Transfer Learning for NLP

2/2565: FRA501 Introduction to Natural Language Processing with Deep learning

Week 10

Paisit Khanarsa, Ph.D.

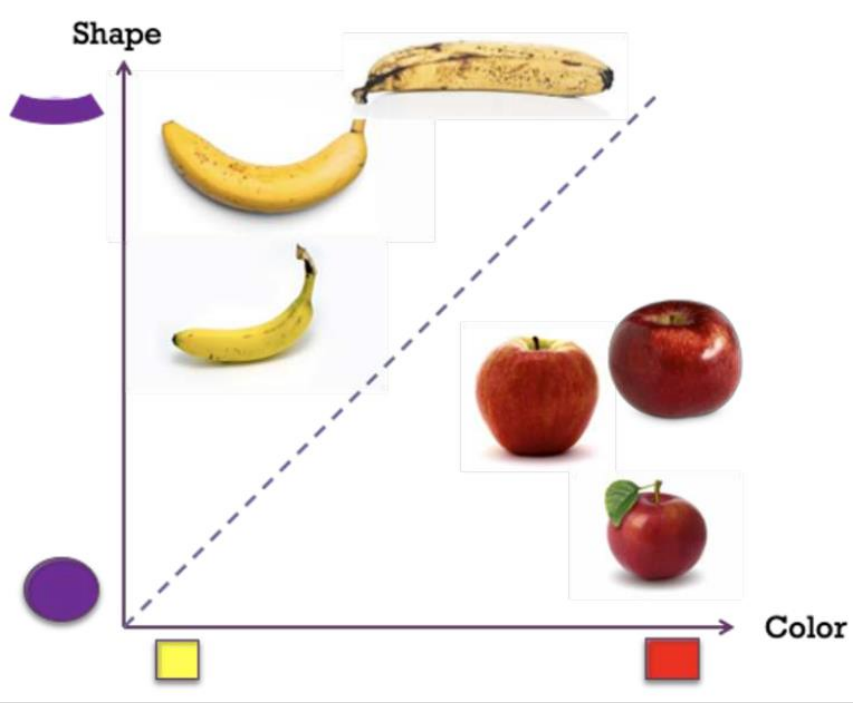
Institute of Field Robotics (FIBO), King Mongkut's University of Technology Thonburi

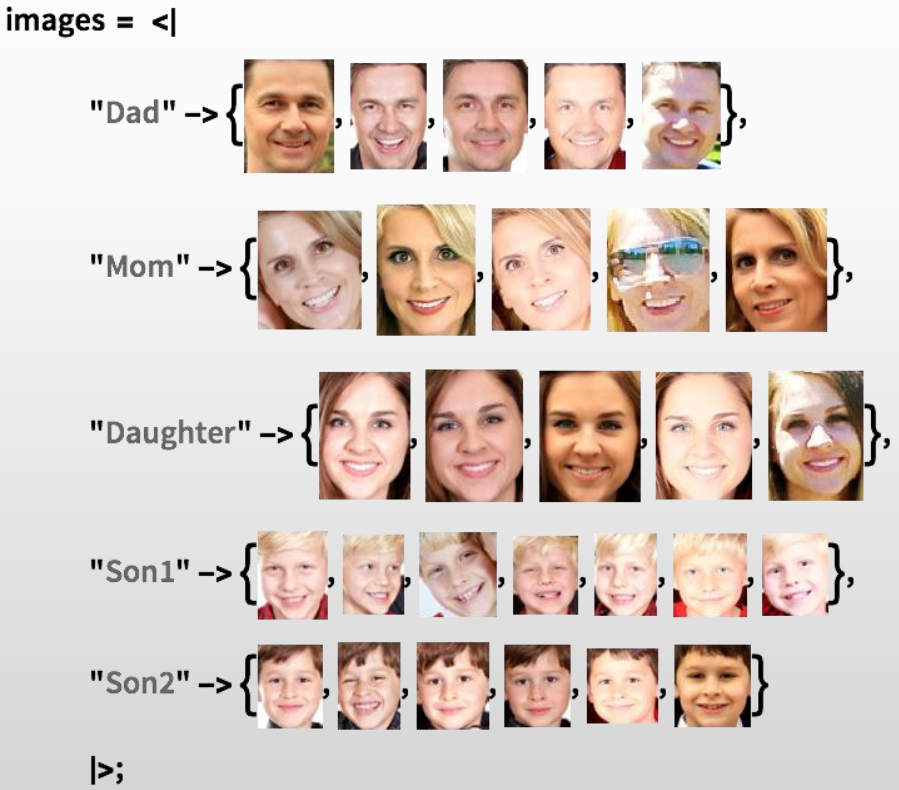
Outlines

- Introduction to transfer learning
- Transfer learning for NLP
- Contextualized representations of pre-trained models
  - ULMFiT (Universal Language Model Fine-tuning)
  - ELMo (Embeddings from Language Models)
  - BERT (Bidirectional Encoder Representations from Transformers)
  - GPT (Generative Pre-Training)
  - Flair (Contextual string embedding for sequence labeling)

Introduction to transfer learning

- We need enough training data to infer the data pattern and to create model
- A simple problem requires less training data.



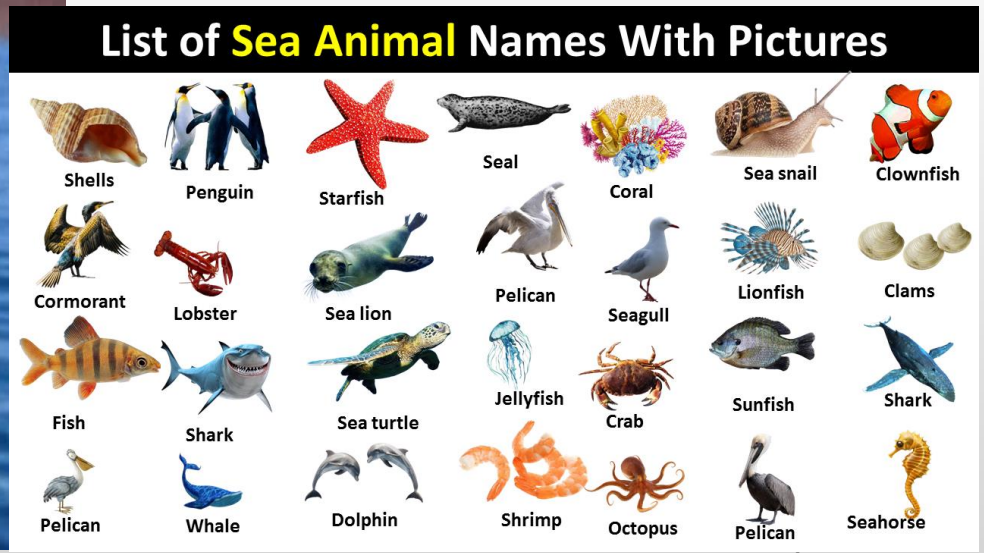


Introduction to transfer learning

- Have you ever seen this creature before?
- Can you guess whether it is a land animal or a water animal?

Introduction to transfer learning

- Have you ever seen this creature before?
- Can you guess whether it is a land animal or a water animal?
- You can **transfer** your knowledge from the past



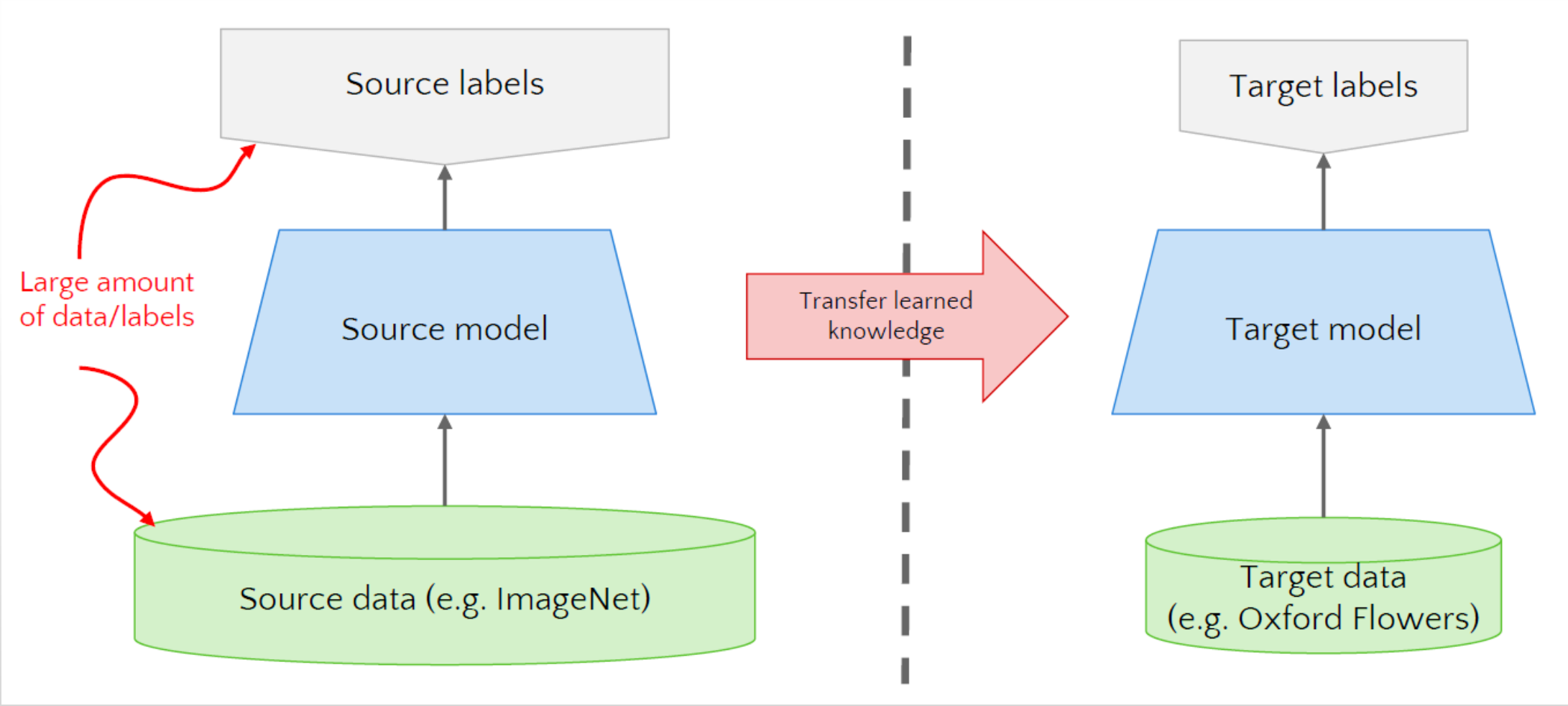
Transfer learning: ideas

- **Myth:** you can't do deep learning unless you have a million labelled examples for your problem.
- **Reality:**
  - You can transfer learned representations from a related task
  - You can train on a nearby surrogate objective for which it is easy to generate labels

Transfer learning: ideas

- Instead of training a deep network from scratch for your task, you can
  - **Take** a network trained on a different domain (data) for a different source task (e.g., LM)
  - **Adapt** it for your domain (data) and your target task (e.g., classification)
- **Variations:**
  - Different domain (data), same task
  - Different domain (data), different task

