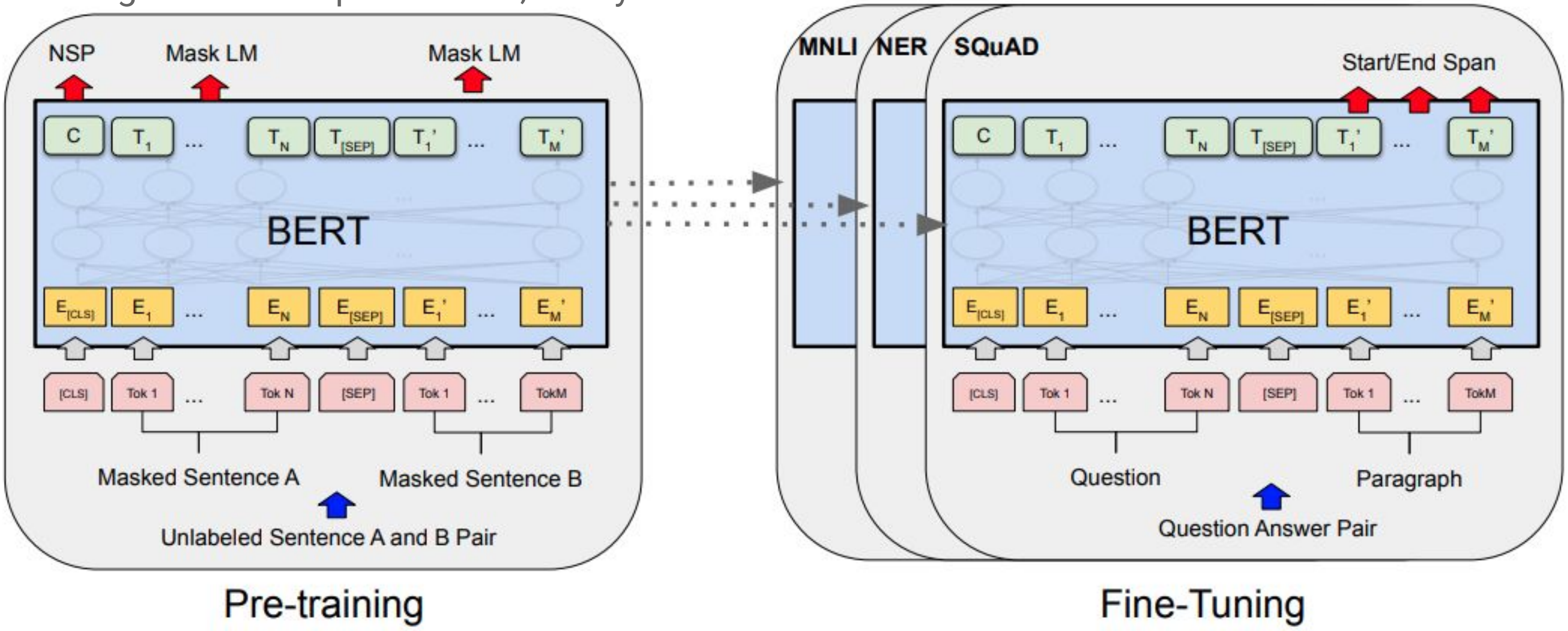
Pretrain large 12-layer left-to-right Transformer, fine tune for sentence, sentence pair and multiple choice questions.

SOTA results for 9 tasks.

(Radford et al., 2018)

BERT

BERT pretrains both sentence and contextual word representations, using masked LM and next sentence prediction.
BERT-large has 340M parameters, 24 layers!



See also: [Logeswaran and Lee, ICLR 2018]

[(Devlin et al. 2019)]

BERT

SOTA GLUE benchmark results (sentence pair classification).

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

(Devlin et al. 2019)

BERT

SOTA SQuAD v1.1 (and v2.0) Q&A

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

[(Devlin et al. 2019)]

Other pretraining objectives

- Contextual string representations (Akbik et al., COLING 2018)—SOTA NER results
- Cross-view training (Clark et al. EMNLP 2018)—improve supervised tasks with unlabeled data
- Cloze-driven pretraining (Baevski et al. (2019)—SOTA NER and constituency parsing

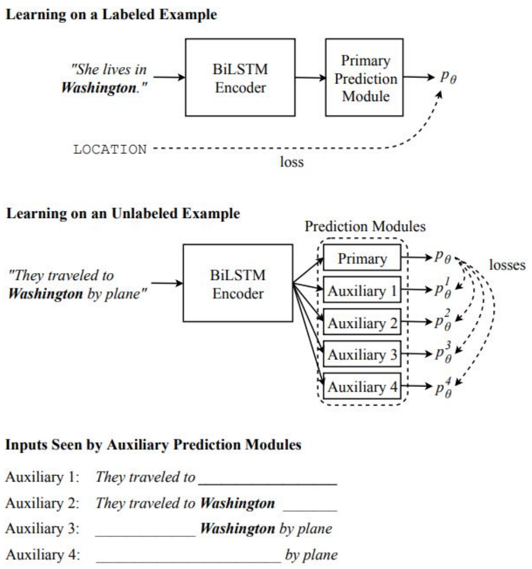


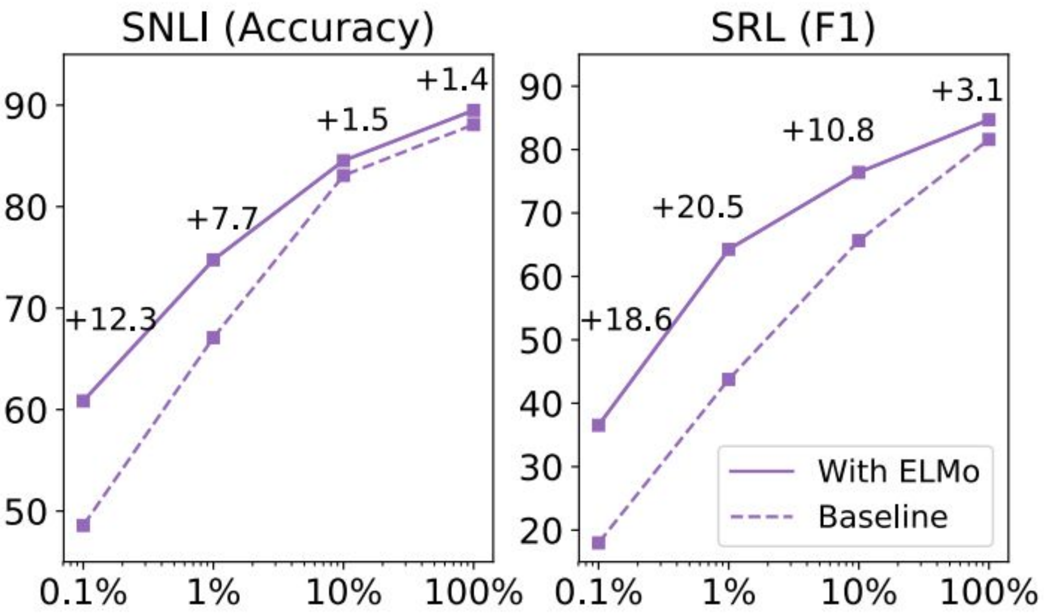
Figure 1: An overview of Cross-View Training. The model is trained with standard supervised learning on labeled examples. On unlabeled examples, auxiliary prediction modules with different views of the input are trained to agree with the primary prediction module. This particular example shows CVT applied to named entity recognition. From the labeled example, the model can learn that "Washington" usually refers to a location. Then, on unlabeled data, auxiliary prediction modules are trained to reach the same prediction without seeing some of the input. In doing so, they improve the contextual representations produced by the model, for example, learning that "traveled to" is usually followed by a location.

Why does language modeling work so well?

- Language modeling is a very difficult task, even for humans.
- Language models are expected to compress any possible context into a vector that generalizes over possible completions.
  - “They walked down the street to ???”
- To have any chance at solving this task, a model is forced to learn syntax, semantics, encode facts about the world, etc.
- Given enough data, a huge model, and enough compute, can do a reasonable job!
- Empirically works better than translation, autoencoding: “Language Modeling Teaches You More Syntax than Translation Does” (Zhang et al. 2018)

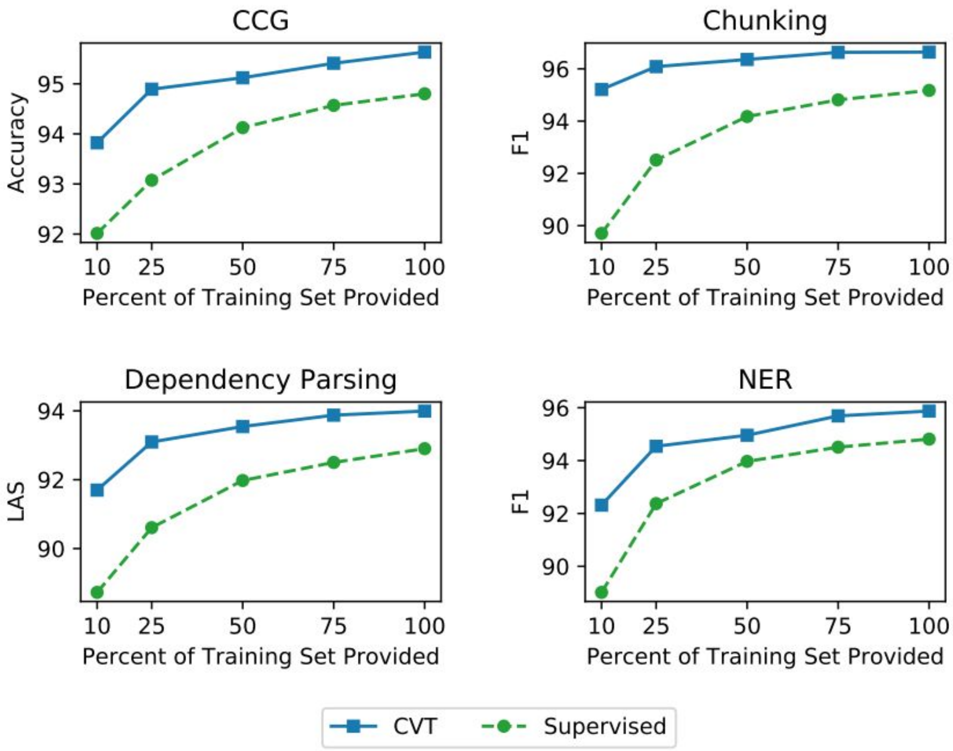
Sample efficiency

Pretraining reduces need for annotated data



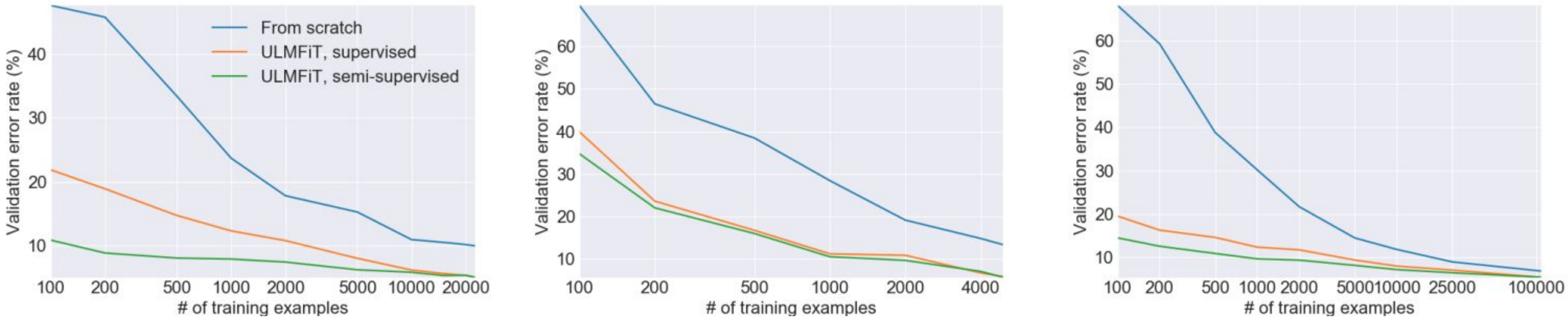
(Peters et al, NAACL 2018)

Pretraining reduces need for annotated data



(Clark et al. EMNLP 2018)