Transfer learning: ideas



Transfer learning: IMAGENET

https://image-net.org/index.php

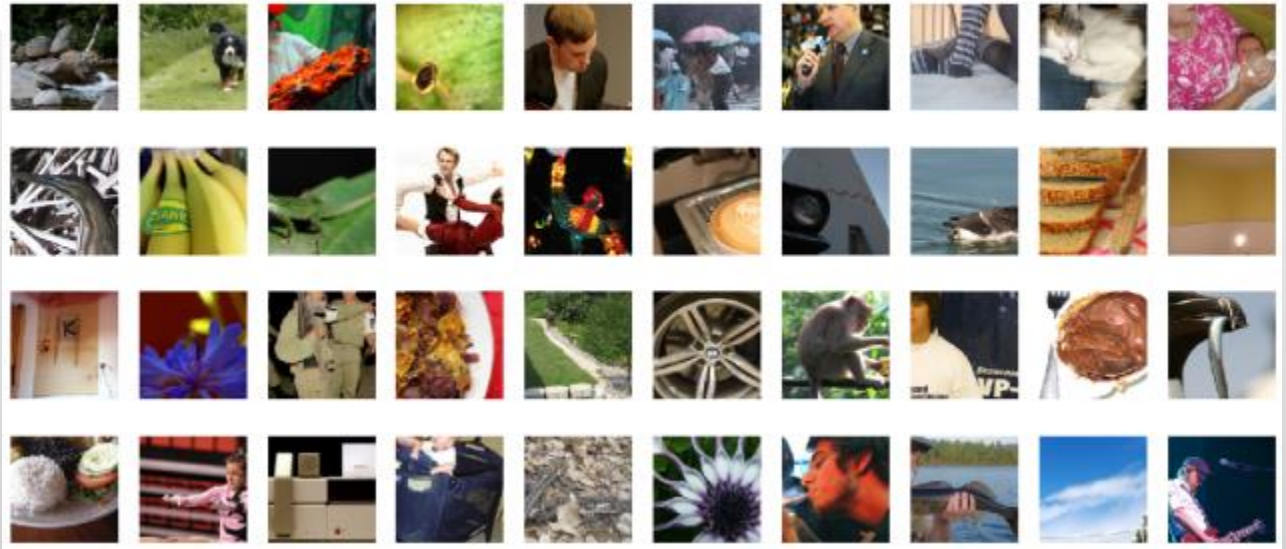
IMAGENET

14,197,122 images, 21841 synsets indexed

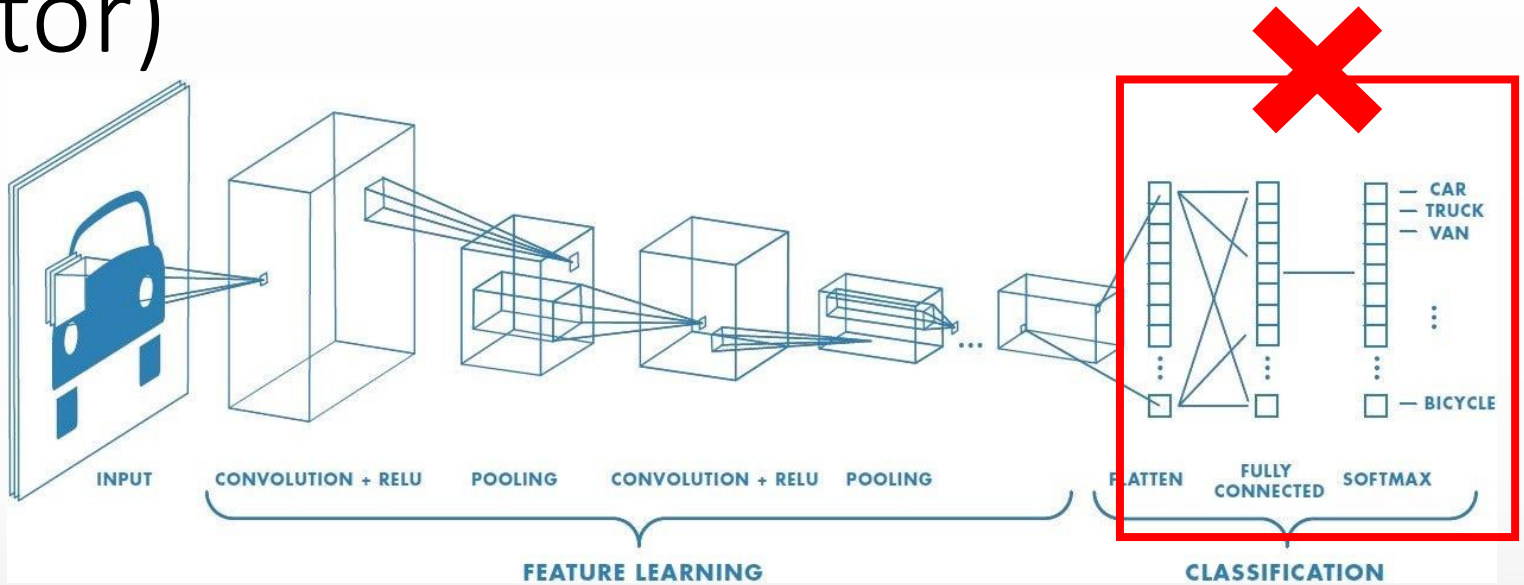
Home Download Challenges About

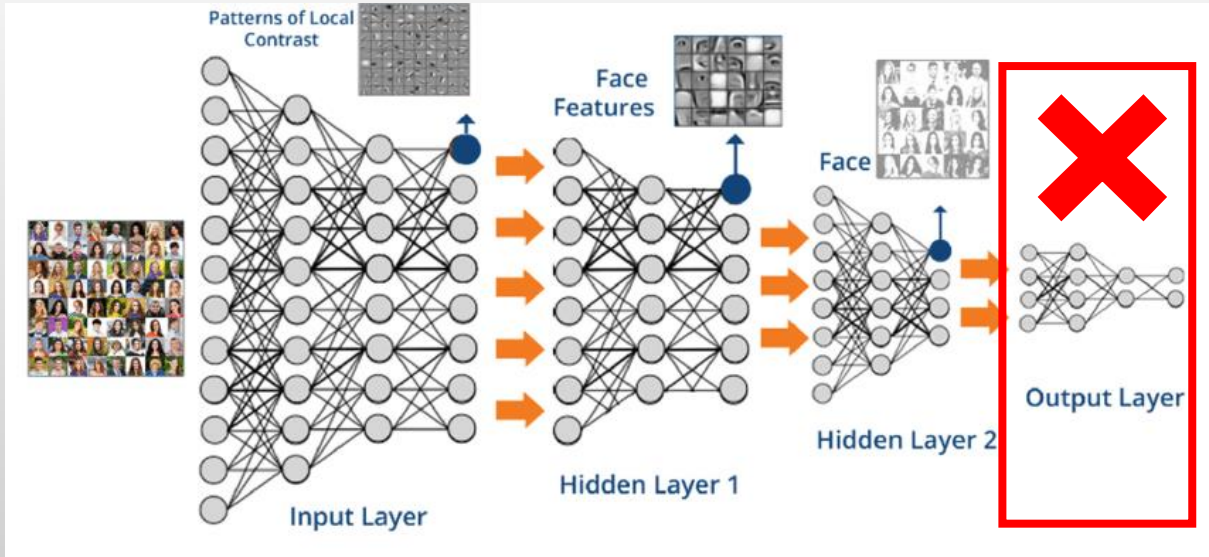
Not logged in. Login | Signup

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The project has been instrumental in advancing computer vision and deep learning research. The data is available for free to researchers for non-commercial use.



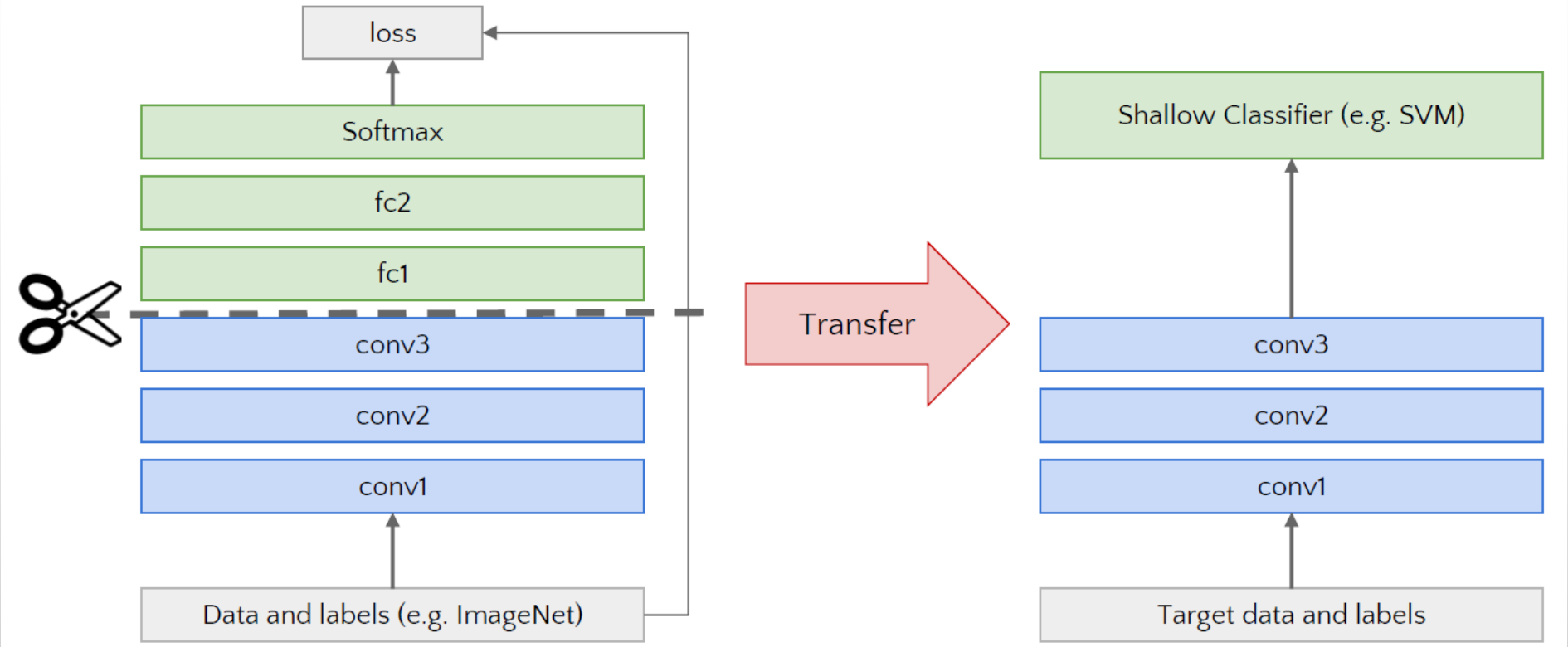
Transfer learning: Off-the-shelf (Feature Extractor)





Transfer learning: Off-the-shelf (Feature Extractor)

- **Idea:** use outputs of one or more layers of a network trained on a different task as generic feature detectors.
- Train a new shallow model on these features.



Transfer learning: benefits

- Lisa Torrey and Jude Shavlik in their chapter on transfer learning describe three possible benefits to look for when using transfer learning:
  - **Higher start:** The initial skill (before refining the model) on the source model is higher than it otherwise would be.
  - **Higher slope:** The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
  - **Higher asymptote:** The converged skill of the trained model is better than it otherwise would be.

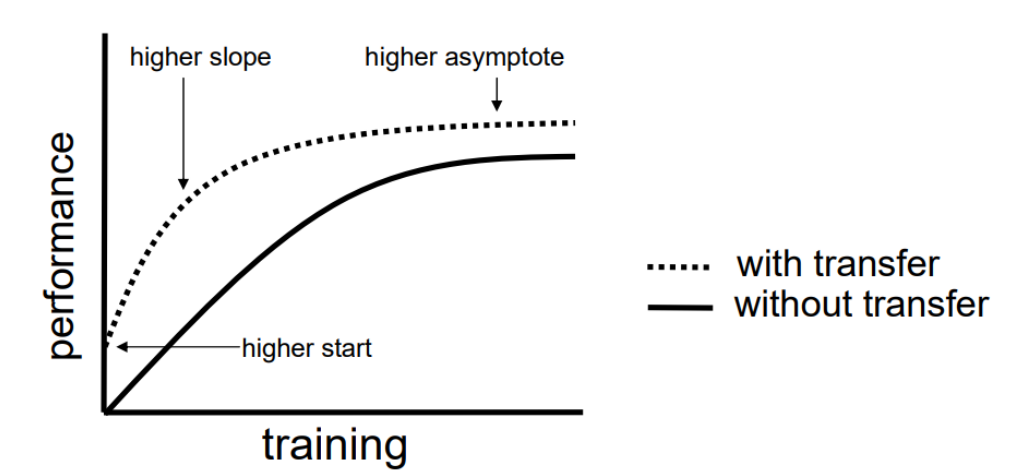
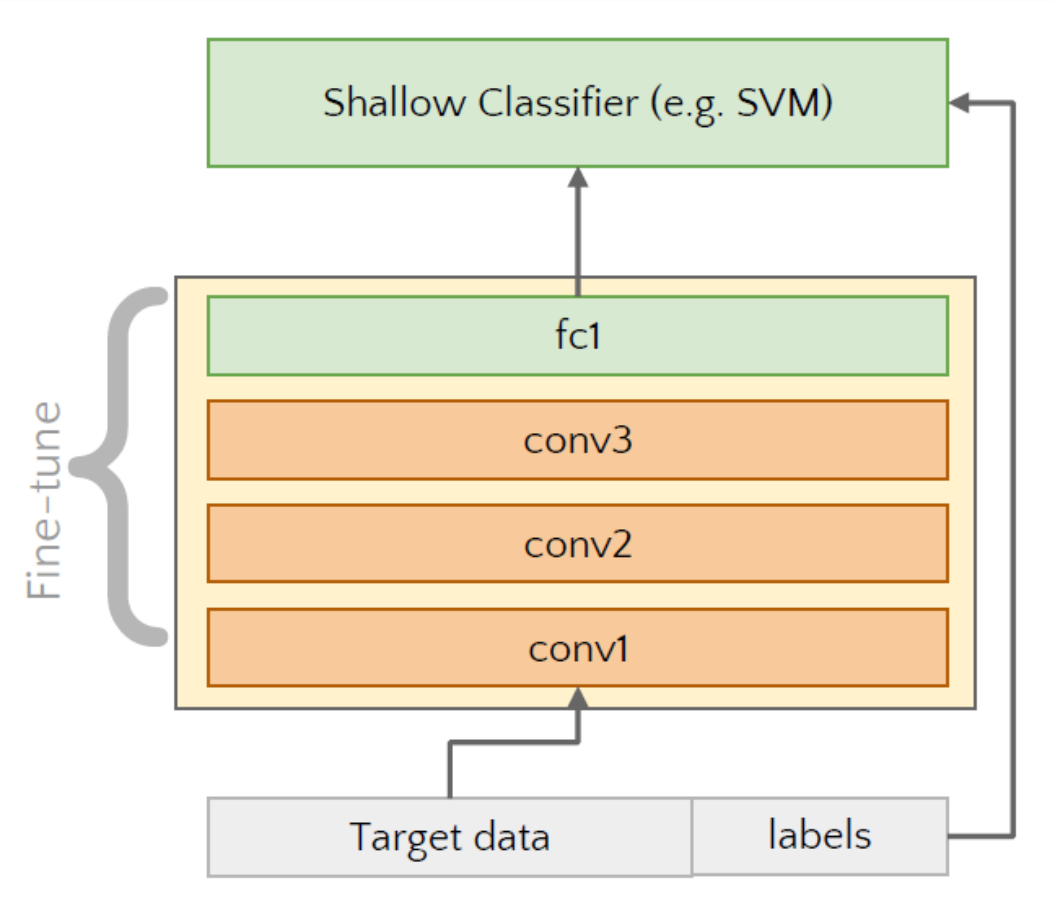


Fig. 2. Three ways in which transfer might improve learning.

https://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

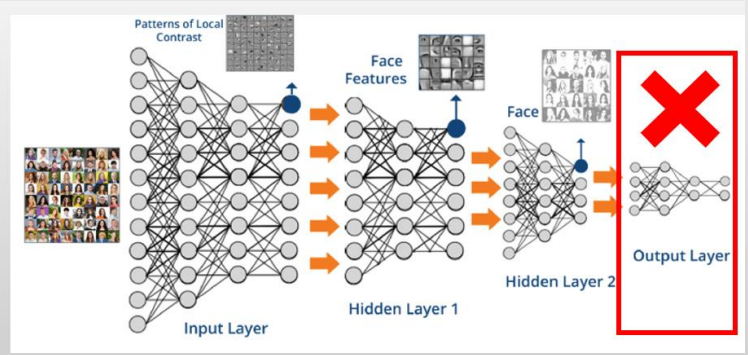
Fine-tuning: supervised task adaptation

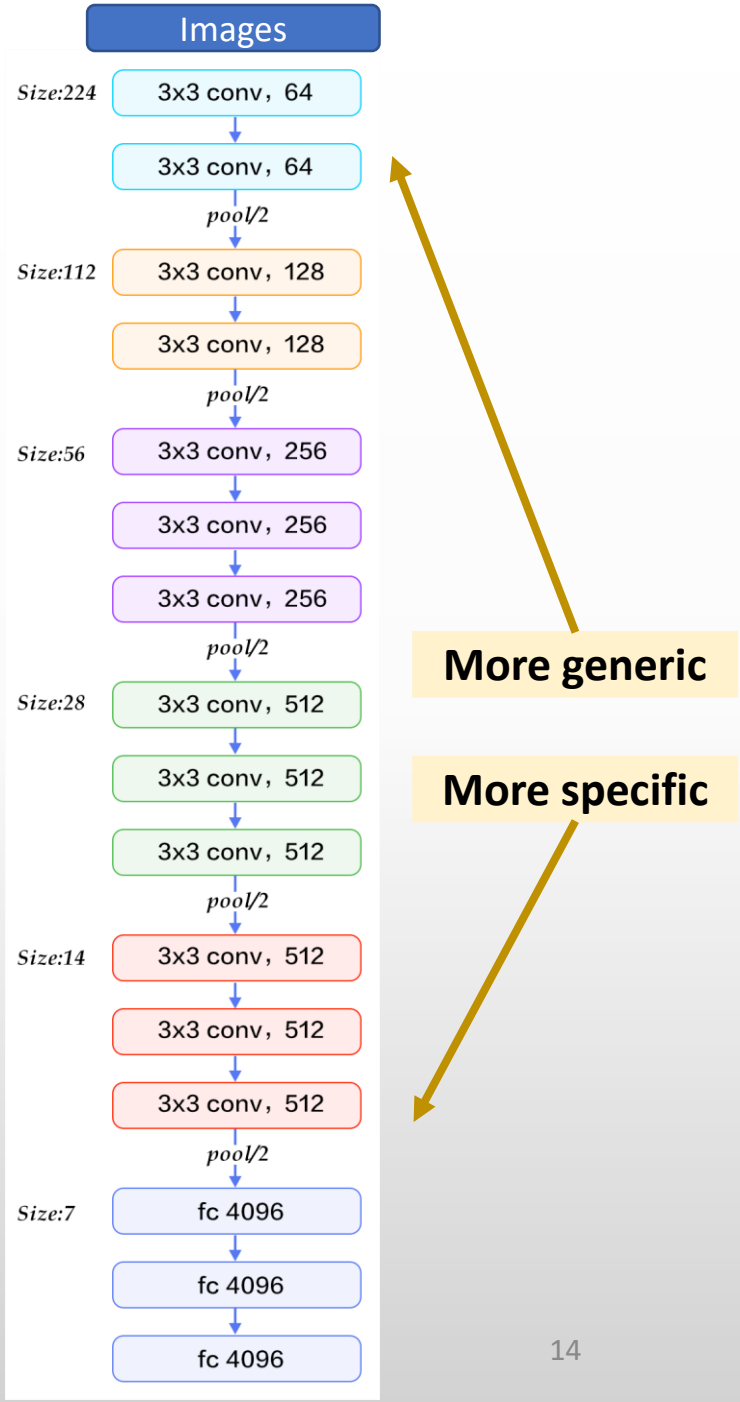
- **How can do better than off-the-shelf ?**
- Train deep net on nearby task for which it is easy to get labels using standard backprop.
  - E.g. ImageNet classification
  - Pseudo classes from augmented data
  - Slow feature learning, ego-motion
- Cut off top layer(s) of network and replace with supervised objective for target domain.
- Fine-tune network using backprop with labels for target domain until validation loss starts to increase.



How transferable are features

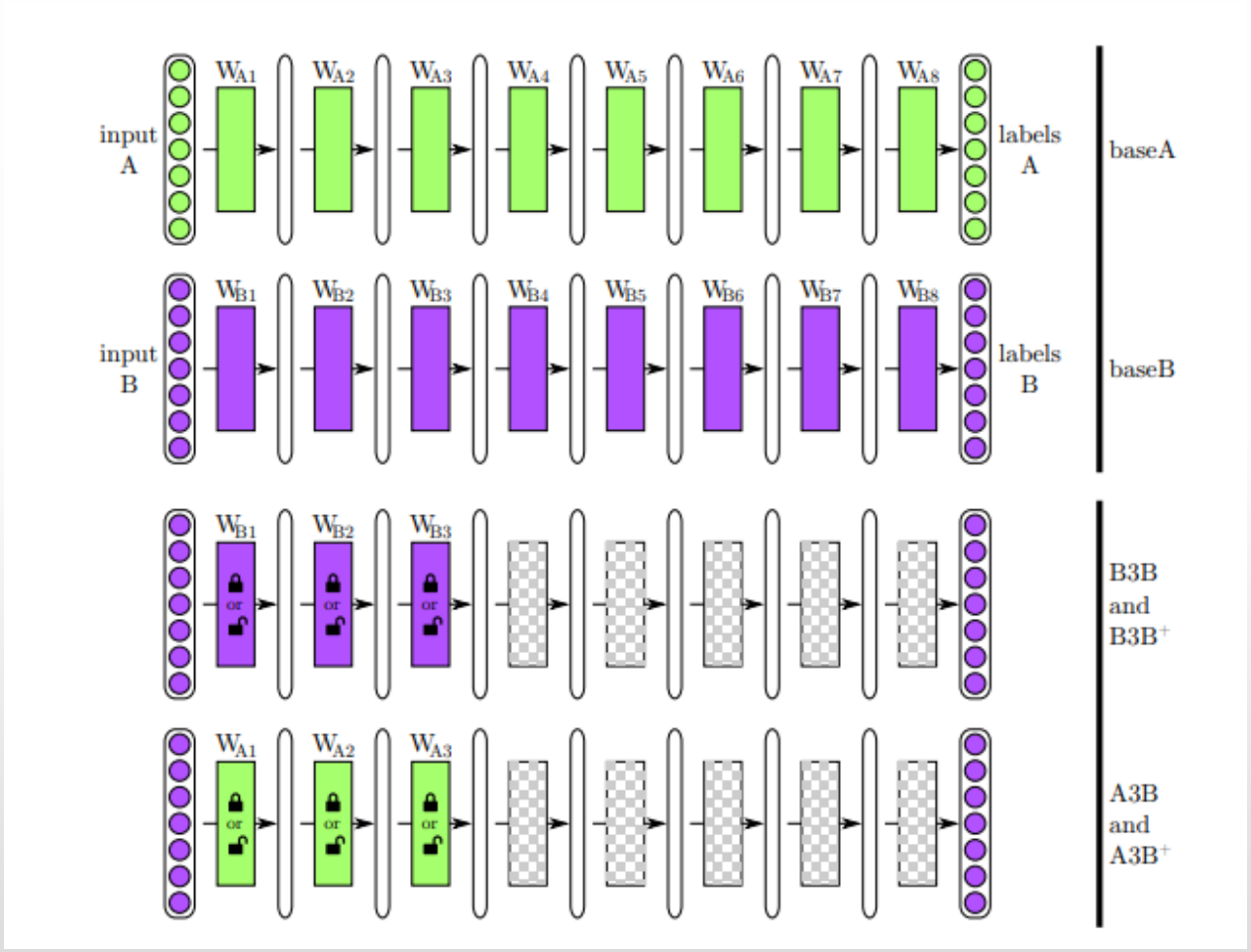
- Lower layers: more general features.
  - Transfer very well to other tasks.
- Higher layers: more task specific.
  - Fine-tuning improves generalization when sufficient examples are available.
- Transfer learning and fine tuning often lead to better performance than training from scratch on the target dataset.
- Even features transferred from distant tasks are often better than random initial weights!





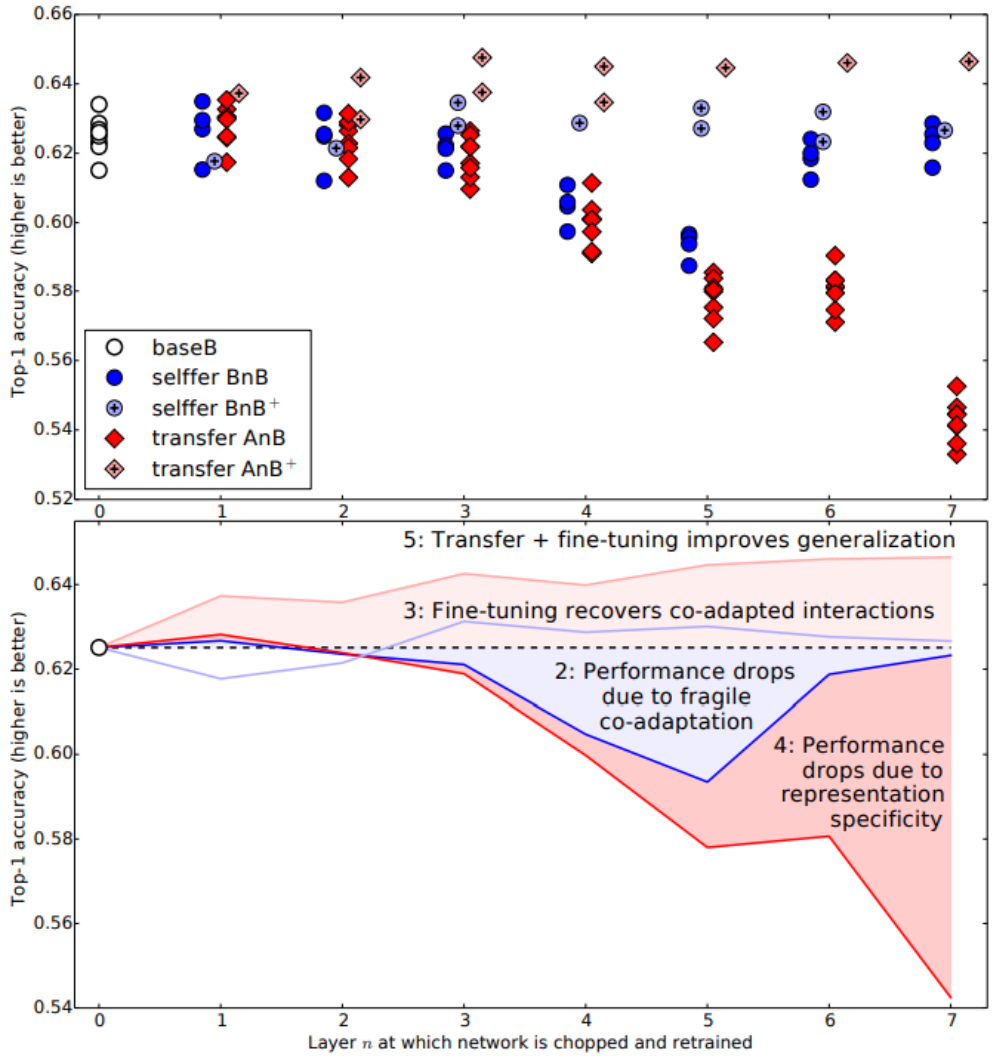
How transferable are features

- **A selffer network B3B:** the first 3 layers are copied from baseB and frozen. The five higher layers (4–8) are initialized randomly and trained on dataset B. This network is a control for the next transfer network.
- **A transfer network A3B:** the first 3 layers are copied from base A and frozen. The five higher layers (4–8) are initialized randomly and trained toward dataset B. Intuitively, here we copy the first 3 layers from a network trained on dataset A and then learn higher layer features on top of them to classify a new target dataset B. If A3B performs as well as base B, there is evidence that the third-layer features are general, at least with respect to B. If performance suffers, there is evidence that the third-layer features are specific to A.
- **B3B+ and A3B+** are just like B3B and A3B, but where all transferred layers are fine-tuned.



Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. *Advances in neural information processing systems*, *27*.

How transferable are features



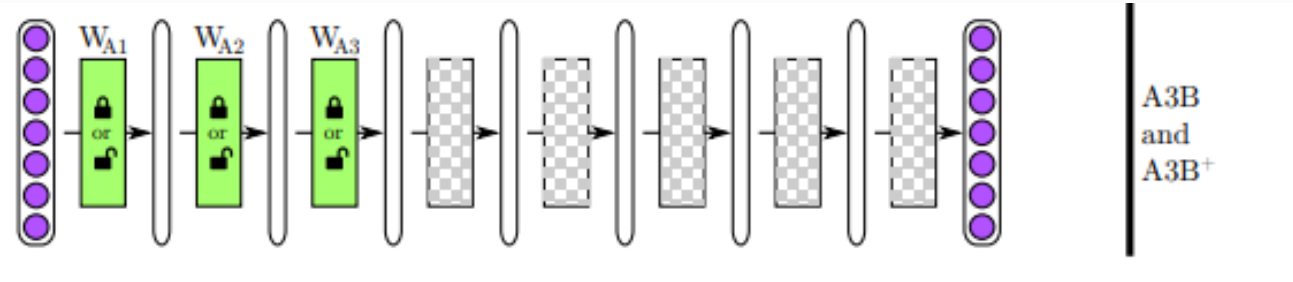


Figure 2: The results from this paper's main experiment. *Top*: Each marker in the figure represents the average accuracy over the validation set for a trained network. The white circles above $n = 0$ represent the accuracy of baseB. There are eight points, because we tested on four separate random A/B splits. Each dark blue dot represents a BnB network. Light blue points represent BnB$^+$ networks, or fine-tuned versions of BnB. Dark red diamonds are AnB networks, and light red diamonds are the fine-tuned AnB$^+$ versions. Points are shifted slightly left or right for visual clarity. *Bottom*: Lines connecting the means of each treatment. Numbered descriptions above each line refer to which interpretation from Section 4.1 applies.