Pretraining reduces need for annotated data



(Howard and Ruder, ACL 2018)

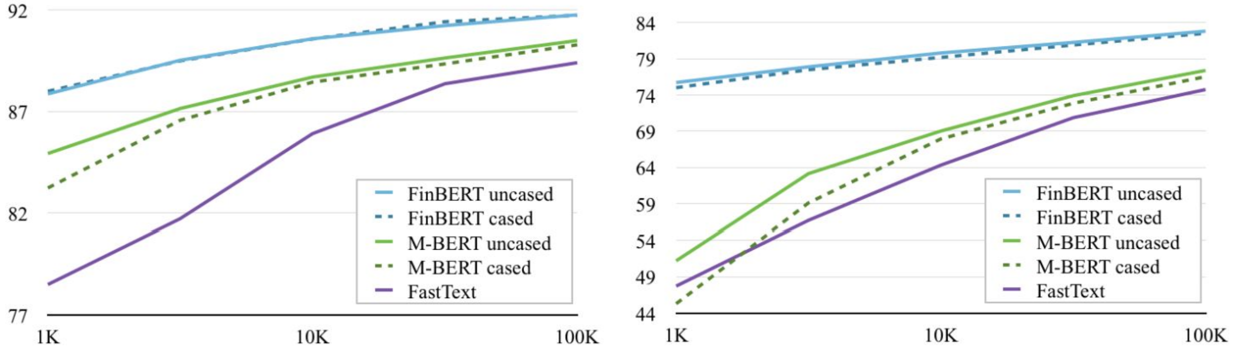
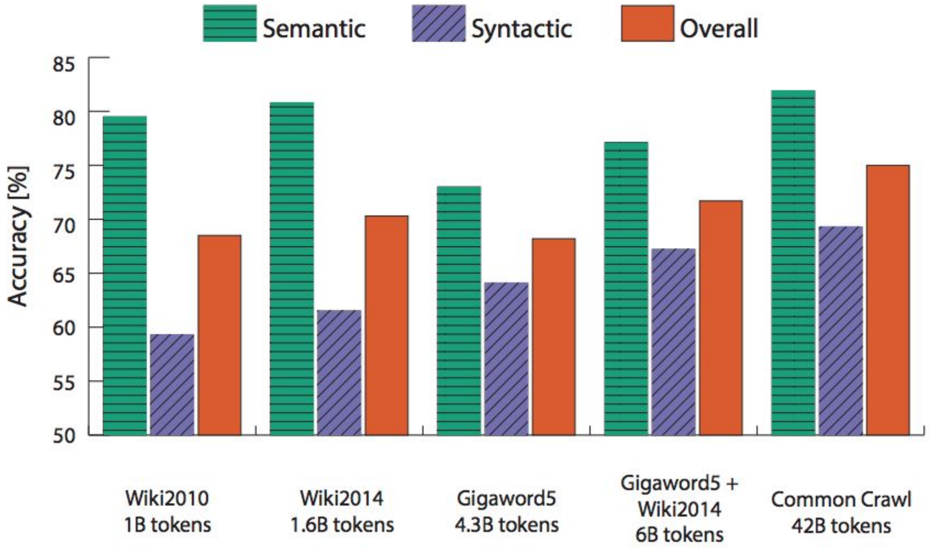


Figure 1: Text classification accuracy with different training data sizes for Yle news (left) and Ylilauta online discussion (right). (Note log x scales and different y ranges.)

Antti Virtanen et al., “Multilingual Is Not Enough: BERT for Finnish,” *ArXiv:1912.07076 [Cs]*, December 15, 2019, http://arxiv.org/abs/1912.07076.

Scaling up pretraining

Scaling up pretraining



More data → better word vectors

(Pennington et al 2014)

Scaling up pretraining

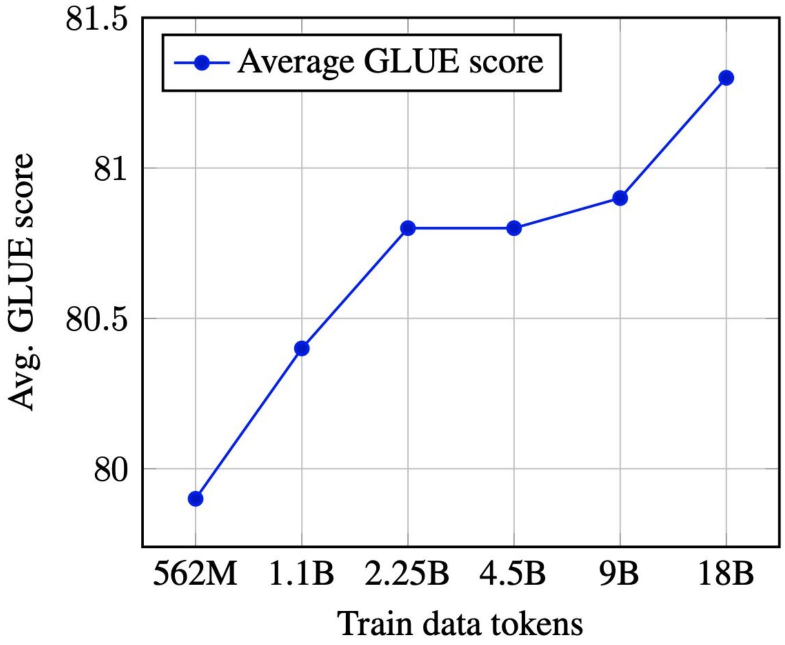


Figure 3: Average GLUE score with different amounts of Common Crawl data for pretraining.

Baevski et al. (2019)

Scaling up pretraining

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. “LM (ppl)” is the masked LM perplexity of held-out training data.

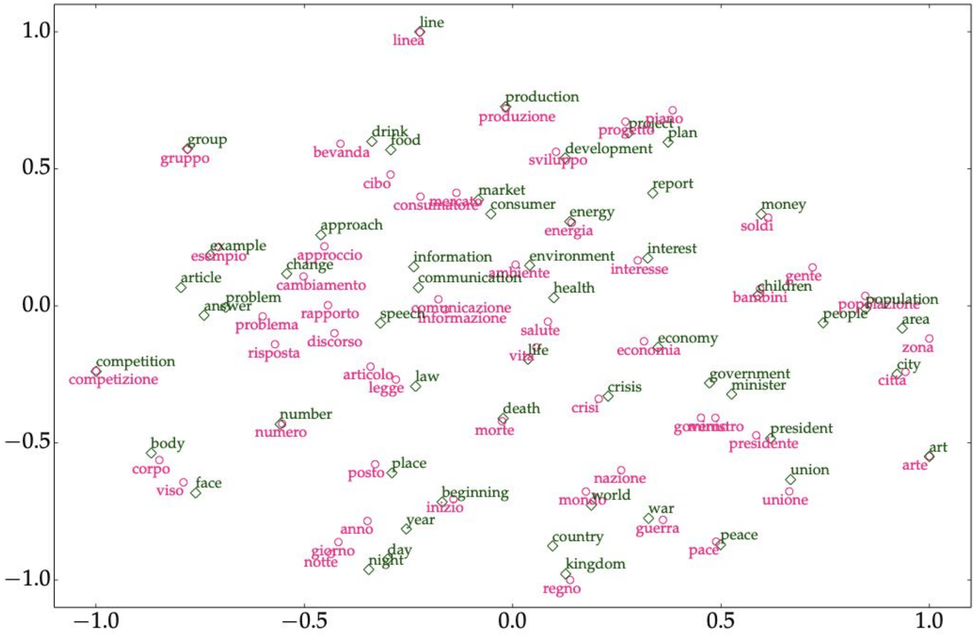
Bigger model → better results

(Devlin et al 2019)

Cross-lingual pretraining

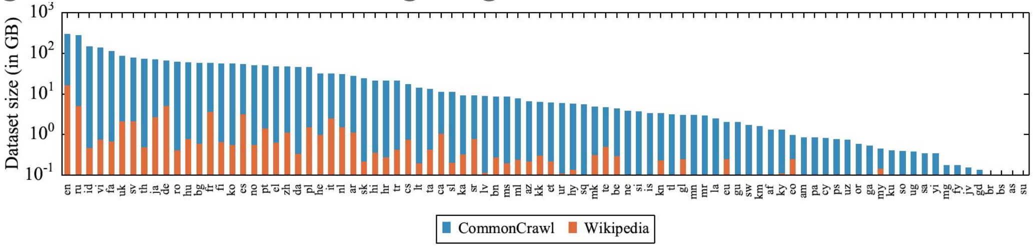
Cross-lingual pretraining

- Much work on training cross-lingual word embeddings (Overview: [Ruder et al. (2017)](#))
- Idea: train each language separately, then align.
- Recent work aligning ELMo: [Schuster et al., (NAACL 2019)](#)
- [ACL 2019 Tutorial on Unsupervised Cross-lingual Representation Learning](#)



Scaling multilingual pretraining

- Scaling to hundred of languages and **TBs** of data

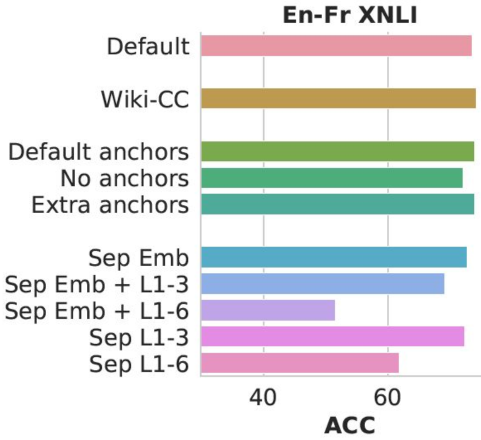


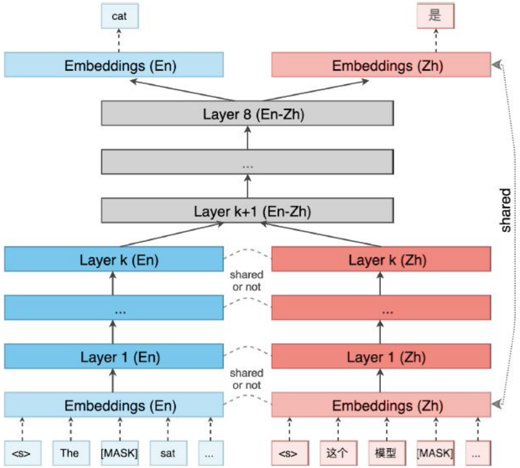
Training on 6.1T tokens (1.5M steps, BS 8k, seq length 512, model: 570M)

Alexis Conneau et al., “Unsupervised Cross-Lingual Representation Learning at Scale,” *ArXiv:1911.02116 [Cs]*, November 5, 2019, http://arxiv.org/abs/1911.02116

- Studying language-universal structures emerging in pretrained language models:
  - **Sharing parameters** is key rather than anchor points
  - **zero-shot crosslingual transfe**

Shijie Wu et al., “Emerging Cross-Lingual Structure in Pretrained Language Models,” *ArXiv:1911.01464 [Cs]*, November 10, 2019, http://arxiv.org/abs/1911.01464.





Cross-lingual Polyglot Pretraining

Key idea: **Share vocabulary** and representations across languages by training one model on many languages.