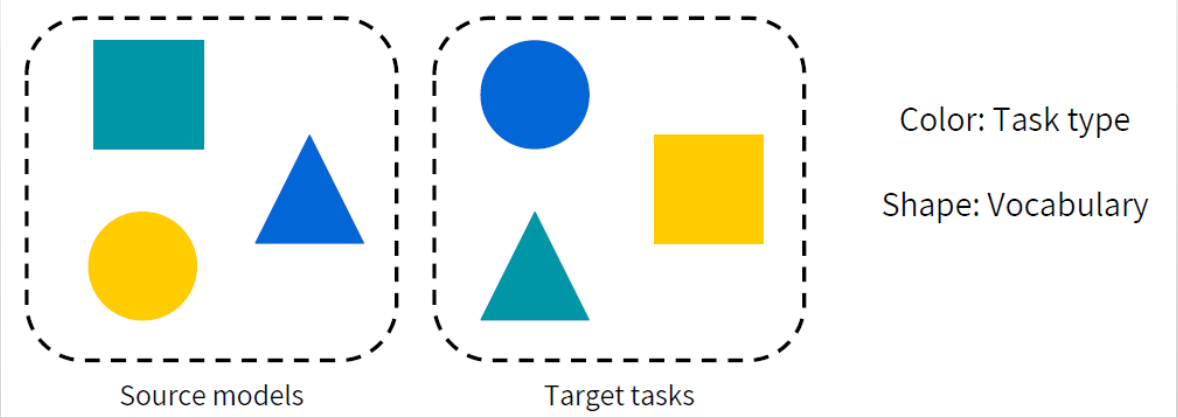
Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. *Advances in neural information processing systems*, *27*.

Transfer learning for NLP vs CV

- Differences between transfer learning for NLP and for Computer vision fields:
  - More variety in types of target tasks.
    - E.g. entity extraction, classification, sequence labeling.
  - More variety in the input data (e.g. source language, field-specific terminology).
  - No clear “ImageNet” equivalent (lack of large, generic, and labeled corpora).

Model Alignment for transfer learning

- Source model is the single most important variable.
- Keep source model and target model well-aligned (close to each other) when possible.
- Source vocabulary should be aligned with target vocabulary (similar domain).
- Source task should be aligned with target task (similar task).
- For example:
  - Good: product review sentiment → product review categorization
  - Good: hotel rating → restaurant rating
  - Less good: product review sentiment → biology paper classification

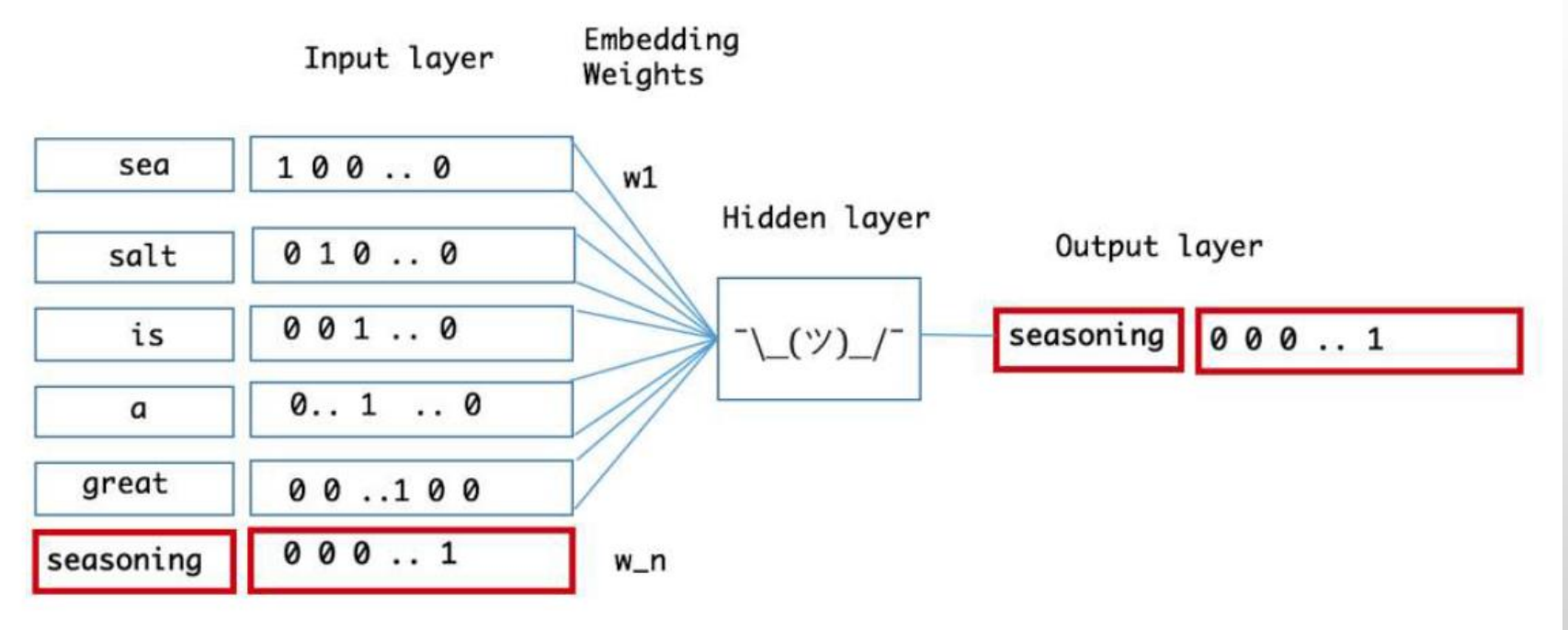


What source tasks?

- We expect the source tasks to be able to produce the good and general representations
  - Natural language inference: are two sentences in agreement, disagreement, or neither?
  - Machine translation: from one language to another language.
  - Multi-task learning: learning to solve many supervised problems at once.
  - Language modeling:
    - learning to model the distribution of natural language.
    - predicting the next word in a sequence given context.
    - No need for labeled data (unsupervised data)

Language modeling: predicting the next word

- Language modeling is a good objective for source model because:
  - It is able to capture long-term dependencies in language (LSTM)
  - Annotation is not required.
  - It effectively help the model learn syntactic (grammar) and semantic (meaning) features



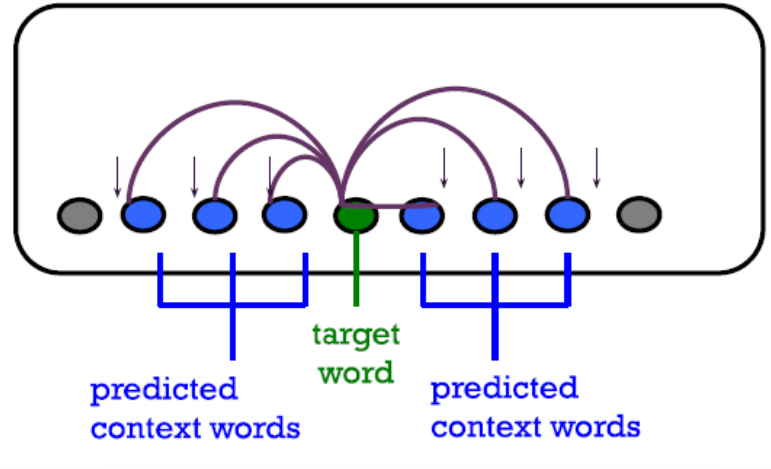
Pre-trained corpus for NLP

- In NLP, for a couple of years now, the trend is to pre-train any model on the huge text corpus:
  - BooksCorpus (980M words) [1]
  - English Wikipedia (2,300M words) [2]
  - WMT News crawl (3,600M words, only from 2008-2012) [3]
- All of these corpora are unlabeled. Since it focuses on LM (predicting next word), it doesn't need any labeling effort.

[1] Zhu et al., Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books, Proceedings of the IEEE international conference on computer vision, 2015, https://arxiv.org/abs/1506.06724
[2] https://linguatools.org/tools/corpora/wikipedia-monolingual-corpora/
[3] http://statmt.org/wmt18/translation-task.html

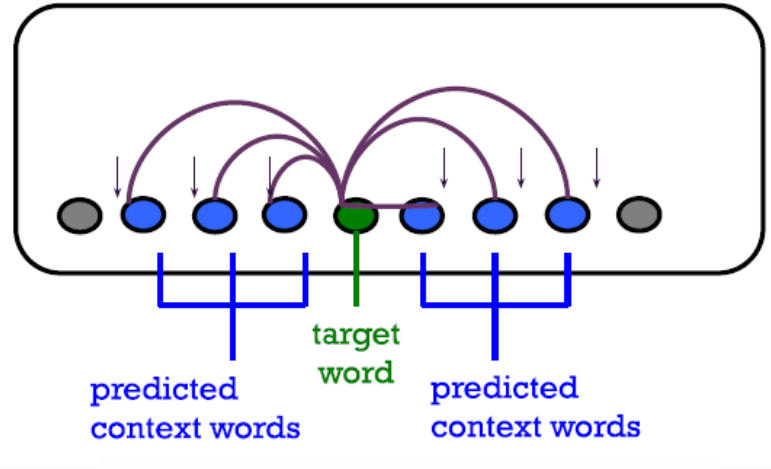
Pre-trained: Word2Vec (Static word representation)

- Two main Word2Vec models:
  - Skip-gram
  - CBOW
- GloVe (Stanford)
  - https://nlp.stanford.edu/projects/glove/
- fastText [Available in Thai language] (Facebook)
  - https://github.com/facebookresearch/fastText
- These word vectors NOT not depend on context.
  - Stick? Bar (n) or Adhere (v)
  - Same vector

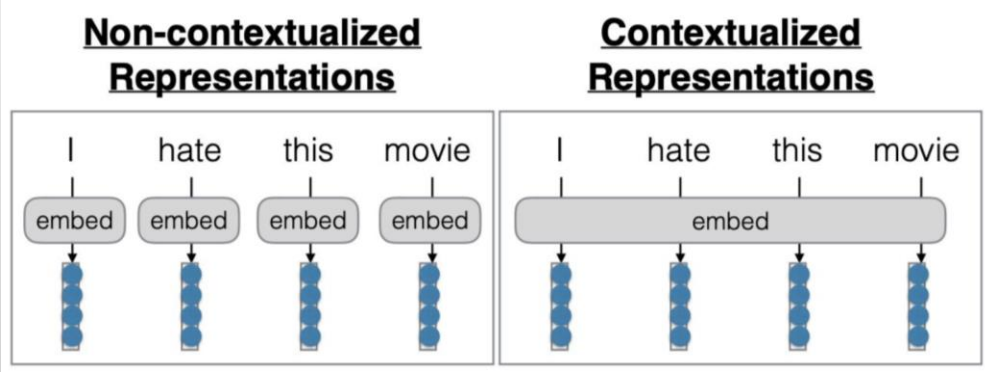


Pre-trained: Word2Vec (Static word representation)

- Two main Word2Vec models:
  - Skip-gram
  - CBOW
- GloVe (Stanford)
  - https://nlp.stanford.edu/projects/glove/
- fastText [Available in Thai language] (Facebook)
  - https://github.com/facebookresearch/fastText
- These word vectors NOT not depend on context.
  - Stick? Bar (n) or Adhere (v)
  - Same vector



Contextualized Word Embedding

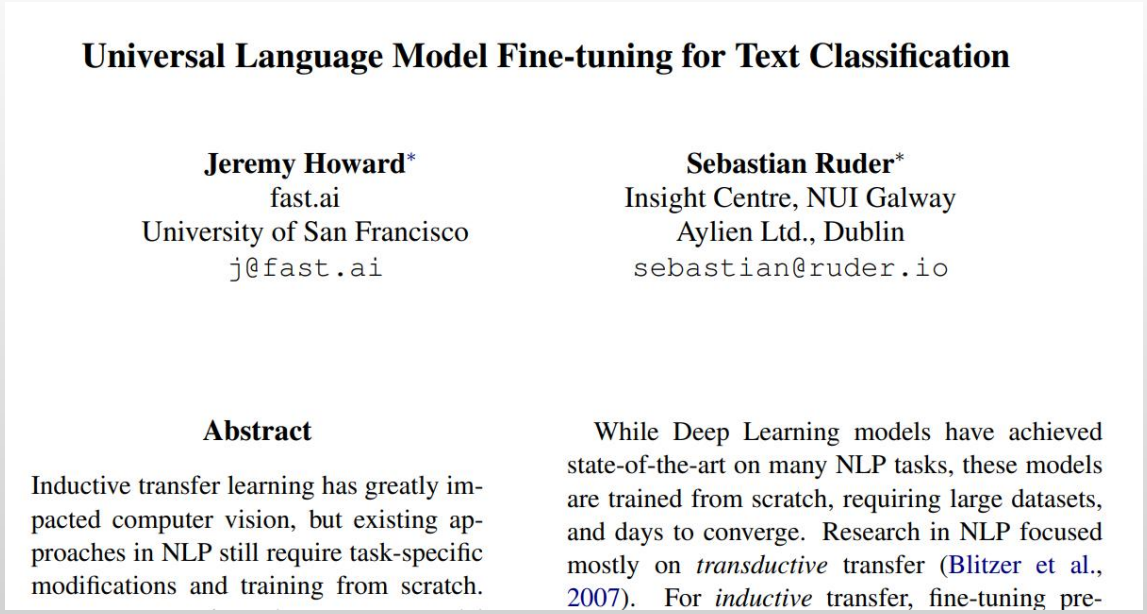


Pre-trained language modeling

- Pre-training allows a model to capture and learn a variety of linguistic phenomena, such as long-term dependencies and negation, from a large-scale unlabeled corpus.
- Then this knowledge is used (transferred) to initialize and then train another model to perform well on a specific NLP tasks.
  - ULMFiT (Universal Language Model Fine-tuning)
  - ELMo (Embeddings from Language Models)
  - BERT (Bidirectional Encoder Representations from Transformers)
  - GPT (Generative Pre-Training)
  - Flair (Contextual string embedding for sequence labeling)

1) Pre-trained language modeling: ULMFiT
(Universal Language Model Fine-tuning for Text Classification)

- ULMFiT introduced methods to effectively utilize a lot of what the model learns during pre-training.
- More than just a single embedding layer, ULMFiT introduced a language model which contains many layers and a process to effectively fine-tune that language model for various tasks.
- Unlike BERT, this language model is unidirectional LSTM which means that each word is only contextualized using the words to its left.



Universal Language Model Fine-tuning for Text Classification

Jeremy Howard*
fast.ai
University of San Francisco
j@fast.ai

Sebastian Ruder*
Insight Centre, NUI Galway
Aylien Ltd., Dublin
sebastian@ruder.io

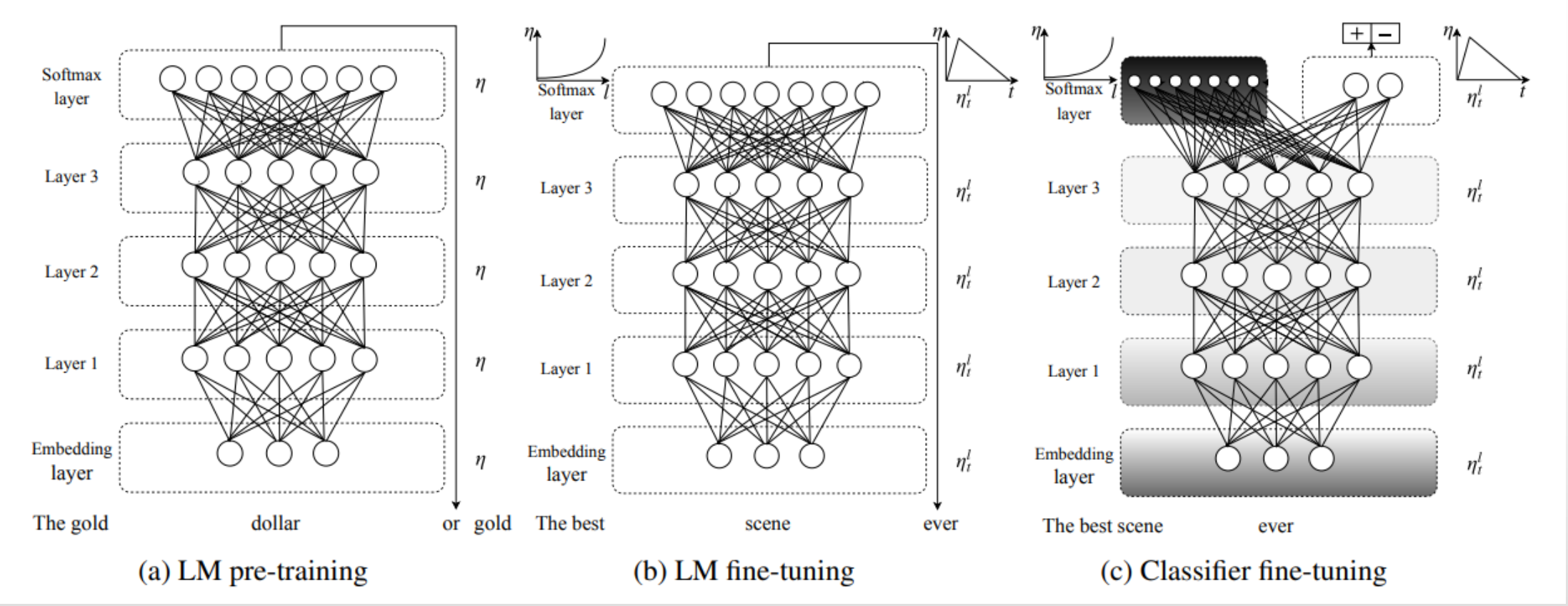
Abstract

Inductive transfer learning has greatly impacted computer vision, but existing approaches in NLP still require task-specific modifications and training from scratch.

While Deep Learning models have achieved state-of-the-art on many NLP tasks, these models are trained from scratch, requiring large datasets, and days to converge. Research in NLP focused mostly on *transductive* transfer (Blitzer et al., 2007). For *inductive* transfer, fine-tuning pre-

Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

1) Pre-trained language modeling: ULMFiT



Train LM using large unlabeled corpus (wiki corpus)

Fine-tune LM using target unlabeled corpus (target data without label)

Train classifier using target corpus (target data with label)

1) Pre-trained language modeling: ULMFiT

- **Slanted triangular learning rates (STLR),** which first linearly increases the learning rate and then linearly decays it according to the following update schedule.

$$cut = \lfloor T \cdot cut\_frac \rfloor$$

$$p = \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t - cut}{cut \cdot (1/cut\_frac - 1)}, & \text{otherwise} \end{cases}$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio}$$

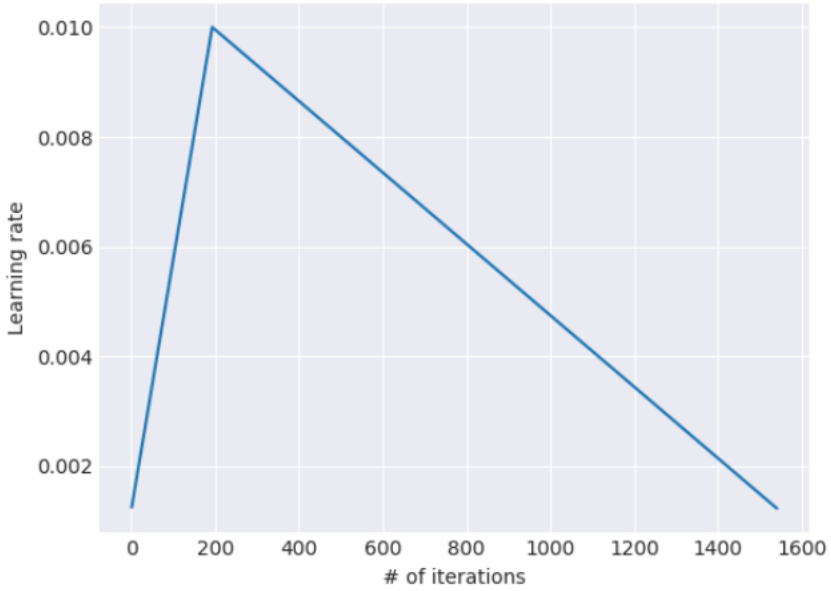
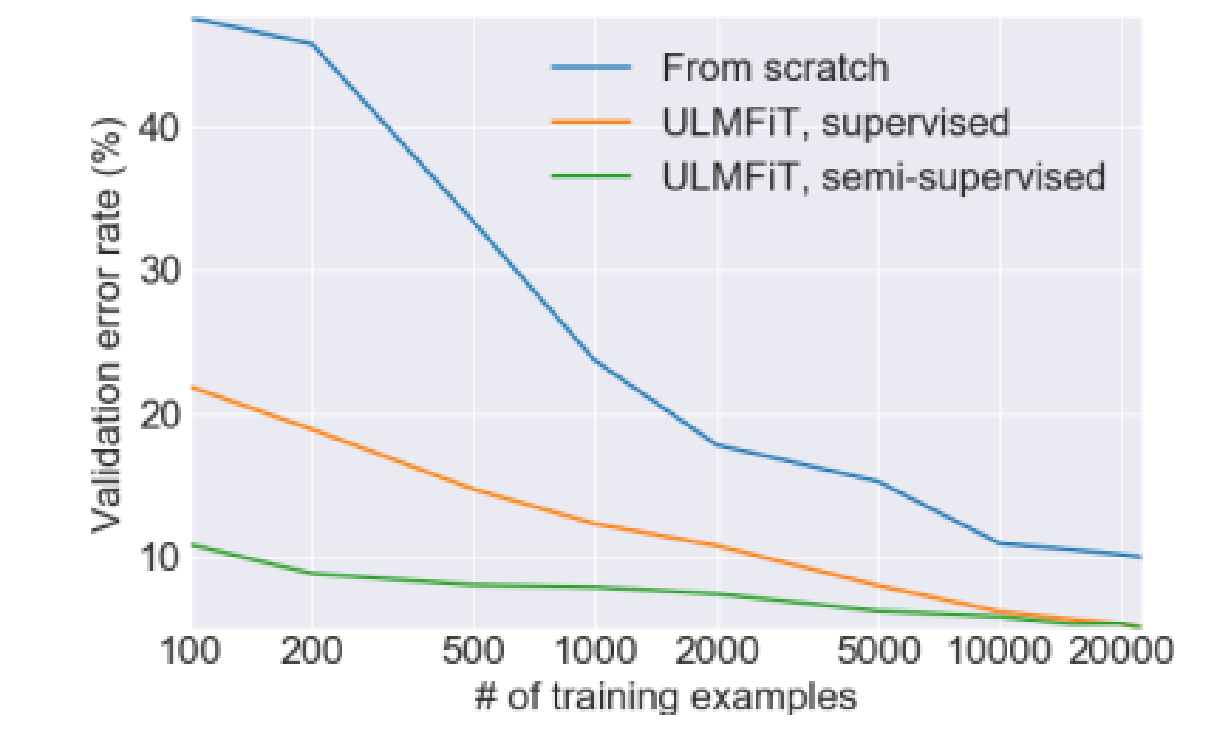


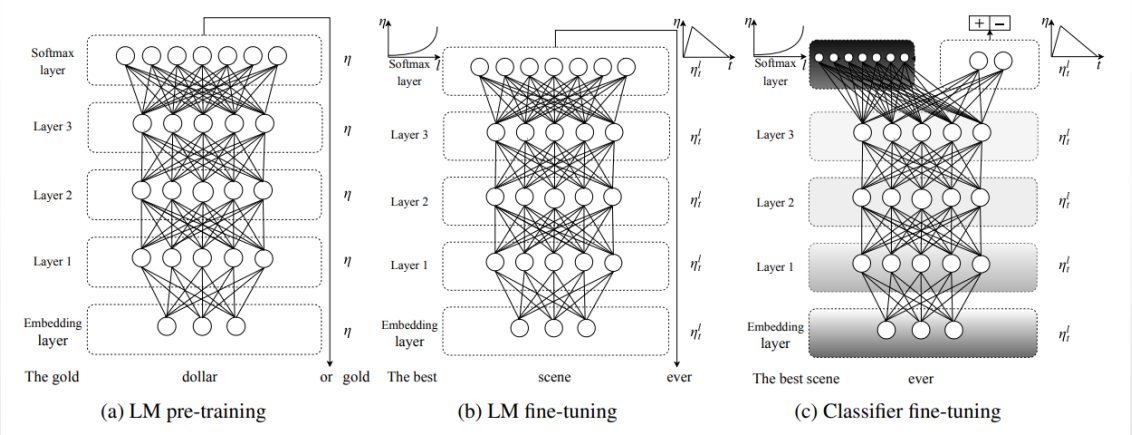
Figure 2: The slanted triangular learning rate schedule used for ULMFiT as a function of the number of training iterations.

- **Discriminative fine-tuning:** Instead of using the same learning rate for all layers of the model, discriminative fine-tuning allows us to tune each layer with different learning rates.
  - Layers that are closer to inputs → large learning rate
  - Layers that are closer to outputs → small learning rate

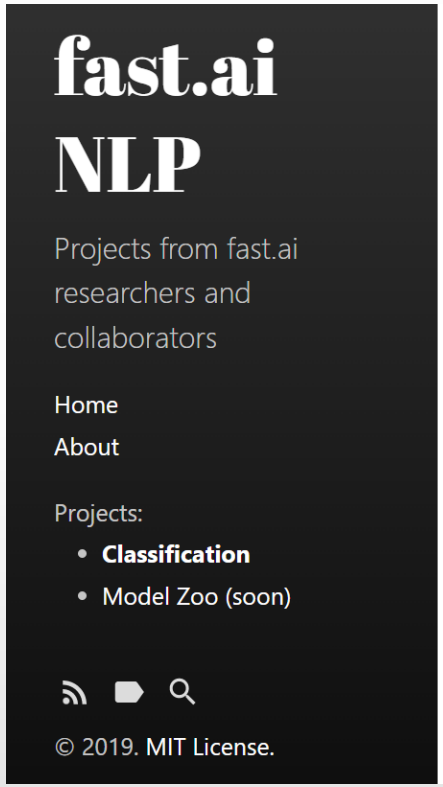
1) Pre-trained language modeling: ULMFiT

- ULMFiT requires orders of magnitude less data than previous approaches.





1) Pre-trained language modeling: ULMFiT(English)



Classification

For NLP classification the current state of the art approach is *Universal Language Model Fine-tuning* (ULMFiT). ULMFiT is an effective transfer learning method that can be applied to any task in NLP, but at this stage we have only studied its use in classication tasks. The approach is described and analyzed in the Universal Language Model Fine-tuning for Text Classification paper by fast.ai's Jeremy Howard and Sebastian Ruder from the NUI Galway Insight Centre.