Advantages: Easy to implement, **enables** cross-lingual pretraining by itself

Disadvantages: Leads to **under-representation** of low-resource languages

- LASER: Use parallel data for sentence representations (Artetxe & Schwenk, 2018)
- Multilingual BERT: BERT trained jointly on 100 languages
- Rosita: Polyglot contextual representations (Mulcaire et al., NAACL 2019)
- XLM: Cross lingual LM (Lample & Conneau, 2019)

Hands-on #1:
Pretraining a Transformer Language Model

Image credit: Chanaky

Hands-on: Overview

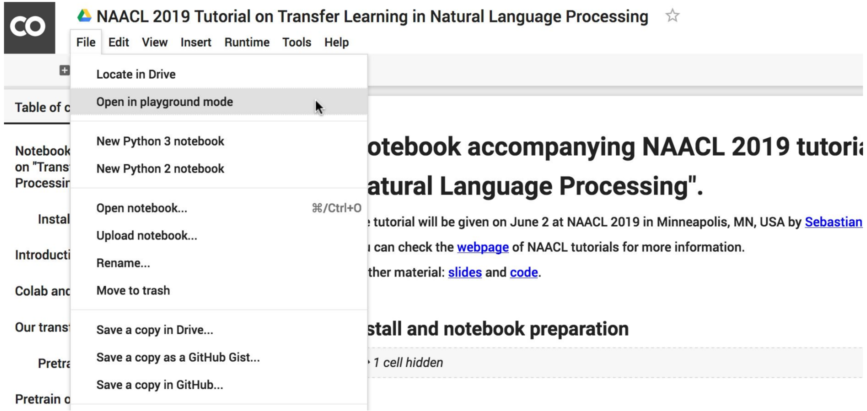
Current developments in Transfer Learning combine new approaches for training schemes (sequential training) as well as models (transformers) ➪ can look intimidating and complex

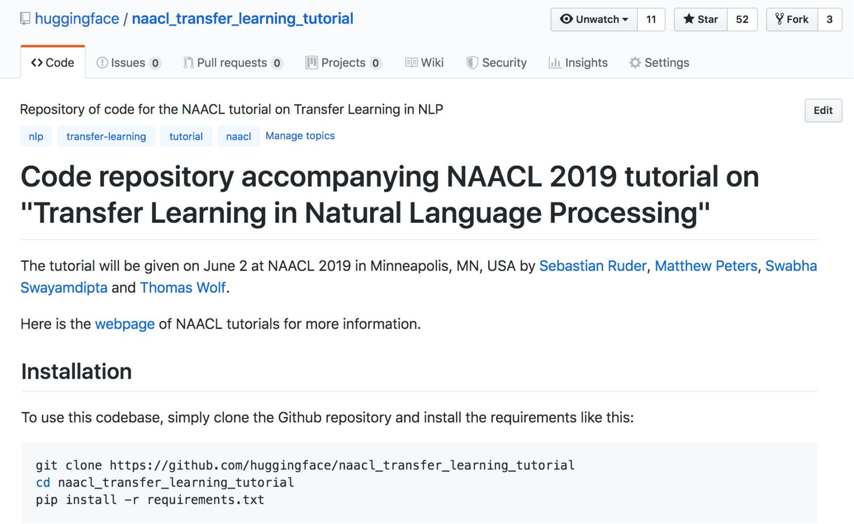
- ❏ Goals:
  - ❏ Let's make these recent works "uncool again" i.e. as accessible as possible
  - ❏ Expose all the details in a simple, concise and self-contained code-base
  - ❏ Show that transfer learning can be simple (less hand-engineering) & fast (pretrained model)
- ❏ Plan
  - ❏ Build a GPT-2 / BERT model
  - ❏ Pretrain it on a rather large corpus with ~100M words
  - ❏ Adapt it for a target task to get SOTA performances
- ❏ Material:
  - ❏ Colab: http://tiny.cc/NAACLTransferColab ➪ code of the following slides
  - ❏ Code: http://tiny.cc/NAACLTransferCode ➪ same code organized in a repo

Hands-on pre-training

Colab: https://tinyurl.com/NAACLTransferColab

Repo: https://tinyurl.com/NAACLTransferCode



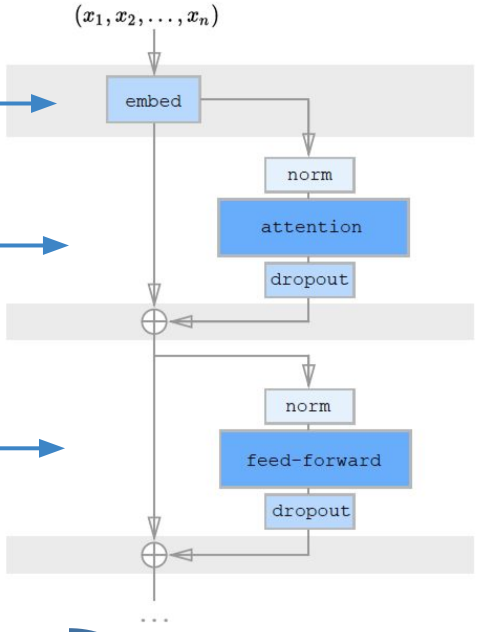


Hands-on pre-training

Our core model will be a Transformer. Large-scale transformer architectures (GPT-2, BERT, XLM…) are very similar to each other and consist of:

- ❏ summing words and position embeddings
- ❏ applying a succession of transformer blocks with:
  - ❏ layer normalisation
  - ❏ a self-attention module
  - ❏ dropout and a residual connection

  - ❏ another layer normalisation
  - ❏ a feed-forward module with one hidden layer and a non linearity: Linear ➪ ReLU/gelu ➪ Linear
  - ❏ dropout and a residual connection



Main differences between GPT/GPT-2/BERT are the objective functions:

- ❏ causal language modeling for GPT
- ❏ masked language modeling for BERT (+ next sentence prediction)

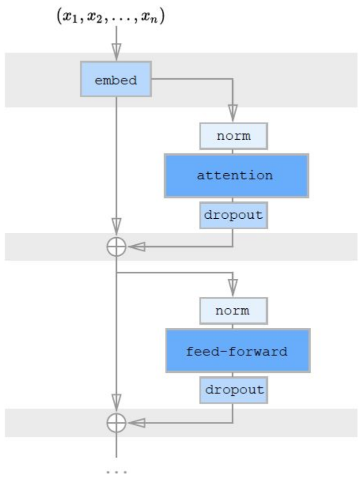
We'll play with both

(Child et al, 2019)

Hands-on pre-training

Let's code the backbone of our model!

PyTorch 1.1 now has a *nn.MultiHeadAttention* module: lets us encapsulate the self-attention logic while still controlling the internals of the Transformer.




```python
import torch
import torch.nn as nn

class Transformer(nn.Module):
    def __init__(self, embed_dim, hidden_dim, num_embeddings, num_max_positions, num_heads, num_layers, dropout, causal):
        super().__init__()
        self.causal = causal
        self.tokens_embeddings = nn.Embedding(num_embeddings, embed_dim)
        self.position_embeddings = nn.Embedding(num_max_positions, embed_dim)
        self.dropout = nn.Dropout(dropout)

        self.attentions, self.feed_forwards = nn.ModuleList(), nn.ModuleList()
        self.layer_norms_1, self.layer_norms_2 = nn.ModuleList(), nn.ModuleList()
        for _ in range(num_layers):
            self.attentions.append(nn.MultiheadAttention(embed_dim, num_heads, dropout=dropout))
            self.feed_forwards.append(nn.Sequential(nn.Linear(embed_dim, hidden_dim),
                                                    nn.ReLU(),
                                                    nn.Linear(hidden_dim, embed_dim)))
            self.layer_norms_1.append(nn.LayerNorm(embed_dim, eps=1e-12))
            self.layer_norms_2.append(nn.LayerNorm(embed_dim, eps=1e-12))

    def forward(self, x, padding_mask=None):
        positions = torch.arange(len(x), device=x.device).unsqueeze(-1)
        h = self.tokens_embeddings(x)
        h = h + self.position_embeddings(positions).expand_as(h)
        h = self.dropout(h)

        attn_mask = None
        if self.causal:
            attn_mask = torch.full((len(x), len(x)), -float('Inf'), device=h.device, dtype=h.dtype)
            attn_mask = torch.triu(attn_mask, diagonal=1)

        for layer_norm_1, attention, layer_norm_2, feed_forward in zip(self.layer_norms_1, self.attentions,
                                                                       self.layer_norms_2, self.feed_forwards):
            h = layer_norm_1(h)
            x, _ = attention(h, h, h, attn_mask=attn_mask, need_weights=False, key_padding_mask=padding_mask)
            x = self.dropout(x)
            h = x + h

            h = layer_norm_2(h)
            x = feed_forward(h)
            x = self.dropout(x)
            h = x + h
        return h
```

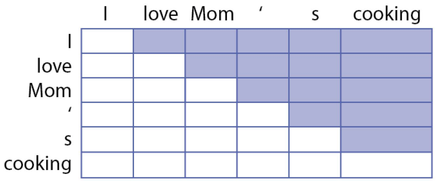
Hands-on pre-training

Two attention masks?

- ❑ `padding_mask` masks the padding tokens. It is specific to each sample in the batch:

I	love	Mom	'	s	cooking
I	love	you	too	!	
No	way				
This	is	the	shit		
Yes					

- ❑ `attn_mask` is the same for all samples in the batch. It masks the previous tokens for causal transformers:




```python
import torch
import torch.nn as nn

class Transformer(nn.Module):
    def __init__(self, embed_dim, hidden_dim, num_embeddings, num_max_positions, num_heads, num_layers, dropout, causal):
        super().__init__()
        self.causal = causal
        self.tokens_embeddings = nn.Embedding(num_embeddings, embed_dim)
        self.position_embeddings = nn.Embedding(num_max_positions, embed_dim)
        self.dropout = nn.Dropout(dropout)

        self.attentions, self.feed_forwards = nn.ModuleList(), nn.ModuleList()
        self.layer_norms_1, self.layer_norms_2 = nn.ModuleList(), nn.ModuleList()
        for _ in range(num_layers):
            self.attentions.append(nn.MultiheadAttention(embed_dim, num_heads, dropout=dropout))
            self.feed_forwards.append(nn.Sequential(nn.Linear(embed_dim, hidden_dim),
                                                    nn.ReLU(),
                                                    nn.Linear(hidden_dim, embed_dim)))
            self.layer_norms_1.append(nn.LayerNorm(embed_dim, eps=1e-12))
            self.layer_norms_2.append(nn.LayerNorm(embed_dim, eps=1e-12))

    def forward(self, x, padding_mask=None):

        positions = torch.arange(len(x), device=x.device).unsqueeze(-1)
        h = self.tokens_embeddings(x)
        h = h + self.position_embeddings(positions).expand_as(h)
        h = self.dropout(h)

        attn_mask = None
        if self.causal:
            attn_mask = torch.full((len(x), len(x)), -float('Inf'), device=h.device, dtype=h.dtype)
            attn_mask = torch.triu(attn_mask, diagonal=1)

        for layer_norm_1, attention, layer_norm_2, feed_forward in zip(self.layer_norms_1, self.attentions,
                                                                       self.layer_norms_2, self.feed_forwards):

            h = layer_norm_1(h)
            x, _ = attention(h, h, h, attn_mask=attn_mask, need_weights=False, key_padding_mask=padding_mask)
            x = self.dropout(x)
            h = x + h

            h = layer_norm_2(h)
            x = feed_forward(h)
            x = self.dropout(x)
            h = x + h

        return h
```

Hands-on pre-training

To pretrain our model, we need to add a few elements: a head, a loss and initialize weights.

We add these elements with a pretraining model encapsulating our model.