To learn to use ULMFiT and access the open source code we have provided, see the following resources:

- ULMFiT is discussed in depth in lesson 10 of fast.ai's Cutting Edge Deep Learning for Coders. A gentler introduction is available in lesson 4 of Practical Deep Learning for Coders
- The fastai library provides modules necessary to train and use ULMFiT models. In particular, you will want to use `fastai.text` and `fastai.lm_rnn`
- The scripts used for the ULMFiT paper are available in the imdb_scripts folder in the fastai repository.
- The pre-trained Wikitext 103 model and vocab are available here
- The paper and code are being discussed in the fast.ai discussion forums. Feel free to join the discussion!

https://nlp.fast.ai/category/classification.html

Efficient multi-lingual language model fine-tuning

Written: 10 Sep 2019 by *Sebastian Ruder and Julian Eisenschlos* • Classification

https://nlp.fast.ai/

1) Pre-trained language modeling: ULMFiT(Thai)

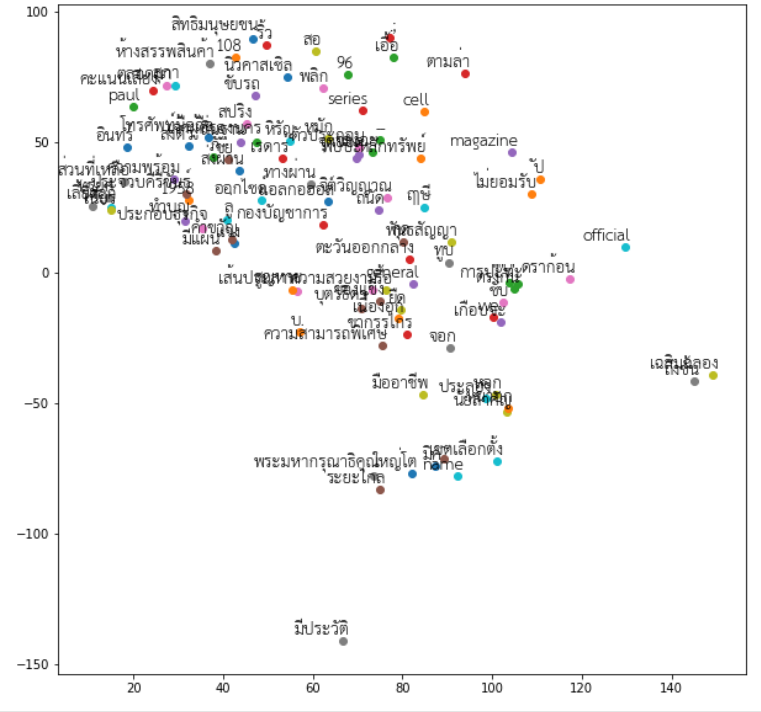
README.md

thai2fit (formerly thai2vec)

ULMFit Language Modeling, Text Feature Extraction and Text Classification in Thai Language. Created as part of pyThaiNLP with ULMFit implementation from fast.ai

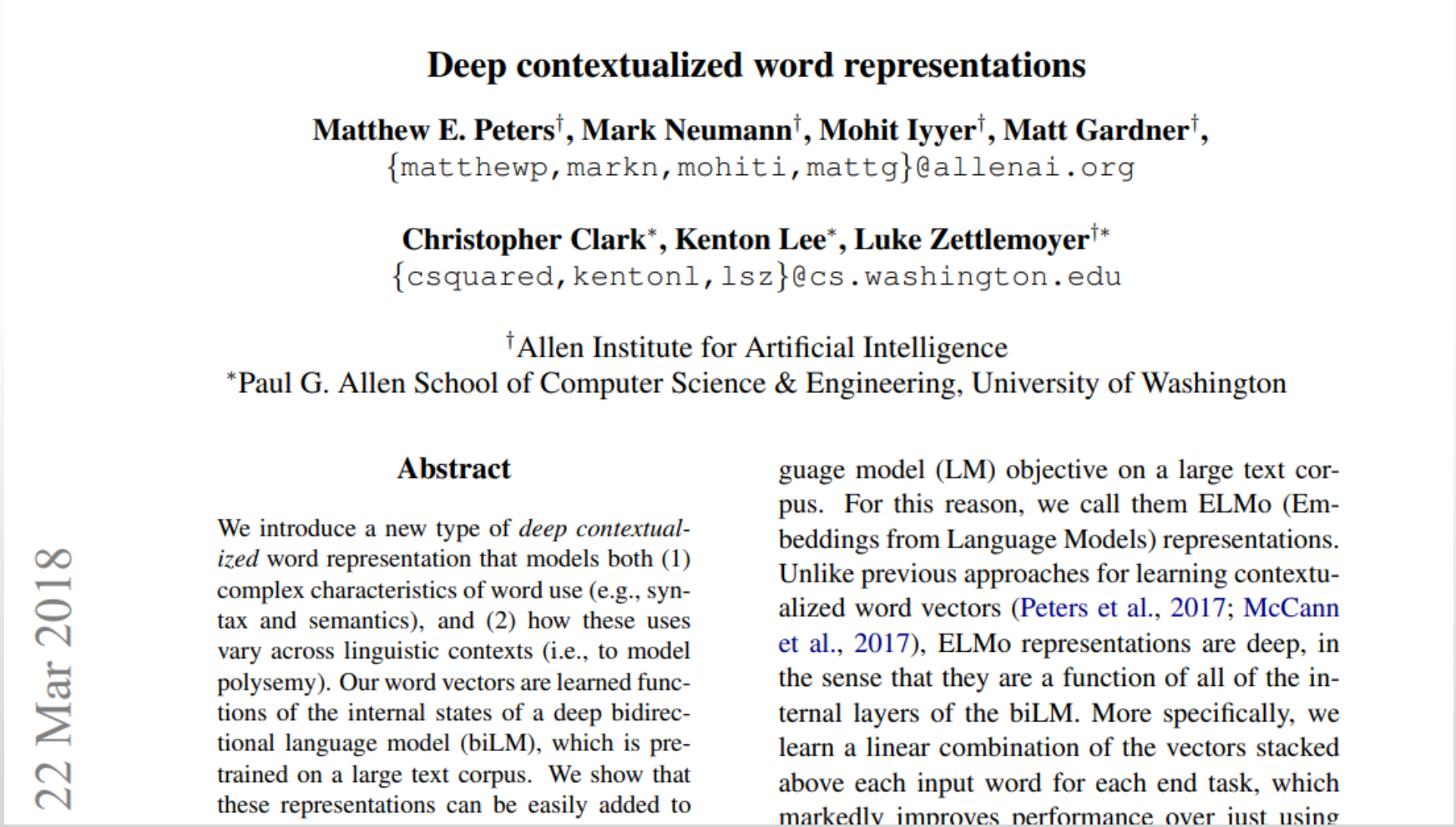
Models and word embeddings can also be downloaded via Dropbox.

We pretrained a language model with 60,005 embeddings on Thai Wikipedia Dump (perplexity of 28.71067) and text classification (micro-averaged F-1 score of 0.60322 on 5-label classification problem. Benchmarked to 0.5109 by fastText and 0.4976 by LinearSVC on Wongnai Challenge: Review Rating Prediction. The language model can also be used to extract text features for other downstream tasks.



https://github.com/cstorm125/thai2fit

2) Pre-trained language modeling: ELMo (Embeddings from Language Models)



Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

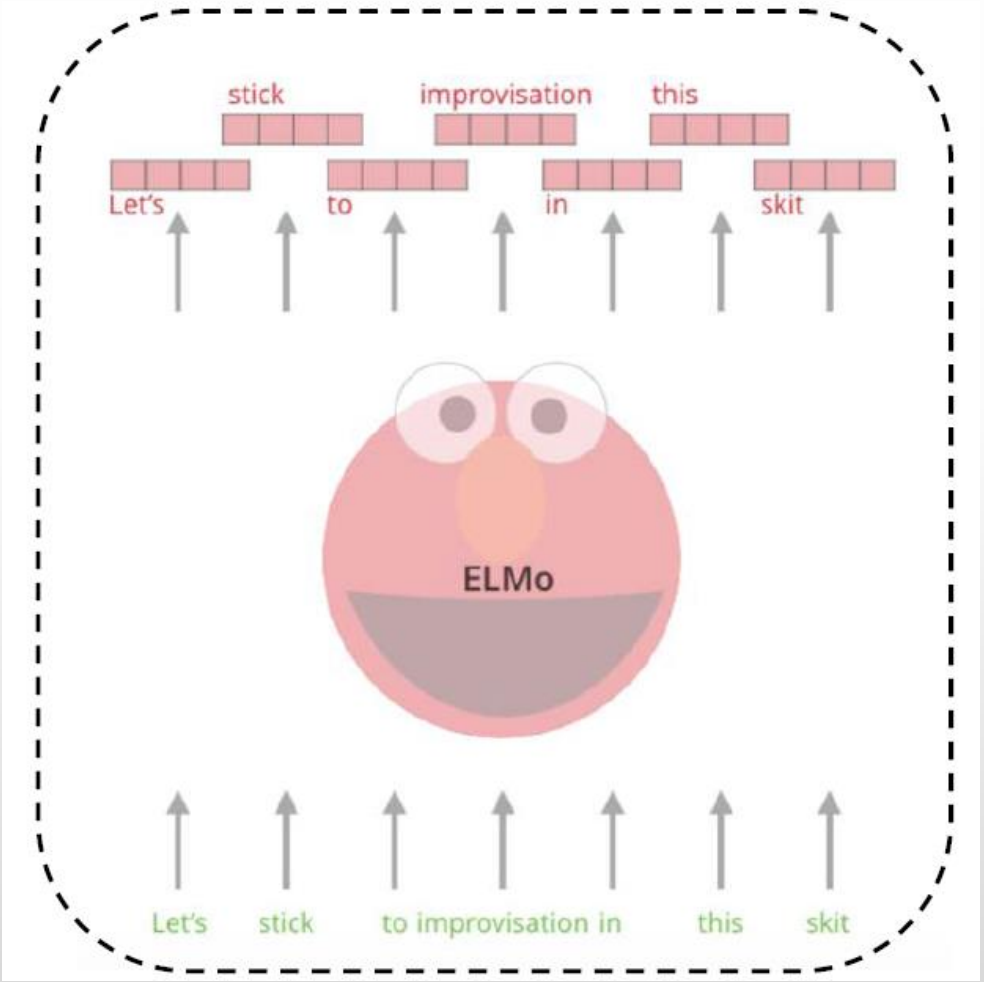
We introduce a new type of *deep contextualized* word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). Our word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. We show that these representations can be easily added to guage model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors (Peters et al., 2017; McCann et al., 2017), ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations, ArXiv Prepr. *ArXiv1802*, *5365*.

https://arxiv.org/pdf/1802.05365.pdf

2) Pre-trained language modeling: ELMo

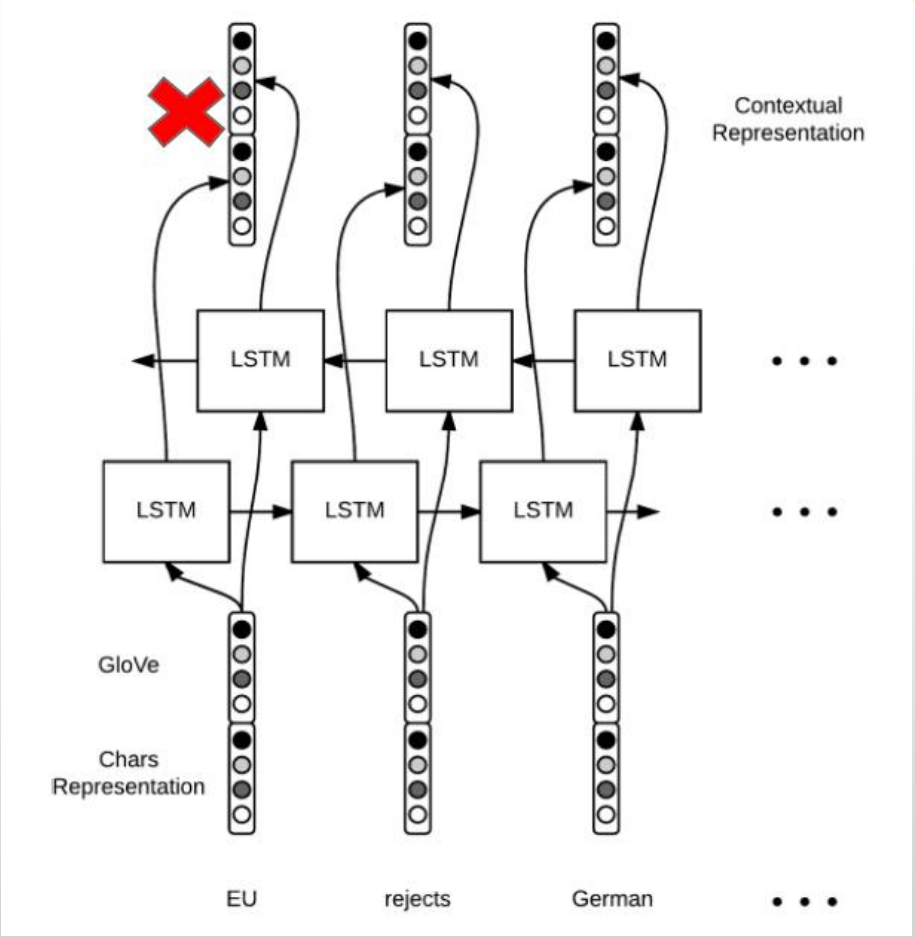
- Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.
- It uses a LSTM-based bidirectional language model (biLM) trained on a LM task to be able to create those embeddings.
- Part of ELMo representations are character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary (OOV) tokens unseen in training.



2) Pre-trained language modeling: ELMo

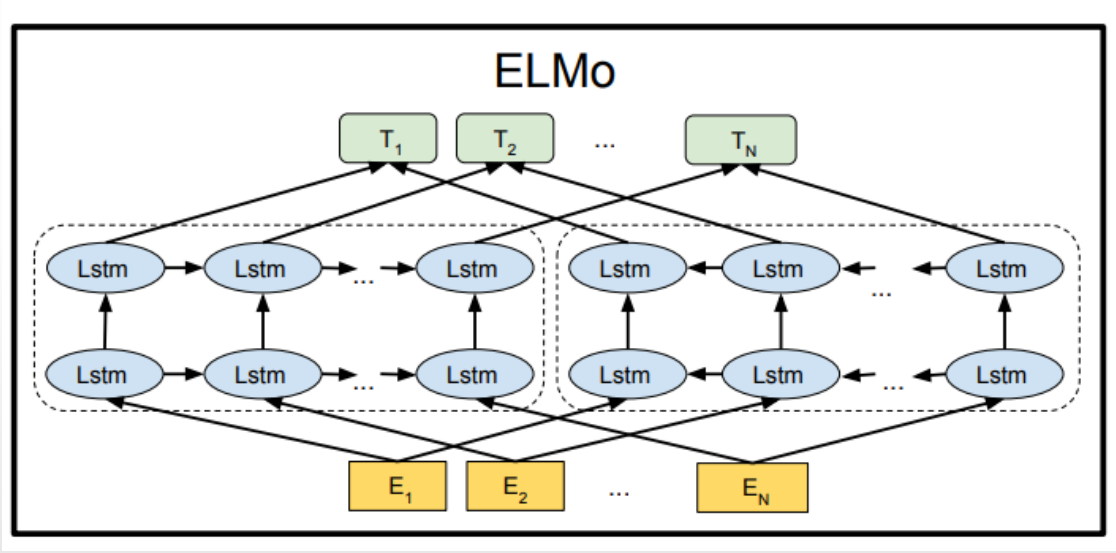
- Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.
- It uses a 2 separated LSTM-based bidirectional language model (biLM) trained on a LM task to be able to create those embeddings.
- Part of ELMo representations are character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary (OOV) tokens unseen in training.

Embedded vector from Bidirectional LSTM is not suitable for LM task since it contains future information. While in the real scenario, we can't see future word, so we cannot embed it.



2) Pre-trained language modeling: ELMo

- Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.
- It uses a 2 separated LSTM-based bidirectional language model (biLM) trained on a LM task to be able to create those embeddings.
- Part of ELMo representations are character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary (OOV) tokens unseen in training.



2) Pre-trained language modeling: ELMo

- Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.
- It uses a 2 separated LSTM-based bidirectional language model (biLM) trained on a LM task to be able to create those embeddings.
  - Not concat vector
  - Combine losses from 2 unidirectional LMs for backpropagation
- Part of ELMo representations are character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary (OOV) tokens unseen in training.

General Bidirectional Language Model

Forward LM

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

Backward LM

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

Joint Maximum Likelihood

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

2) Pre-trained language modeling: ELMo

- Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.
- It uses a 2 separated LSTM-based bidirectional language model (biLM) trained on a LM task to be able to create those embeddings.
- Part of ELMo representations are character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary (OOV) tokens unseen in training.

ELMo Representation

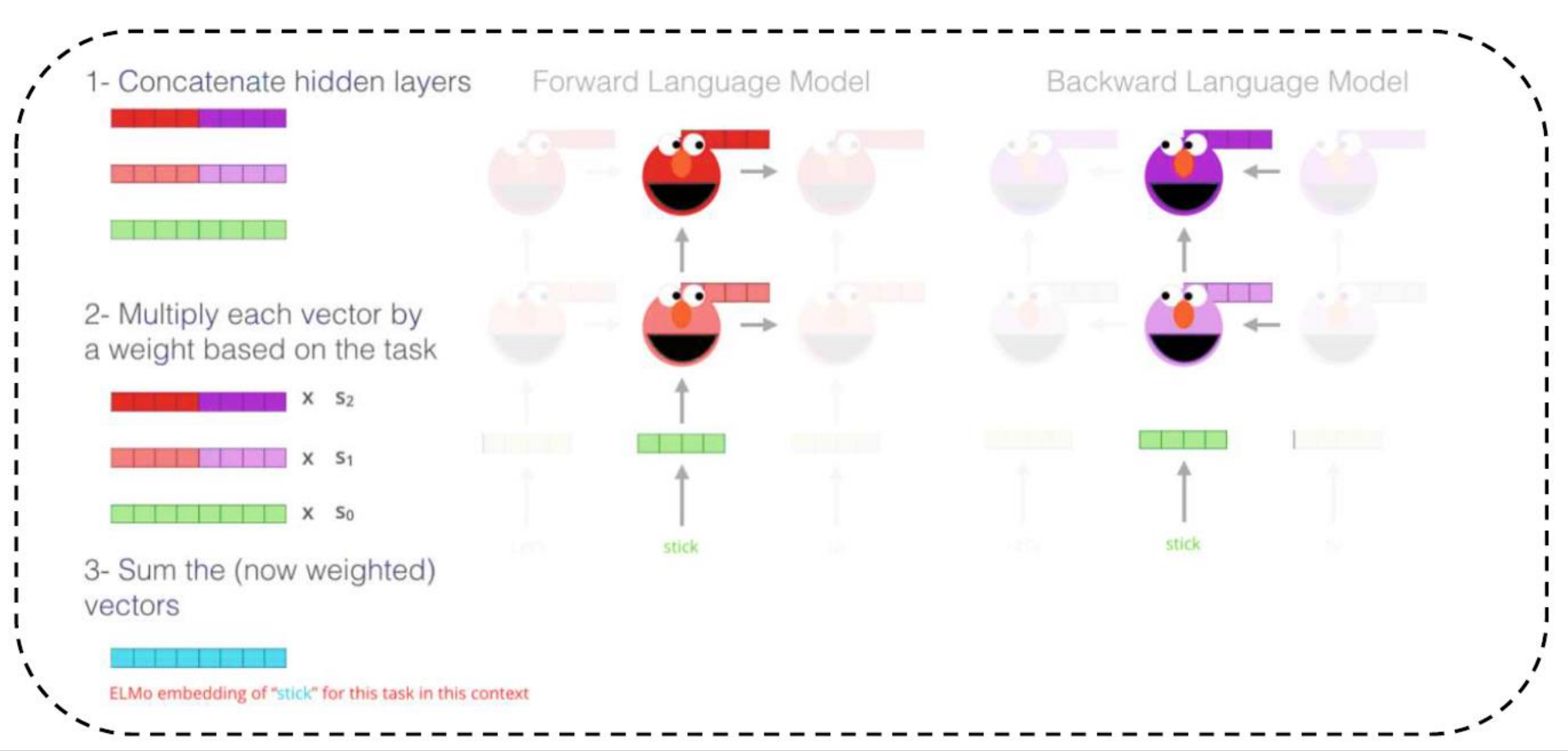
For each token t_k, a L-layer biLM computes a set of 2L+1 representations. Where $\mathbf{x}_k$ is a context-independent token representation layer and $\mathbf{h}_{k,j}$ is the **concatenation forward** and **backward context-dependent** representation for each biLSTM layer.

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

Note: $\mathbf{x}_k$ can be either token representation or a CNN over characters. **ELMo uses a character-based CNN representations** (Peters et al., 2017).

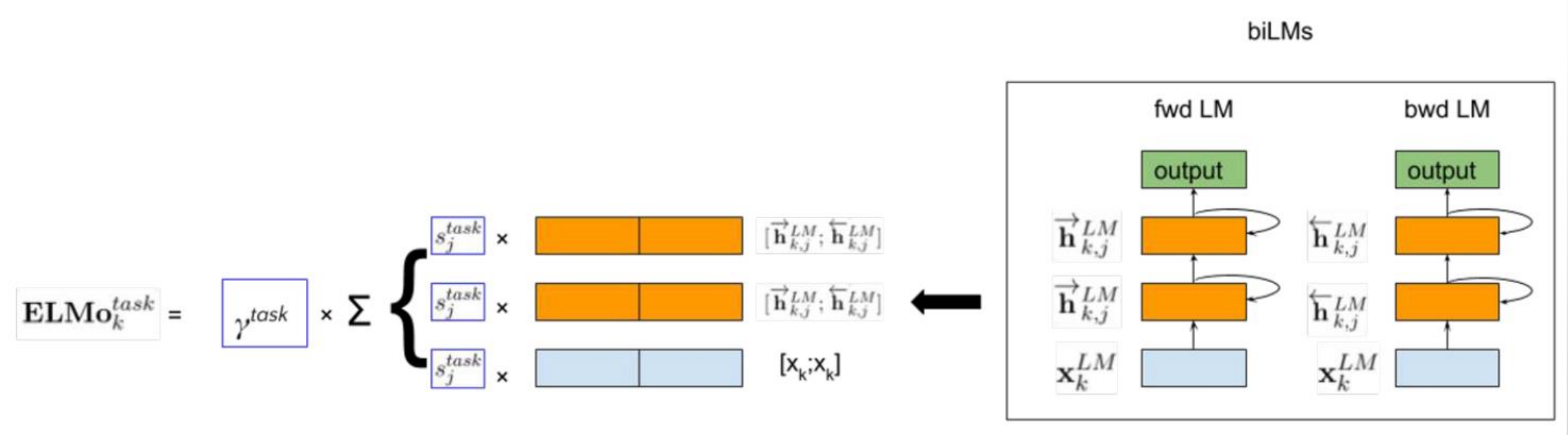
2) Pre-trained language modeling: ELMo

- Embedding of “stick” in “Let’s stick to” (word-level task)



2) Pre-trained language modeling: ELMo

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



2) Pre-trained language modeling: ELMo

AllenNLP > Software

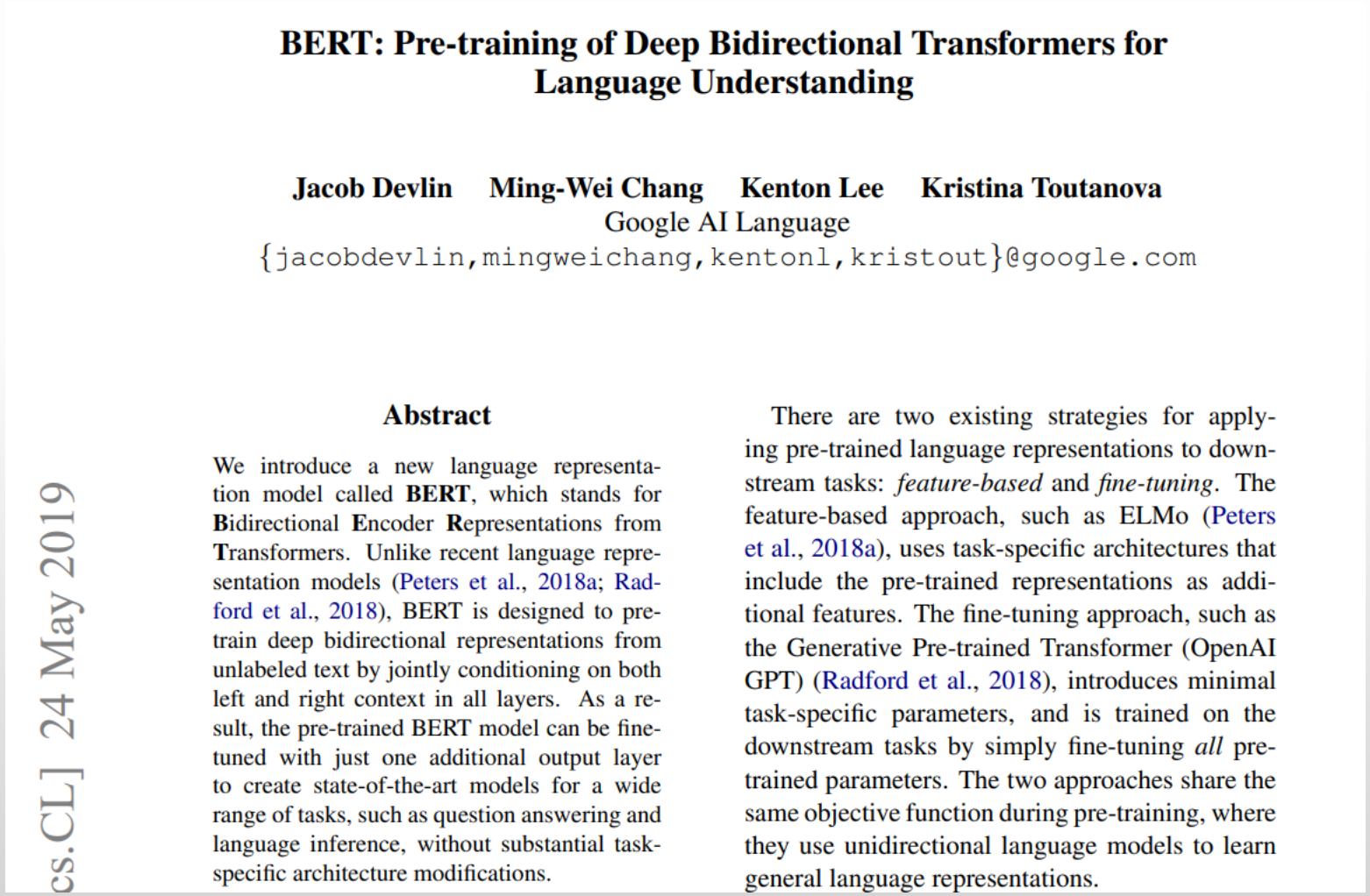
AI2 Tango AllenNLP Library All Software

Pre-trained ELMo Models

Model	Link (Weights/Options File)	# Parameters (Millions)	LSTM Hidden Size/Output size	# Highway Layers	SRL F1	Constituency Parsing F1
Small	weights | options	13.6	1024/128	1	83.62	93.12
Medium	weights | options	28.0	2048/256	1	84.04	93.60
Original	weights | options	93.6	4096/512	2	84.63	93.85
Original (5.5B)	weights | options	93.6	4096/512	2	84.93	94.01

https://allenai.org/allennlp/software/elmo

3) Pre-trained language modeling: BERT
(Bidirectional Encoder Representation from Transformers)