1. **A pretraining head** on top of our core model: we choose a language modeling head with tied weights

2. **Initialize** the weights

3. Define a **loss function**: we choose a cross-entropy loss on current (or next) token predictions

```
class TransformerWithLMHead(nn.Module):
    def __init__(self, config):
        """ Transformer with a language modeling head on top (tied weights) """
        super().__init__()
        self.config = config
        self.transformer = Transformer(config.embed_dim, config.hidden_dim, config.num_embeddings,
                                       config.num_max_positions, config.num_heads, config.num_layers,
                                       config.dropout, causal=not config.mlm)

        self.lm_head = nn.Linear(config.embed_dim, config.num_embeddings, bias=False)
        self.apply(self.init_weights)
        self.tie_weights()

    def tie_weights(self):
        self.lm_head.weight = self.transformer.tokens_embeddings.weight

    def init_weights(self, module):
        """ initialize weights - nn.MultiheadAttention is already initalized by PyTorch (xavier) """
        if isinstance(module, (nn.Linear, nn.Embedding, nn.LayerNorm)):
            module.weight.data.normal_(mean=0.0, std=self.config.initializer_range)
        if isinstance(module, (nn.Linear, nn.LayerNorm)) and module.bias is not None:
            module.bias.data.zero_()

    def forward(self, x, labels=None, padding_mask=None):
        """ x has shape [seq length, batch], padding_mask has shape [batch, seq length] """
        hidden_states = self.transformer(x, padding_mask)
        logits = self.lm_head(hidden_states)

        if labels is not None:
            shift_logits = logits[:-1] if self.transformer.causal else logits
            shift_labels = labels[1:] if self.transformer.causal else labels
            loss_fct = nn.CrossEntropyLoss(ignore_index=-1)
            loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1))
            return logits, loss

        return logits
```

Hands-on pre-training

Now let's take care of our data and configuration

We'll use a pre-defined open vocabulary tokenizer: BERT's model cased tokenizer.

```
from pytorch_pretrained_bert import BertTokenizer, cached_path

tokenizer = BertTokenizer.from_pretrained('bert-base-cased', do_lower_case=False)
```

Hyper-parameters taken from Dai et al., 2018 (Transformer-XL) ➪ ~50M parameters causal model.

```
from collections import namedtuple

Config = namedtuple('Config',
  field_names="embed_dim, hidden_dim, num_max_positions, num_embeddings      , num_heads, num_layers,"
              "dropout, initializer_range, batch_size, lr, max_norm, n_epochs, n_warmup,"
              "mlm, gradient_accumulation_steps, device, log_dir, dataset_cache")
args = Config( 410      , 2100      , 256               , len(tokenizer.vocab), 10       , 16        ,
               0.1     , 0.02              , 16         , 2.5e-4, 1.0 , 50      , 1000     ,
               False, 4, "cuda" if torch.cuda.is_available() else "cpu", "./"    , "./dataset_cache.bin")
```

Use a large dataset for pre-trainining: WikiText-103 with 103M tokens (Merity et al., 2017).

```
dataset_file = cached_path("https://s3.amazonaws.com/datasets.huggingface.co/wikitext-103/"
                           "wikitext-103-train-tokenized-bert.bin")
datasets = torch.load(dataset_file)

# Convert our encoded dataset to torch.tensors and reshape in blocks of the transformer's input length
for split_name in ['train', 'valid']:
    tensor = torch.tensor(datasets[split_name], dtype=torch.long)
    num_sequences = (tensor.size(0) // args.num_max_positions) * args.num_max_positions
    datasets[split_name] = tensor.narrow(0, 0, num_sequences).view(-1, args.num_max_positions)
```

Instantiate our model and optimizer (Adam)

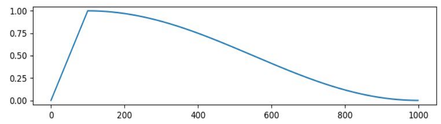
```
model = TransformerWithLMHead(args).to(args.device)
optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
```

Hands-on pre-training

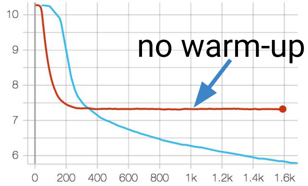
And we're done: let's train!

A simple update loop.
We use gradient accumulation to have a large batch … GPU (>64)

Learning r…
- linear wa…
- then cos… square root decrease



training/loss




```python
import os
from torch.utils.data import DataLoader
from ignite.engine import Engine, Events
from ignite.metrics import RunningAverage
from ignite.handlers import ModelCheckpoint
from ignite.contrib.handlers import CosineAnnealingScheduler, create_lr_scheduler_with_warmup, ProgressBar

dataloader = DataLoader(datasets['train'], batch_size=args.batch_size, shuffle=True)

# Define training function
                                                                  [seq length, batch]
```

```python
for batch in self.state.dataloader:
    self.state.batch = batch
    self.state.iteration += 1
    self._fire_event(Events.ITERATION_STARTED)
    self.state.output = self._process_function(self, batch)
    self._fire_event(Events.ITERATION_COMPLETED)
```

```python
RunningAverage(output_transform=lambda x: x).attach(trainer, "loss")
ProgressBar(persist=True).attach(trainer, metric_names=['loss'])

# Learning rate schedule: linearly warm-up to lr and then decrease the learning rate to zero with cosine
cos_scheduler = CosineAnnealingScheduler(optimizer, 'lr', args.lr, 0.0, len(dataloader) * args.n_epochs)
scheduler = create_lr_scheduler_with_warmup(cos_scheduler, 0.0, args.lr, args.n_warmup)
trainer.add_event_handler(Events.ITERATION_STARTED, scheduler)

# Save checkpoints and training config
checkpoint_handler = ModelCheckpoint(args.log_dir, 'checkpoint', save_interval=1, n_saved=5)
trainer.add_event_handler(Events.EPOCH_COMPLETED, checkpoint_handler, {'mymodel': model})
torch.save(args, os.path.join(args.log_dir, 'training_args.bin'))
```

Go!

```python
trainer.run(train_dataloader, max_epochs=args.n_epochs)
```

Epoch [1/50] [365/28874] 1%| , loss=2.30e+00 [03:43<4:52:22]

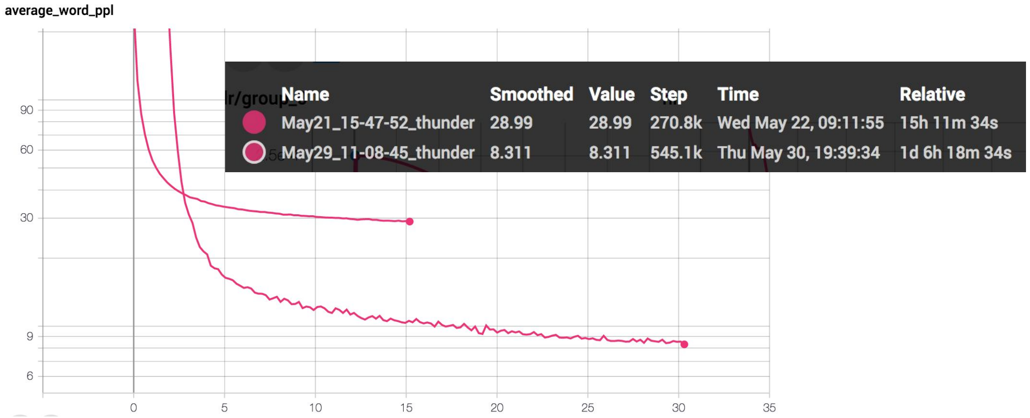
Hands-on pre-training — Concluding remarks

- On pretraining
  - **Intensive**: in our case 5h–20h on 8 V100 GPUs (few days w. 1 V100) to reach a good perplexity ➪ share your pretrained models
  - **Robust to the choice of hyper-parameters** (apart from needing a warm-up for transformers)
  - Language modeling is a hard task, your model should **not have enough capacity to overfit** if your dataset is large enough ➪ you can just start the training and let it run.
  - **Masked-language modeling**: typically 2-4 times slower to train than LM
    We only mask 15% of the tokens ➪ smaller signal

- For the rest of this tutorial
  We don't have enough time to do a full pretraining
  ➪ we pretrained **two models** for you before the tutorial

Hands-on pre-training — Concluding remarks

- First model:
  - **exactly the one** we built together ➪ a 50M parameters causal Transformer
  - Trained 15h on 8 V100
  - Reached a **word-level perplexity of 29** on wikitext-103 validation set (quite competitive)
- Second model:
  - Same model but trained with a **masked-language modeling** objective (see the repo)
  - Trained 30h on 8 V100
  - Reached a “masked-word” perplexity of 8.3 on wikitext-103 validation set

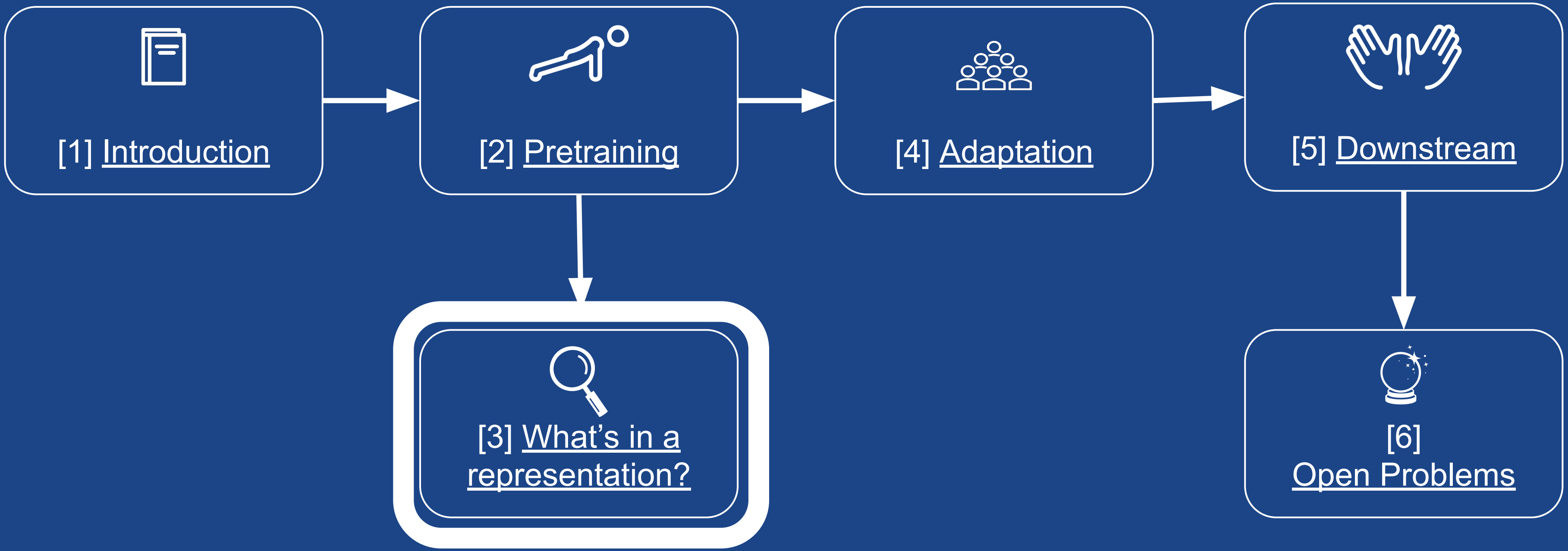


Model	#Params	Validation PPL	Test PPL
Grave et al. (2016b) – LSTM	-	-	48.7
Bai et al. (2018) – TCN	-	-	45.2
Dauphin et al. (2016) – GCNN-8	-	-	44.9
Grave et al. (2016b) – LSTM + Neural cache	-	-	40.8
Dauphin et al. (2016) – GCNN-14	-	-	37.2
Merity et al. (2018) – 4-layer QRNN	151M	32.0	33.0
Rae et al. (2018) – LSTM + Hebbian + Cache	-	29.7	29.9
Ours – Transformer-XL Standard	151M	**23.1**	**24.0**
Baevski & Auli (2018) – adaptive input◇	247M	19.8	20.5
Ours – Transformer-XL Large	257M	**17.7**	**18.3**

Wikitext-103 Validation/Test PPL

Dai et al., 2018

Agenda


[1] Introduction
[2] Pretraining
[4] Adaptation
[5] Downstream
[3] What's in a representation?
[6] Open Problems

3. What is in a Representation?

Image credit: Caique Lima

Why care about what is in a representation?

- ❏ Extrinsic evaluation with downstream tasks
  - ❏ Complex, diverse with task-specific quirks

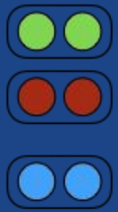
- ❏ Language-aware representations
  - ❏ To generalize to other tasks, new inputs
  - ❏ As intermediates for possible improvements to pretraining

- ❏ Interpretability!
  - ❏ Are we getting our results because of the right reasons?
  - ❏ Uncovering biases...

[Swayamdipta, 2019]

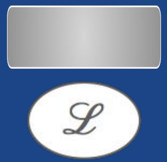
What to analyze?

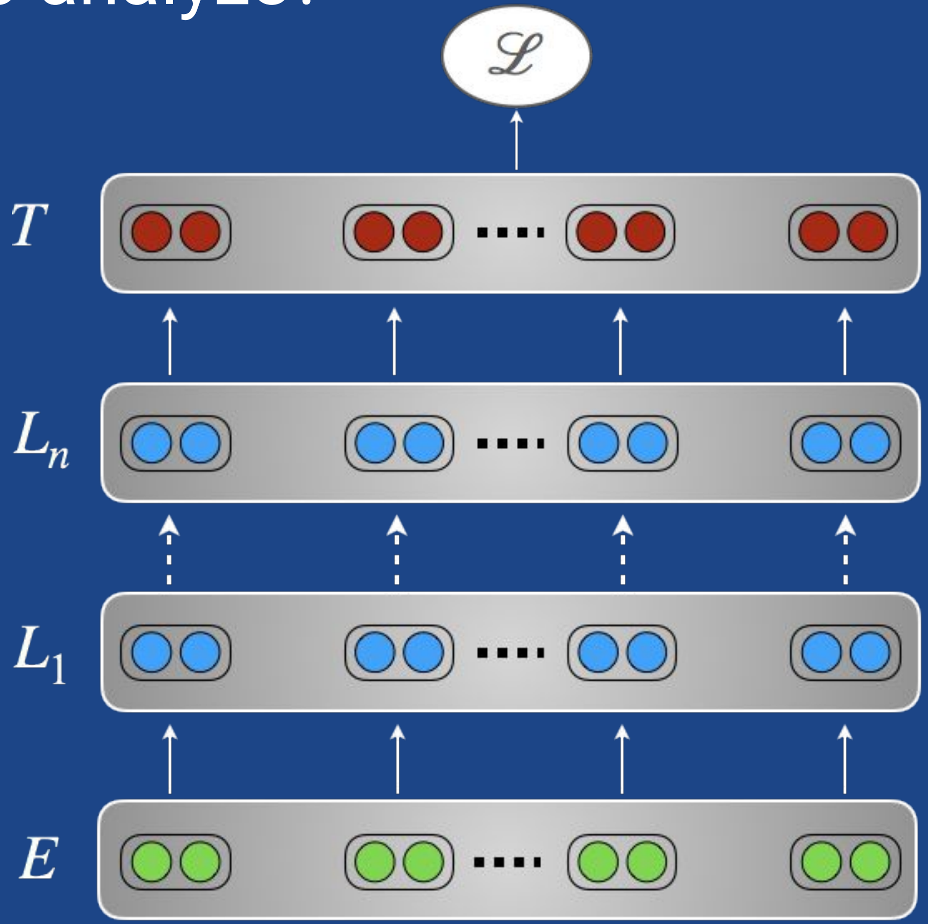
- Embeddings
  - Word
  - Contextualized



- Network Activations

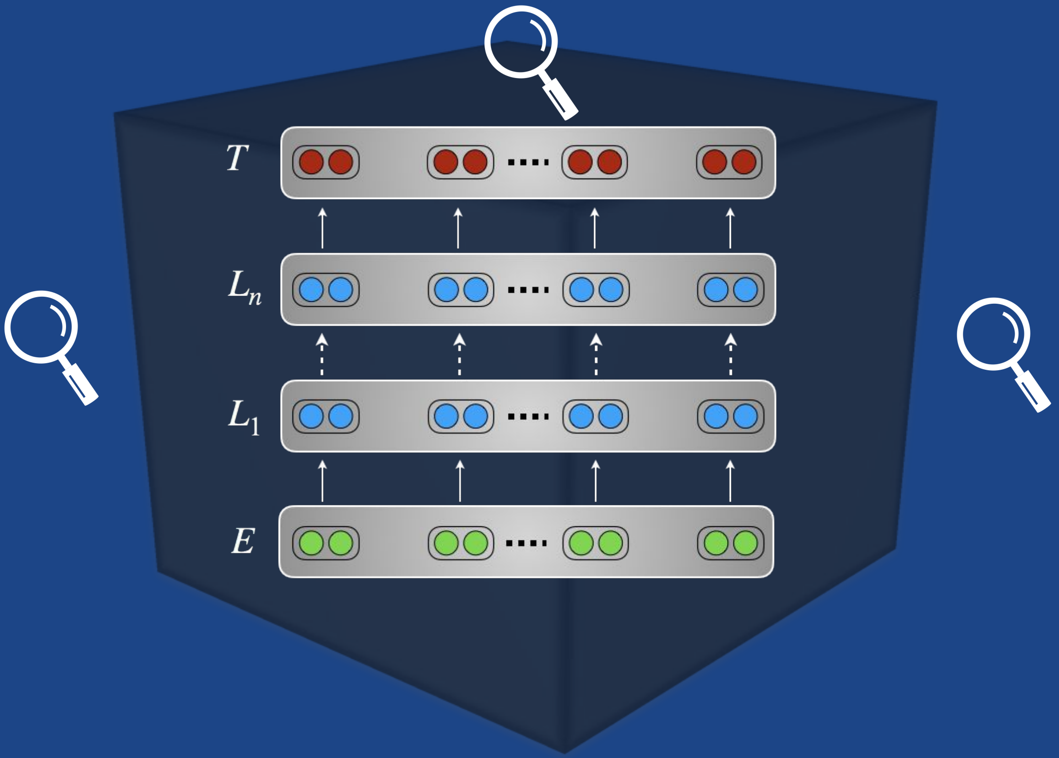
- Variations
  - Architecture (RNN / Transformer)
  - Layers
  - Pretraining Objectives





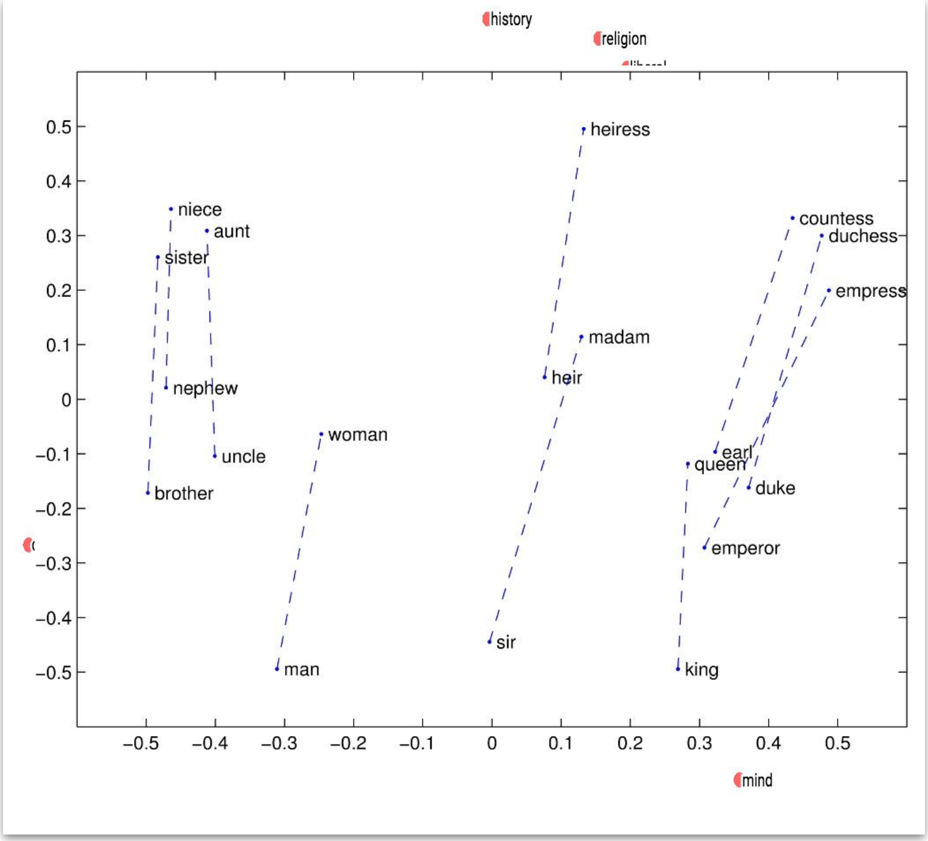
Analysis Method 1: Visualization

Hold the embeddings / network activations static or **frozen**



Visualizing Embedding Geometries

- Plotting embeddings in a lower dimensional (2D/3D) space
  - t-SNE van der Maaten & Hinton, 2008
  - PCA projections
- Visualizing word analogies Mikolov et al. 2013
  - Spatial relations
  - $w_{king} - w_{man} + w_{woman} \sim w_{queen}$
- High-level view of lexical semantics
  - Only a limited number of examples
  - Connection to other tasks is unclear Goldberg, 2017



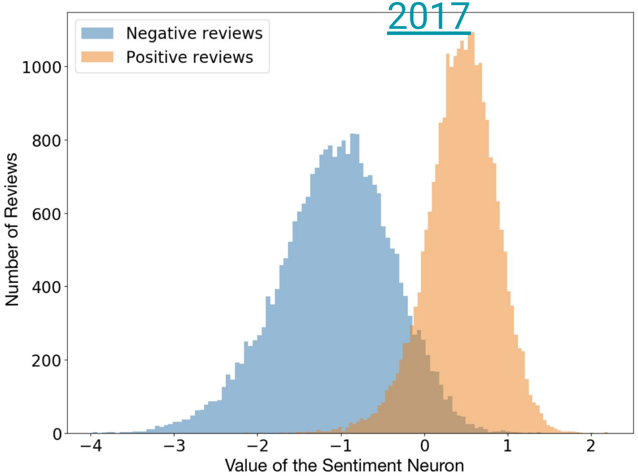
Pennington et al., 2014

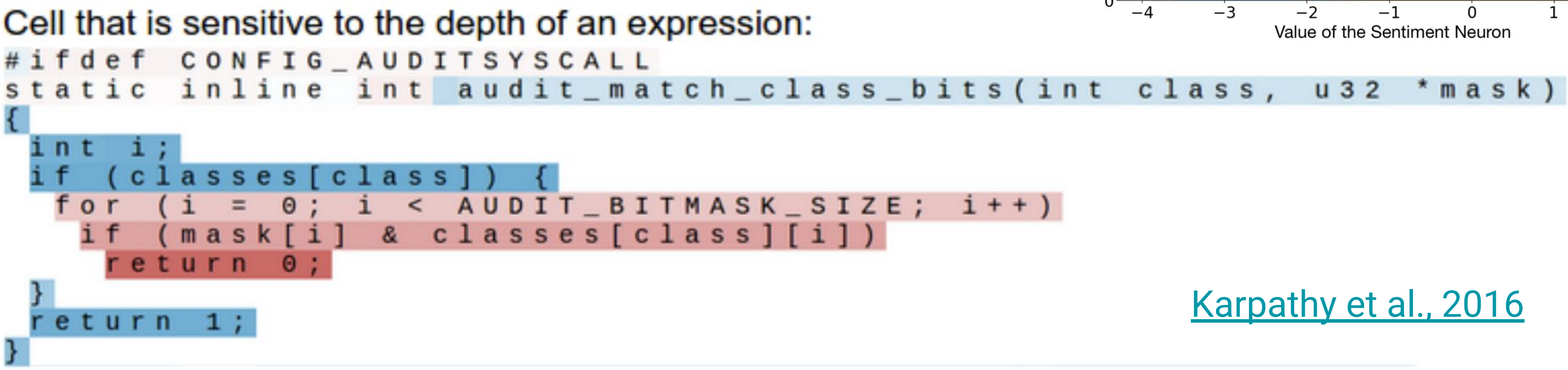
Image: Tensorflow

Visualizing Neuron Activations

- Neuron activation values correlate with features / labels
- Indicates learning of recognizable features
  - How to select which neuron? Hard to scale!
  - Interpretable != Important (Morcos et al., 2018)

[Radford et al., 2017](#)

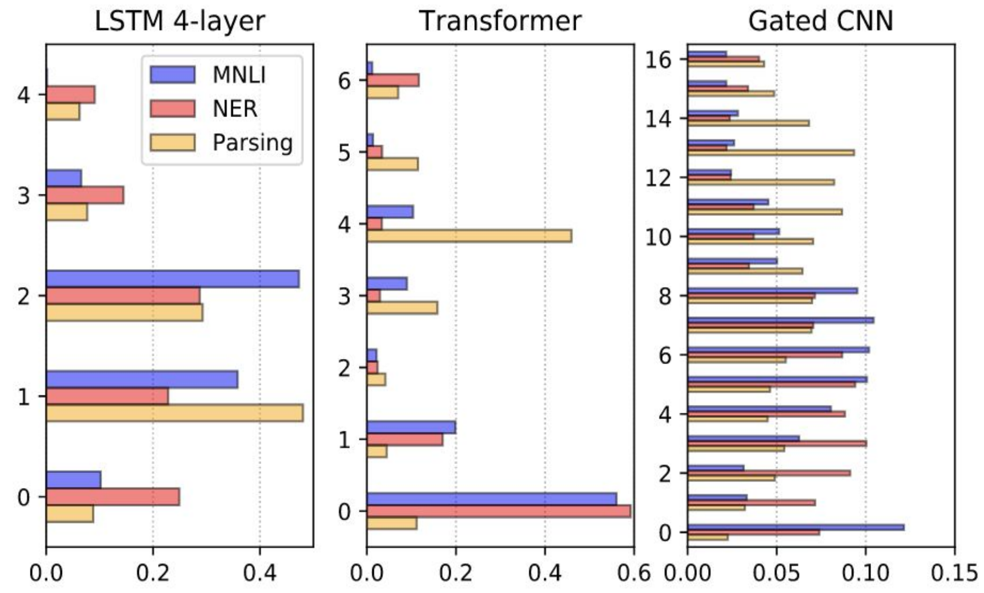




Karpathy et al., 2016

Visualizing Layer-Importance Weights

- How important is each layer for a **given performance** on a downstream task?
  - Weighted average of layers



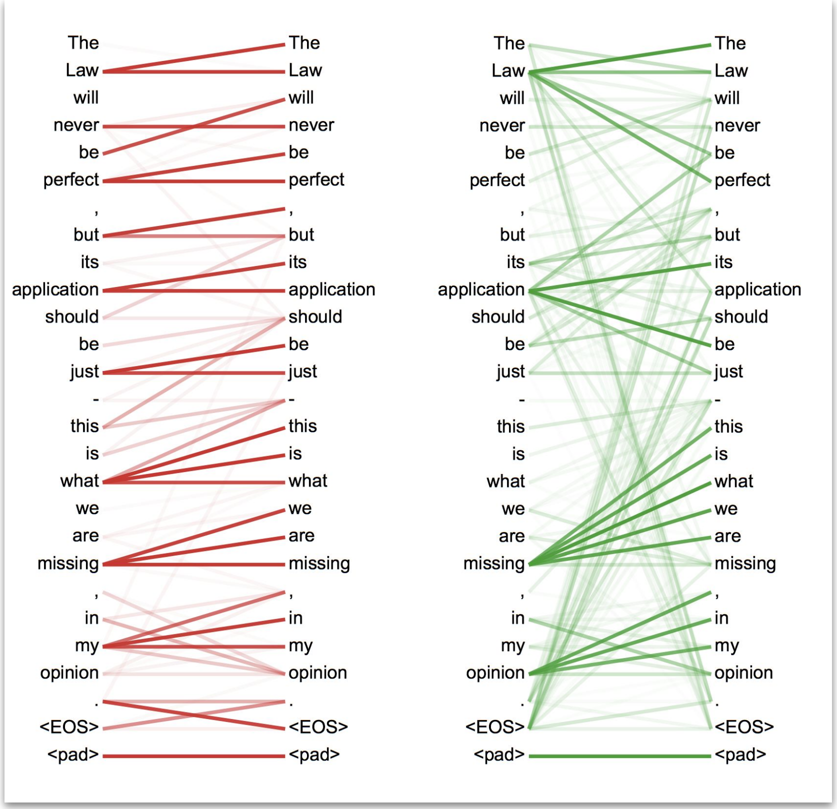
- Task and architecture specific!

Also see [Tenney et al., ACL 2019]()

[Peters et al.. EMNLP 2018]()

Visualizing Attention Weights

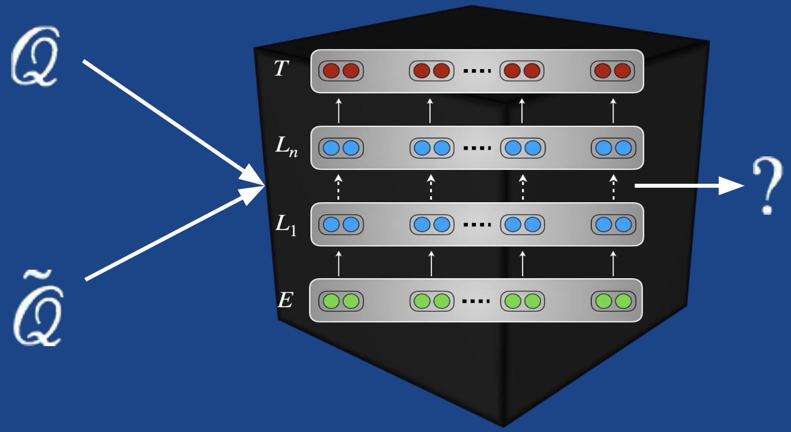
- Popular in machine translation, or other seq2seq architectures:
  - **Alignment** between words of source and target.
  - Long-distance word-word **dependencies** (intra-sentence attention)
- Sheds light on architectures
  - Having sophisticated attention mechanisms can be a good thing!
  - Layer-specific
- Interpretation can be tricky
  - Few examples only - cherry picking?
  - Robust **corpus-wide** trends? Next!



Vaswani et al., 2017

Analysis Method 2: Behavioral Probes

- RNN-based language models
  - **number agreement** in subject-verb dependencies
  - natural and nonce or ungrammatical sentences
  - evaluate on output perplexity



- RNNs outperform other non-neural baselines.

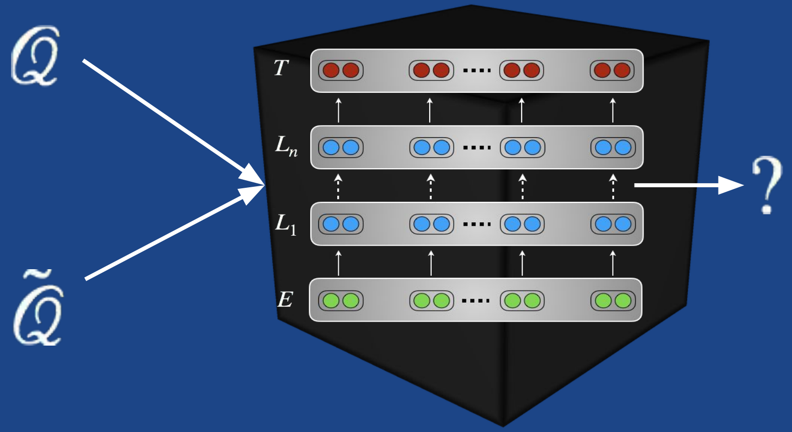
- Performance improves when trained explicitly with syntax (Kuncoro et al. 2018)

Kuncoro et al. 2018

Linzen et al., 2016; Gulordava et al. 2018; Marvin et al., 2018

Analysis Method 2: Behavioral Probes

- RNN-based language models (RNN-based)
  - **number agreement** in subject-verb dependencies
  - For natural and nonce/ungrammatical sentences
  - LM perplexity differences



- RNNs outperform other non-neural baselines.

- Performance improves when trained explicitly with syntax (Kuncoro et al. 2018)

- Probe: Might be vulnerable to co-occurrence biases
  - "dogs in the neighborhood bark(s)"
  - Nonce sentences might be too different from original...

Kuncoro et al. 2018

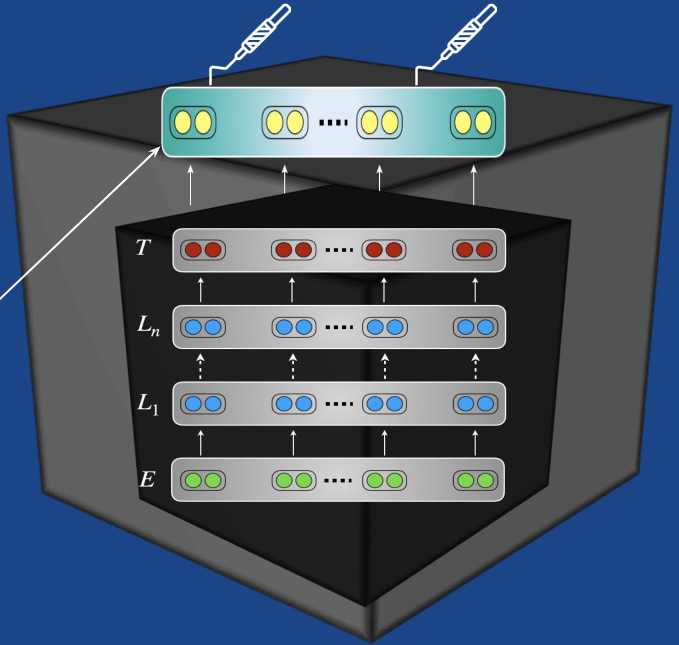
Linzen et al., 2016; Gulordava et al. 2018; Marvin et al., 2018

Analysis Method 3: Classifier Probes

Hold the embeddings / network activations static and

train a **simple supervised** model on top

Probe classification task
(Linear / MLP)



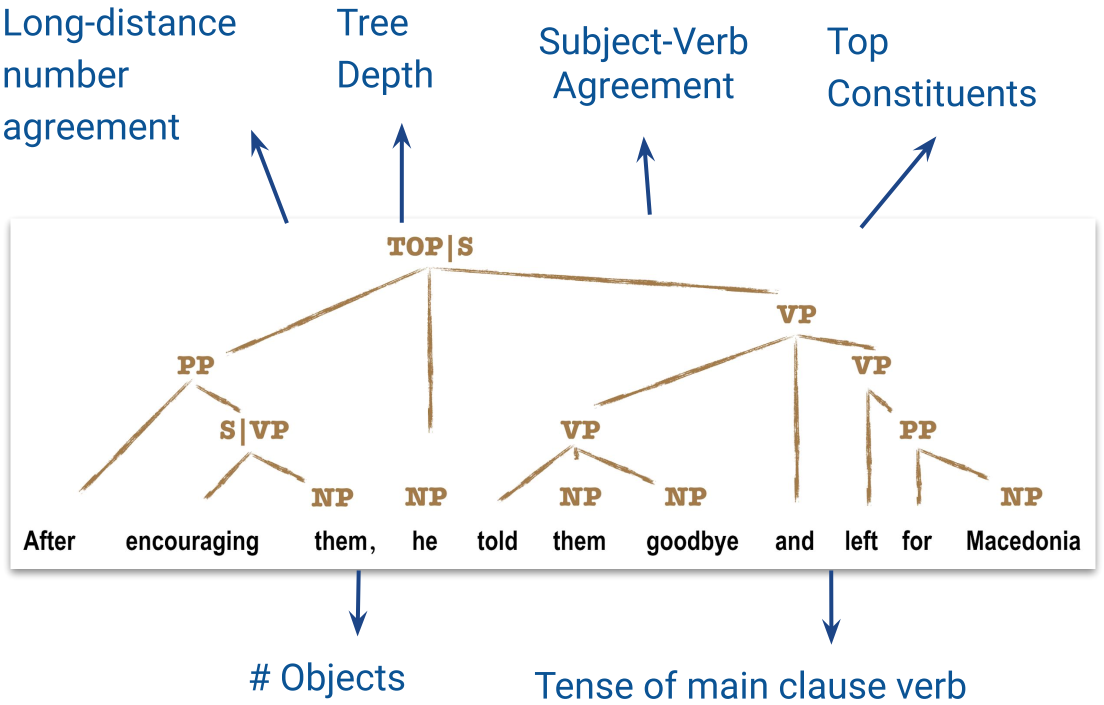
Probing Surface-level Features

- Given a sentence, predict properties such as
  - Length
  - Is a word in the sentence?
- Given a word in a sentence predict properties such as:
  - **Previously seen** words, contrast with language model
  - Position of word in the sentence
- Checks ability to memorize
  - Well-trained, richer architectures tend to fare better
  - Training on linguistic data memorizes better

Zhang et al. 2018; Liu et al., 2018; Conneau et al., 2018

Probing Morphology, Syntax, Semantics

- Morphology
- Word-level syntax
  - POS tags, CCG supertags
  - Constituent parent, grandparent…
- Partial syntax
  - Dependency relations
- Partial semantics
  - Entity Relations
  - Coreference
  - Roles



Adi et al., 2017; Conneau et al., 2018; Belinkov et al., 2017; Zhang et al., 2018; Blevins et al., 2018; Tenney et al. 2019; Liu et al., 2019

Probing classifier findings

	CoVe Lex.	CoVe Full	CoVe Abs. Δ	ELMo Lex.	ELMo Full	ELMo Abs. Δ	GPT Lex.	GPT cat	GPT mix
Part-of-Speech	85.7	94.0	8.4	90.4	**96.7**	6.3			
Constituents	56.1	81.6	25.4	69.1	**84.6**	15.4			
Dependencies	75.0	83.6	8.6	80.4	**93.9**	13.6			
Entities	88.4	90.3	1.9	92.0	**95.6**	3.5			
SRL (all)	59.7	80.4	20.7	74.1	**90.1**	16.0			
Core roles	*56.2*	*81.0*	*24.7*	*73.6*	***92.6***	*19.0*			
Non-core roles	*67.7*	*78.8*	*11.1*	*75.4*	***84.1***	*8.8*			
OntoNotes coref.	72.9	79.2	6.3	75.3	84.0	8.7			
SPR1	73.7	77.1	3.4	80.1	**84.8**	4.7			
SPR2	76.6	80.2	3.6	82.1	83.1	1.0			
Winograd coref.	52.1	**54.3**	2.2	**54.3**	53.5	-0.8			
Rel. (SemEval)	51.0	60.6	9.6	55.7	77.8	22.1			
Macro Average	69.1	78.1	9.0	75.4	**84.4**	9.1			

	BERT-base F1 Score Lex.	BERT-base F1 Score cat	BERT-base F1 Score mix	BERT-base Abs. Δ ELMo	BERT F1 Score Lex.	BERT F1 Score cat	BERT F1 Score mi		
Part-of-Speech	88.4	**97.0**	96.7	0.0	88.1	96.5	**96.9**	0.2	0.2
Constituents	68.4	83.7	86.7	2.1	69.0	80.1	**87.0**	0.4	2.5
Dependencies	80.1	93.0	95.1	1.1	80.2	91.5	**95.4**	0.3	1.4
Entities	90.9	96.1	96.2	0.6	91.8	96.2	**96.5**	0.3	0.9
SRL (all)	75.4	89.4	91.3	1.2	76.5	88.2	**92.3**	1.0	2.2
Core roles	*74.9*	*91.4*	*93.6*	*1.0*	*76.3*	*89.9*	***94.6***	*1.0*	*2.0*
Non-core roles	*76.4*	*84.7*	*85.9*	*1.8*	*76.9*	*84.1*	***86.9***	*1.0*	*2.8*
OntoNotes coref.	74.9	88.7	90.2	6.3	75.7	89.6	**91.4**	1.2	7.4
SPR1	79.2	84.7	**86.1**	1.3	79.6	85.1	**85.8**	-0.3	1.0
SPR2	81.7	83.0	**83.8**	0.7	81.6	83.2	**84.1**	0.3	1.0
Winograd coref.	54.3	53.6	54.9	1.4	53.0	53.8	**61.4**	6.5	7.8
Rel. (SemEval)	57.4	78.3	82.0	4.2	56.2	77.6	**82.4**	0.5	4.6
Macro Average	75.1	84.8	86.3	1.9	75.2	84.2	**87.3**	1.0	2.9

Pretrained Representation	Avg.	CCG	POS PTB	POS EWT	Chunk	NER	ST	GED	Supersense ID PS-Role	Supersense ID PS-Fxn	EF
ELMo (original) best layer	81.58	93.31	97.26	95.61	90.04	82.85	93.82	29.37	75.44	84.87	73.20
ELMo (4-layer) best layer	81.58	93.81	**97.31**	95.60	89.78	82.06	**94.18**	29.24	74.78	85.96	73.03
ELMo (transformer) best layer	80.97	92.68	97.09	95.13	93.06	81.21	93.78	30.80	72.81	82.24	70.88
OpenAI transformer best layer	75.01	82.69	93.82	91.28	86.06	58.14	87.81	33.10	66.23	76.97	74.03
BERT (base, cased) best layer	84.09	93.67	96.95	95.21	92.64	82.71	93.72	43.30	**79.61**	87.94	75.11
BERT (large, cased) best layer	**85.07**	**94.28**	96.73	**95.80**	**93.64**	**84.44**	93.83	**46.46**	79.17	**90.13**	**76.25**
GloVe (840B.300d)	59.94	71.58	90.49	83.93	62.28	53.22	80.92	14.94	40.79	51.54	49.70
Previous state of the art (without pretraining)	83.44	94.7	97.96	95.82	95.77	91.38	95.15	39.83	66.89	78.29	77.10

Liu et al. NAACL 2019

Method	Distance UUAS	Distance DSpr.	Depth Root%	Depth NSpr.
LINEAR	48.9	0.58	2.9	0.27
ELMO0	26.8	0.44	54.3	0.56
DECAY0	51.7	0.61	54.3	0.56
PROJ0	59.8	0.73	64.4	0.75
ELMO1	77.0	0.83	86.5	0.87
BERTBASE7	79.8	0.85	88.0	0.87
BERTLARGE15	**82.5**	0.86	89.4	0.88
BERTLARGE16	81.7	**0.87**	**90.1**	**0.89**

Hewitt et al., 2019

Tenney et al., ACL 2019

Probing classifier findings

	CoVe			ELMo			GPT		
	Lex.	Full	Abs. Δ	Lex.	Full	Abs. Δ	Lex.	cat	mix
Part-of-Speech	85.7	94.0	8.4	90.4	**96.7**	6.3			
Constituents	56.1	81.6	25.4						
Dependencies	75.0	83.6	8.6						
Entities	88.4	90.3	1.9						
SRL (all)	59.7	80.4	20.7						
Core roles	*56.2*	*81.0*	*24.7*						
Non-core roles	*67.7*	*78.8*	*11.1*						
OntoNotes coref.	72.9	79.2	6.3						
SPR1	73.7	77.1	3.4						
SPR2	76.6	80.2	3.6						
Winograd coref.	52.1	**54.3**	2.2						
Rel. (SemEval)	51.0	60.6	9.6						
Macro Average	69.1	78.1	9.0						

	BERT-base F1 Score			A					
	Lex.	cat	mix	F					
Part-of-Speech	88.4	**97.0**	96.7						
Constituents	68.4	83.7	86.7						
Dependencies	80.1	93.0	95.1						
Entities	90.9	96.1	96.2						
SRL (all)	75.4	89.4	91.3						
Core roles	*74.9*	*91.4*	*93.6*						
Non-core roles	*76.4*	*84.7*	*85.9*						
OntoNotes coref.	74.9	88.7	90.2	6.3	75.7	89.6	**91.4**	1.2	7.4
SPR1	79.2	84.7	**86.1**	1.3	79.6	85.1	**85.8**	-0.3	1.0
SPR2	81.7	83.0	**83.8**	0.7	81.6	83.2	**84.1**	0.3	1.0
Winograd coref.	54.3	53.6	54.9	1.4	53.0	53.8	**61.4**	6.5	7.8
Rel. (SemEval)	57.4	78.3	82.0	4.2	56.2	77.6	**82.4**	0.5	4.6
Macro Average	75.1	84.8	86.3	1.9	75.2	84.2	**87.3**	1.0	2.9

[Tenney et al., ACL 2019](#)

- Contextualized > non-contextualized
  - Especially on **syntactic** tasks
  - Closer performance on semantic tasks
  - **Bidirectional** context is important
- **BERT** (large) almost always gets the highest performance
  - Grain of salt: Different contextualized representations were trained on different data, using different architectures...

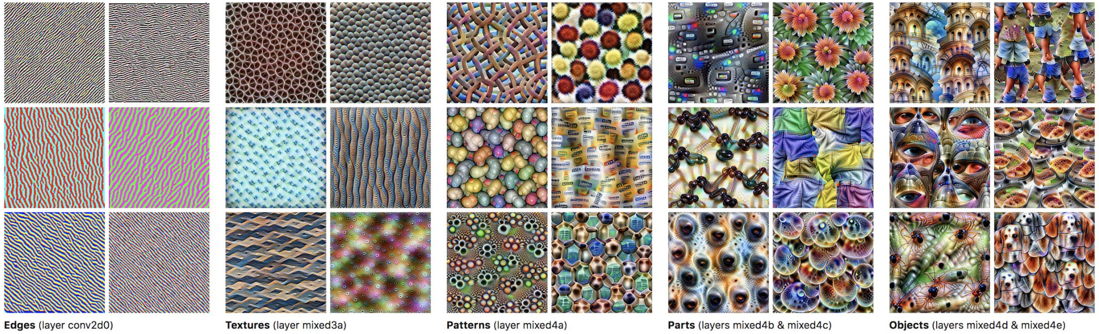
			Supersense ID		
	T	GED	PS-Role	PS-Fxn	EF
	.82	29.37	75.44	84.87	73.20
	.18	29.24	74.78	85.96	73.03
	.78	30.80	72.81	82.24	70.88
	.81	33.10	66.23	76.97	74.03
	.72	43.30	**79.61**	87.94	75.11
	.83	**46.46**	79.17	**90.13**	**76.25**
	.92	14.94	40.79	51.54	49.70
	.15	39.83	66.89	78.29	77.10

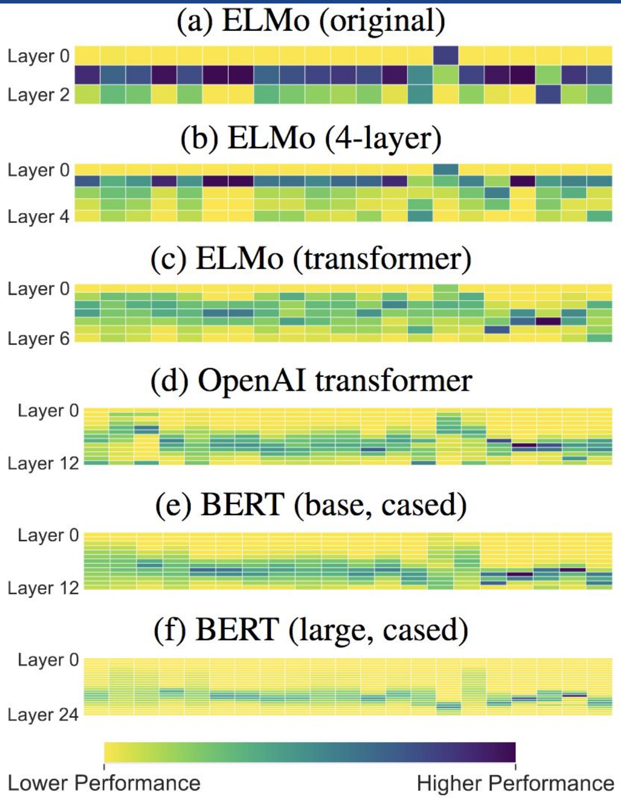
2019)

				Spr.
				.27
ELMo0	26.8	0.44	54.3	0.56
DECAY0	51.7	0.61	54.3	0.56
PROJ0	59.8	0.73	64.4	0.75
ELMo1	77.0	0.83	86.5	0.87
BERTBASE7	79.8	0.85	88.0	0.87
BERTLARGE15	**82.5**	0.86	89.4	0.88
BERTLARGE16	81.7	**0.87**	**90.1**	**0.89**

[Hewitt et. al., 2019](#)

Probing: Layers of the network





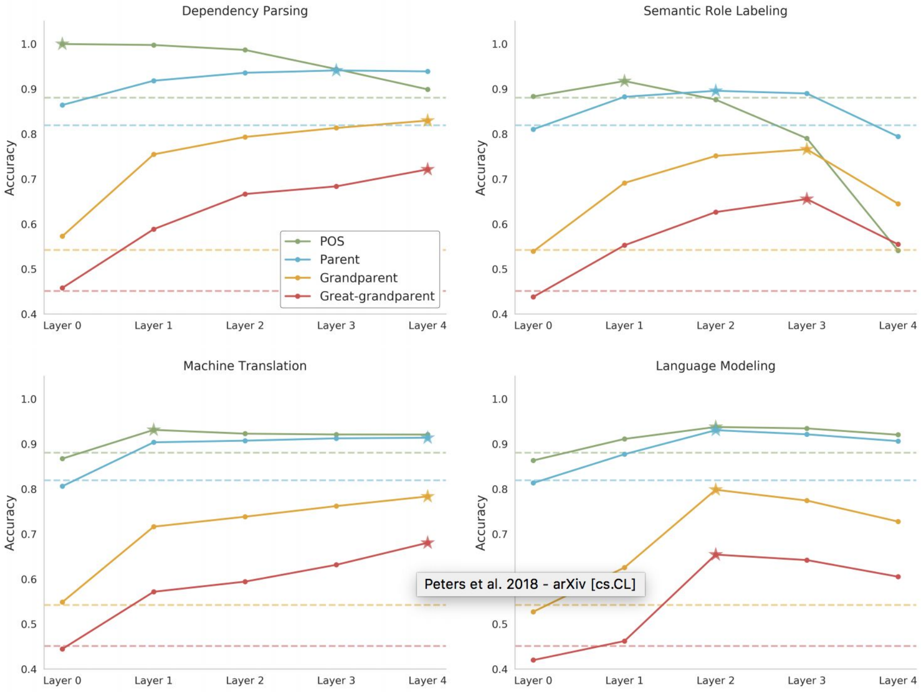
- ❑ **RNN** layers: General linguistic properties
  - ❑ Lowest layers: **morphology**
  - ❑ Middle layers: **syntax**
  - ❑ Highest layers: Task-specific **semantics**
- ❑ **Transformer** layers:
  - ❑ Different trends for different tasks; **middle-heavy**
  - ❑ Also see Tenney et. al., 2019

Fig. from Liu et al. (NAACL 2019)

Probing: Pretraining Objectives

- Language modeling **outperforms** other unsupervised and supervised objectives.
  - Machine Translation
  - Dependency Parsing
  - Skip-thought

- **Low-resource** settings (size of training data) might result in opposite trends.



Zhang et al., 2018; Blevins et al., 2018; Liu et al., 2019;

What have we learnt so far?

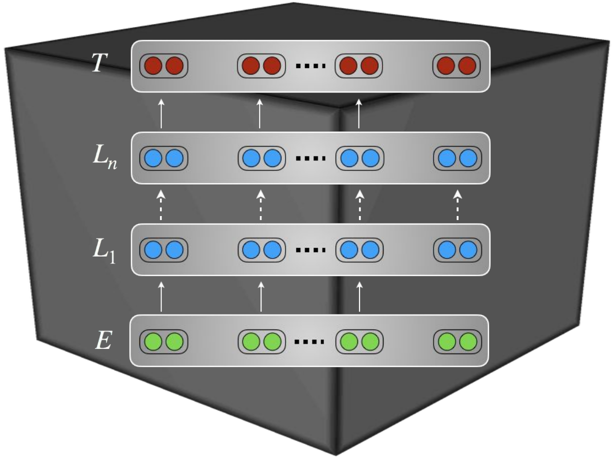
- Representations are **predictive** of certain linguistic phenomena:
  - **Alignments** in translation, Syntactic **hierarchies**
- Pretraining with and without syntax:
  - Better performance with syntax
  - But without, some notion of syntax at least (Williams et al. 2018)
- Network architectures determine what is in a representation
  - Syntax and BERT Transformer (Tenney et al., 2019; Goldberg, 2019)
  - Different layer-wise trends across architectures

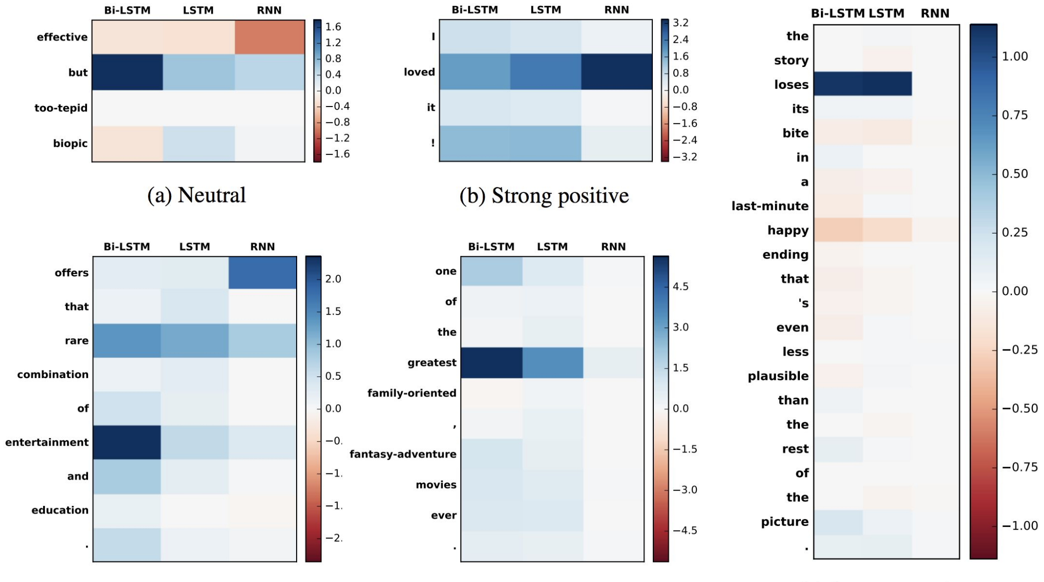
Open questions about probes

- What information should a good probe look for?
  - Probing a probe!
- What does probing performance tell us?
  - Hard to synthesize results across a variety of baselines...
- Can introduce some complexity in itself
  - linear or non-linear classification.
  - behavioral: design of input sentences
- Should we be using **probes as evaluation metrics**?
  - might defeat the purpose...

Analysis Method 4: Model Alterations

- Progressively erase or mask network components
  - Word embedding dimensions
  - Hidden units
  - Input - words / phrases





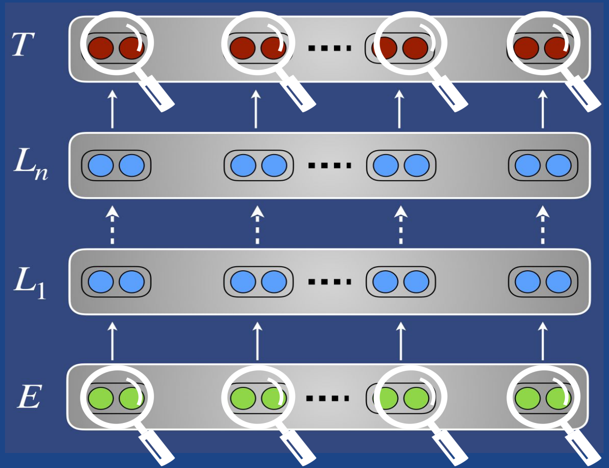
(a) Neutral (b) Strong positive (c) Strong positive (d) Strong positive (e) Strong negative

Figure 5: Heatmap of word importance (computed using Eq. 1) in sentiment analysis.

Li et al., 2016

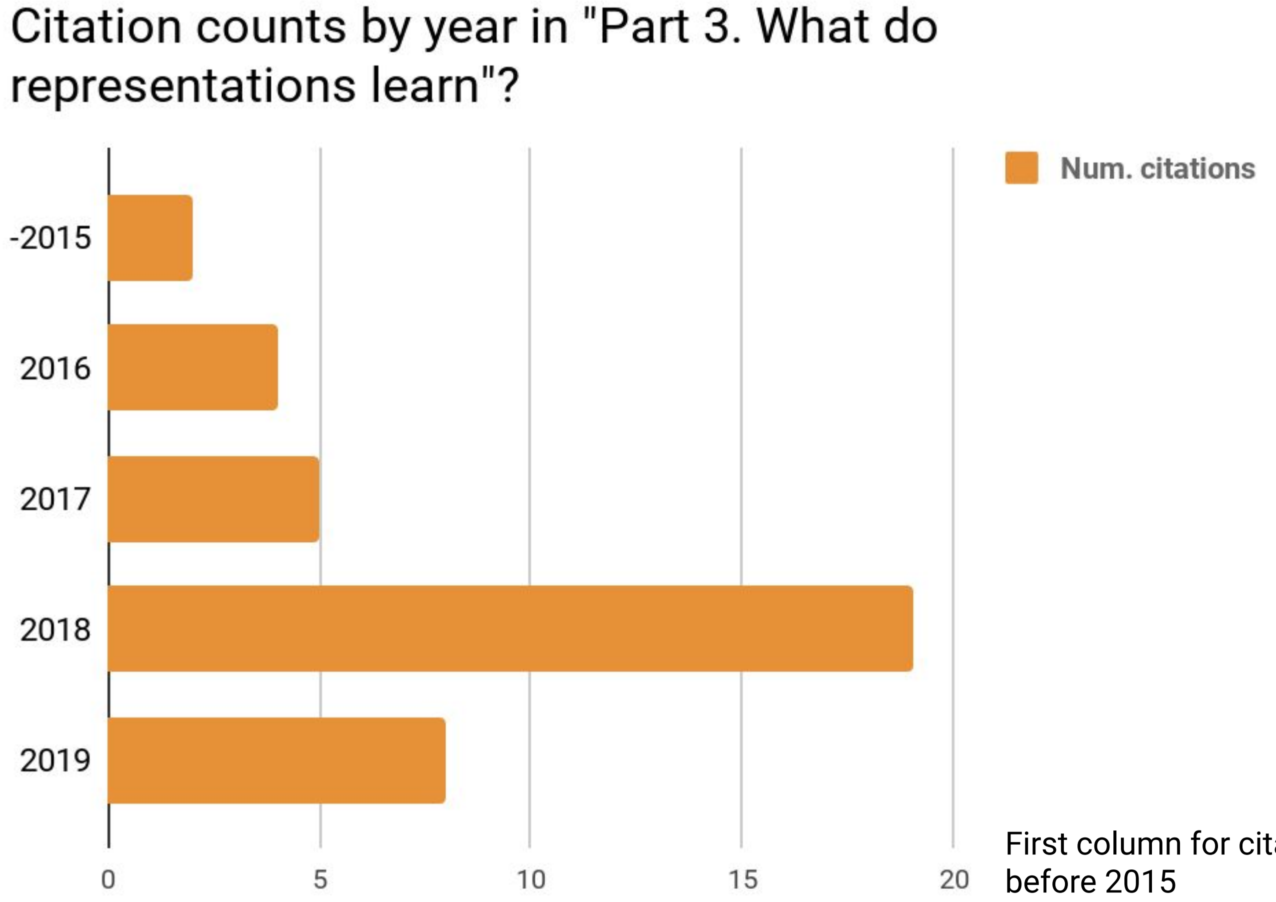
So, what is in a representation?

- Depends on how you look at it!
  - **Visualization**:
    - **bird's eye view**
    - **few** samples -- might call to mind cherry-picking
  - **Probes**:
    - discover corpus-wide **specific** properties
    - may introduce own biases...
  - **Network ablations**:
    - great for **improving modeling**,
    - could be task specific



- Analysis methods as tools to aid model development!

Very current and ongoing!



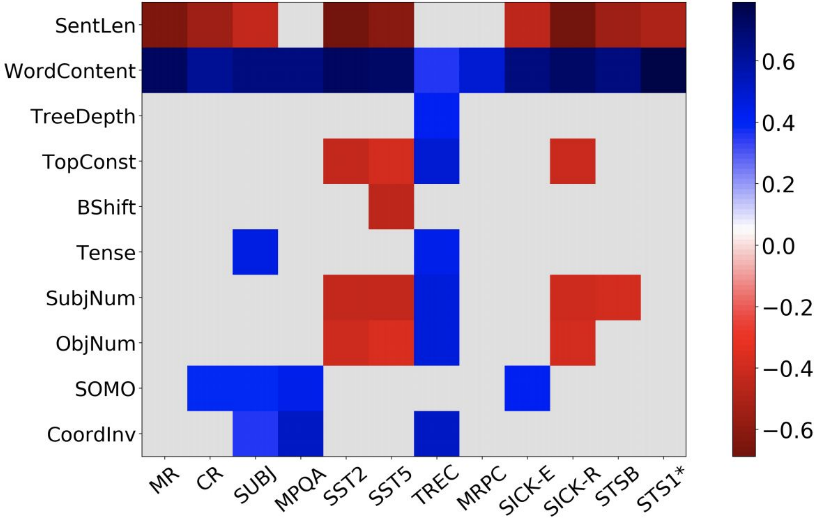
First column for citations in and before 2015

What's next?

- ❑ Linguistic Awareness
- ❑ Interpretability

Interpretability + transferability to downstream tasks is key

→ Up next!



[Conneau et al., 2018]

Correlation of probes to downstream tasks

Some Pointers

- Suite of word-based and word-pair-based tasks: Liu et al. 2019

  https://github.com/nelson-liu/contextual-repr-analysis

- Structural Probes: Hewitt & Manning 2019
- Overview of probes : Belinkov & Glass, 2019

That’s all for this time

Image credit: Andrejs Kirma