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

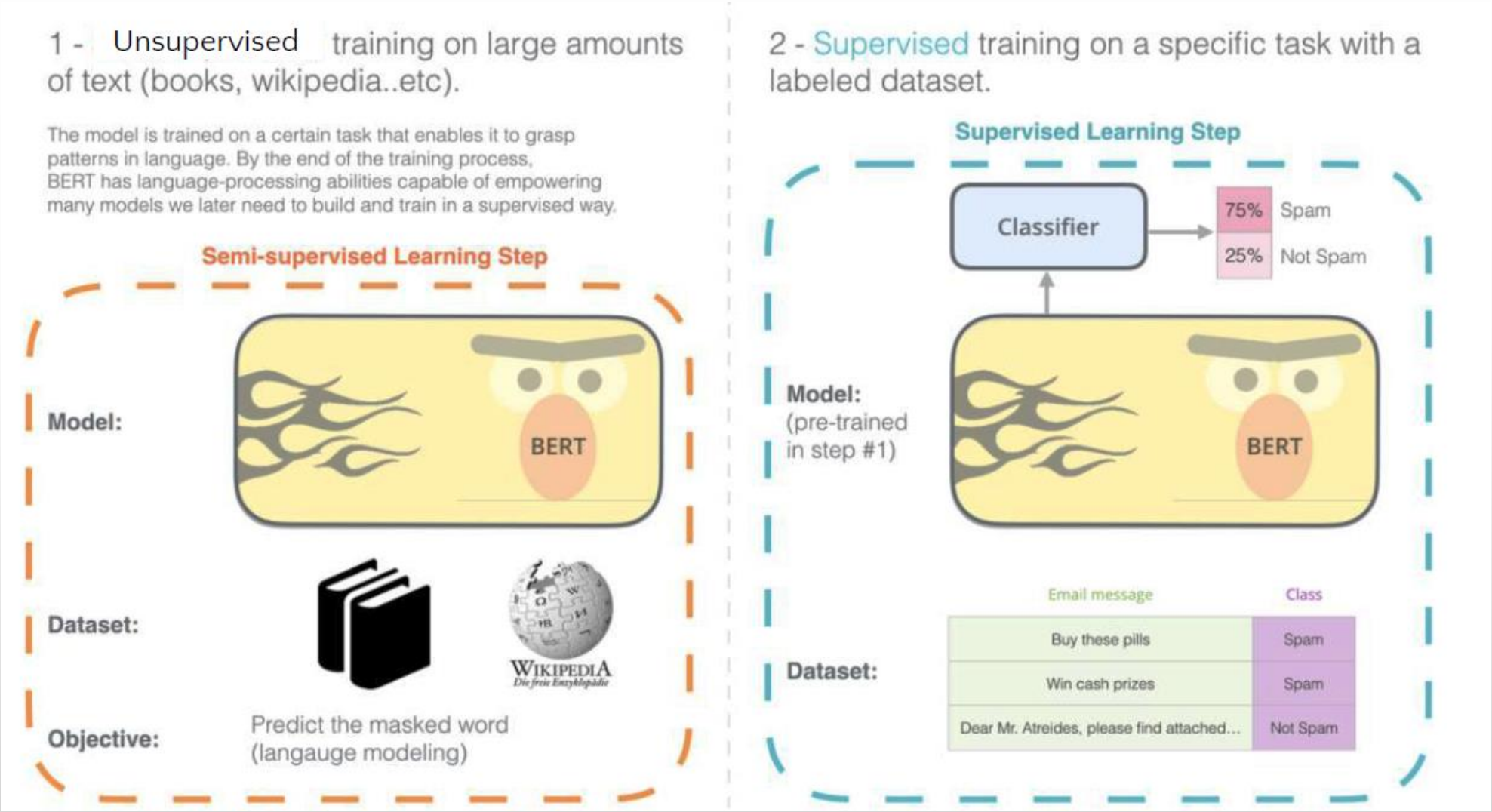
We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

https://arxiv.org/pdf/1810.04805.pdf

3) Pre-trained language modeling: BERT



3) Pre-trained language modeling: BERT

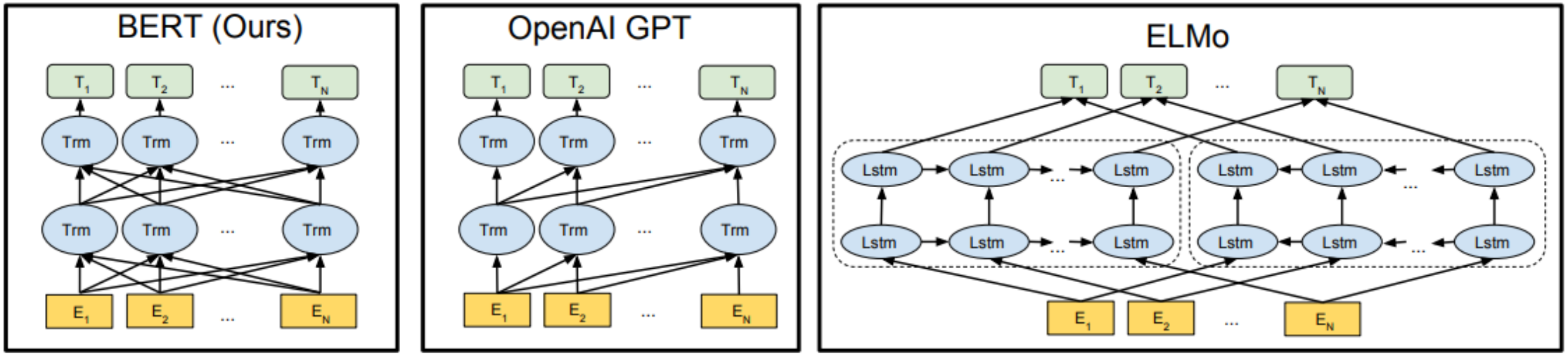


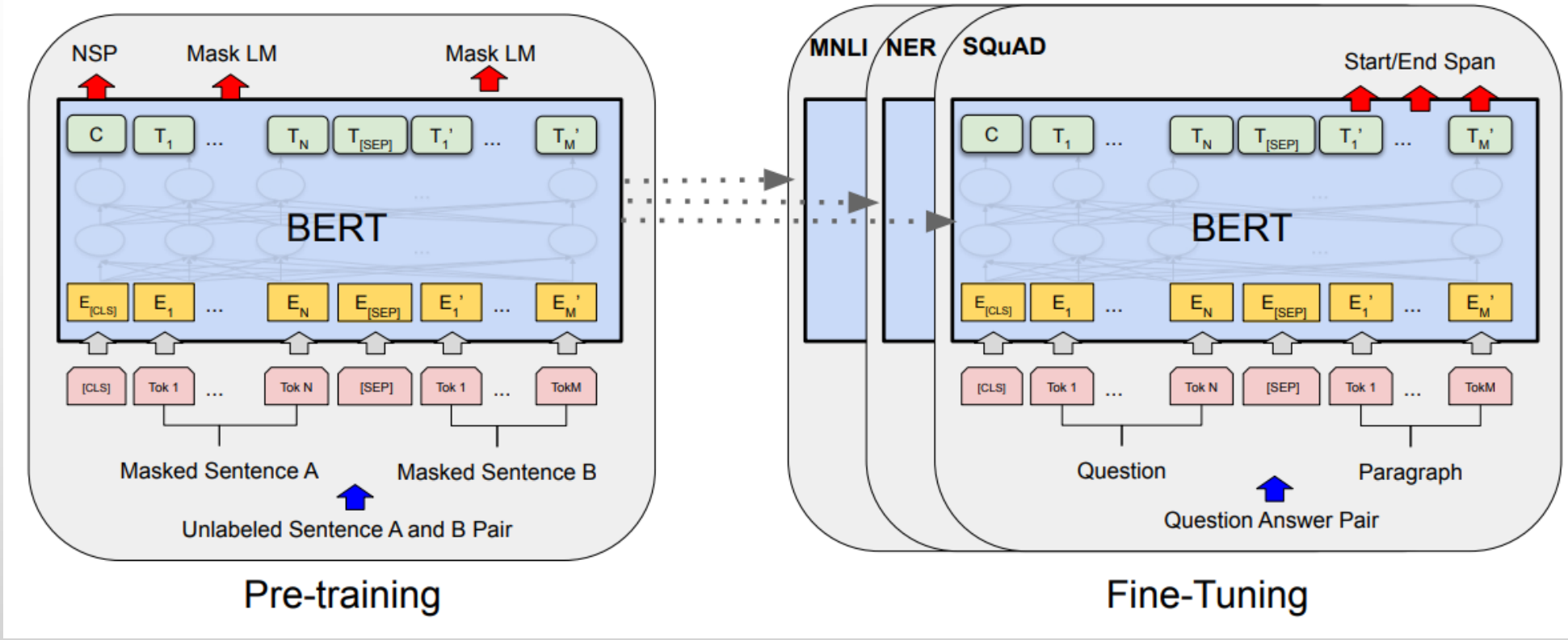
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

3) Pre-trained language modeling: BERT

- Overall idea

Phase1: Semi-Unsupervised Learning
- Mask LM
- Next sentence prediction

Phase2: Supervised learning
- E.g. Finetune on QA, Text classification, etc.



3) Pre-trained language modeling: BERT

- **Phase1: Unsupervised Phase**
- Semi-supervised training, using 2 prediction tasks:
  - Mask language modeling
    - represents the word using both its left and right context, so called deeply bidirectional.
    - Mask out 15% of the words in the input, run the entire sequence through a deep bidirectional encoder, and then predict only the masked words.

```
Input: the man went to the [MASK1] . he bought a [MASK2] of milk.
Labels: [MASK1] = store; [MASK2] = gallon
```

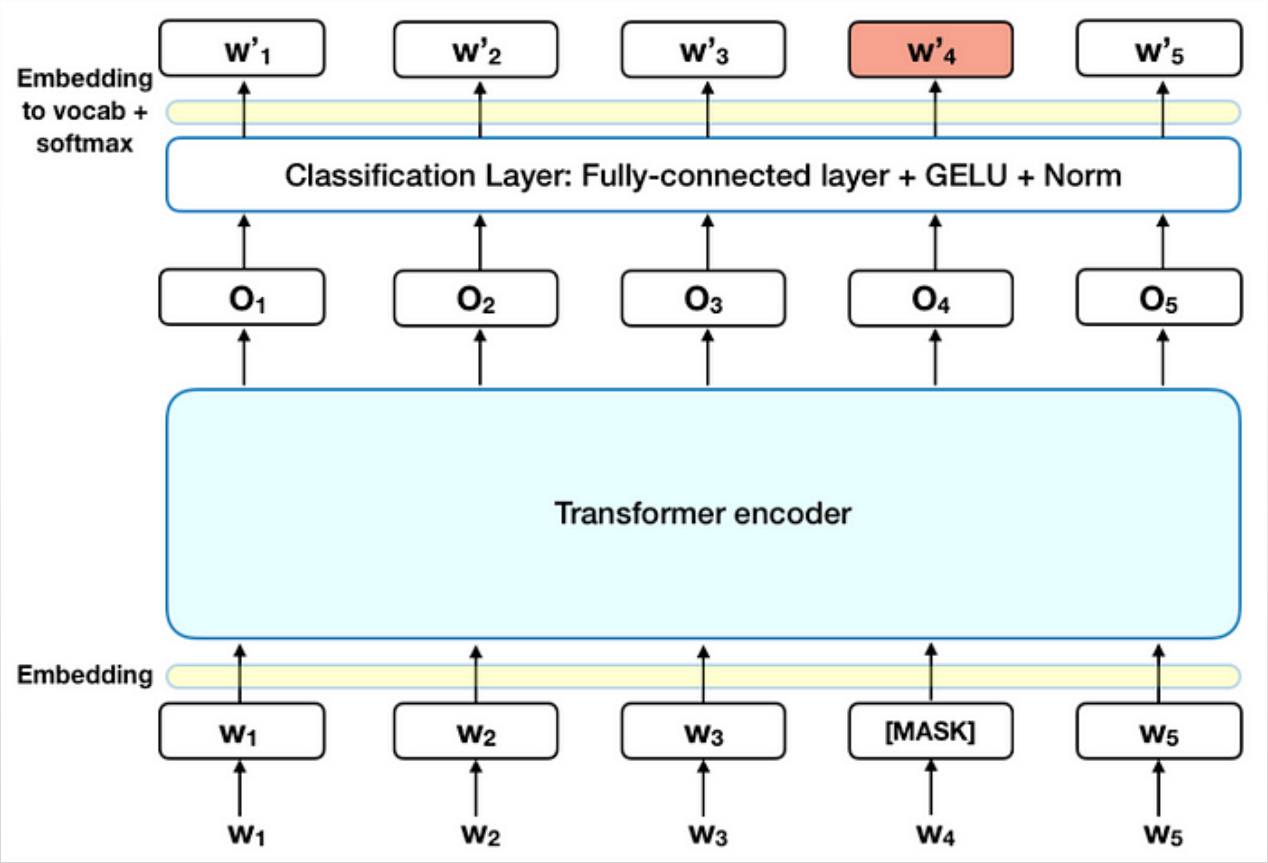
  - Next sentence prediction (NSP)
    - to learn relationships between sentences.

```
Sentence A: the man went to the store .
Sentence B: he bought a gallon of milk .
Label: IsNextSentence
```

```
Sentence A: the man went to the store .
Sentence B: penguins are flightless .
Label: NotNextSentence
```

3) Pre-trained language modeling: BERT

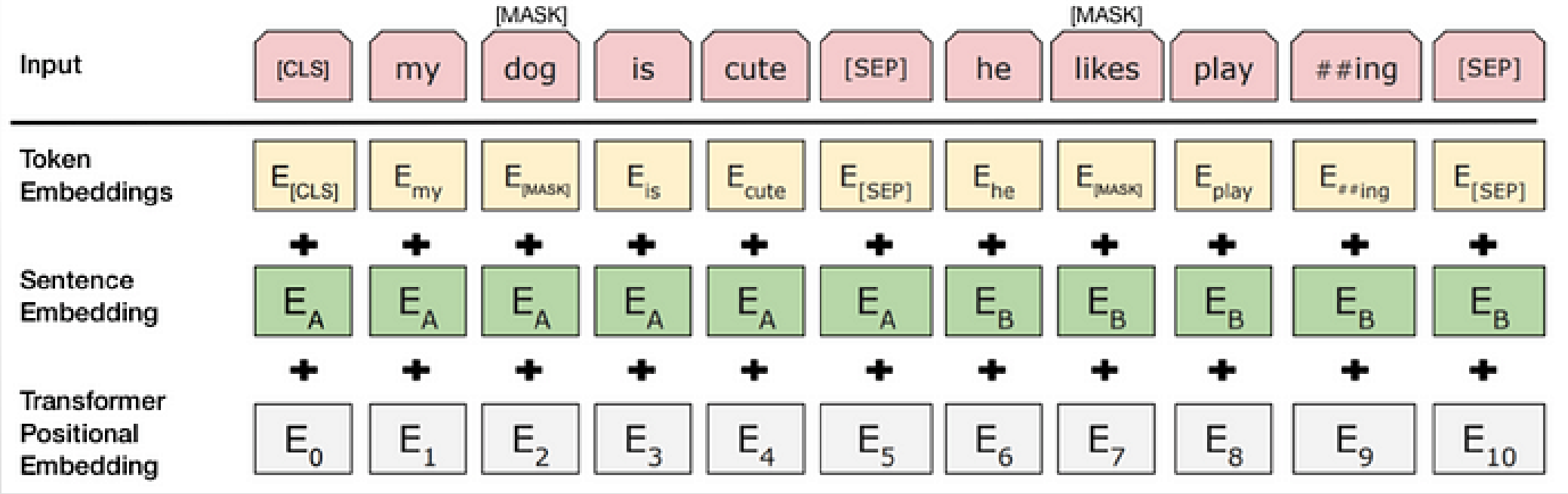
- **Phase1: Unsupervised Phase**
- Mask language modeling:



https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

3) Pre-trained language modeling: BERT

- **Phase1: Unsupervised Phase**
- Next Sentence Prediction

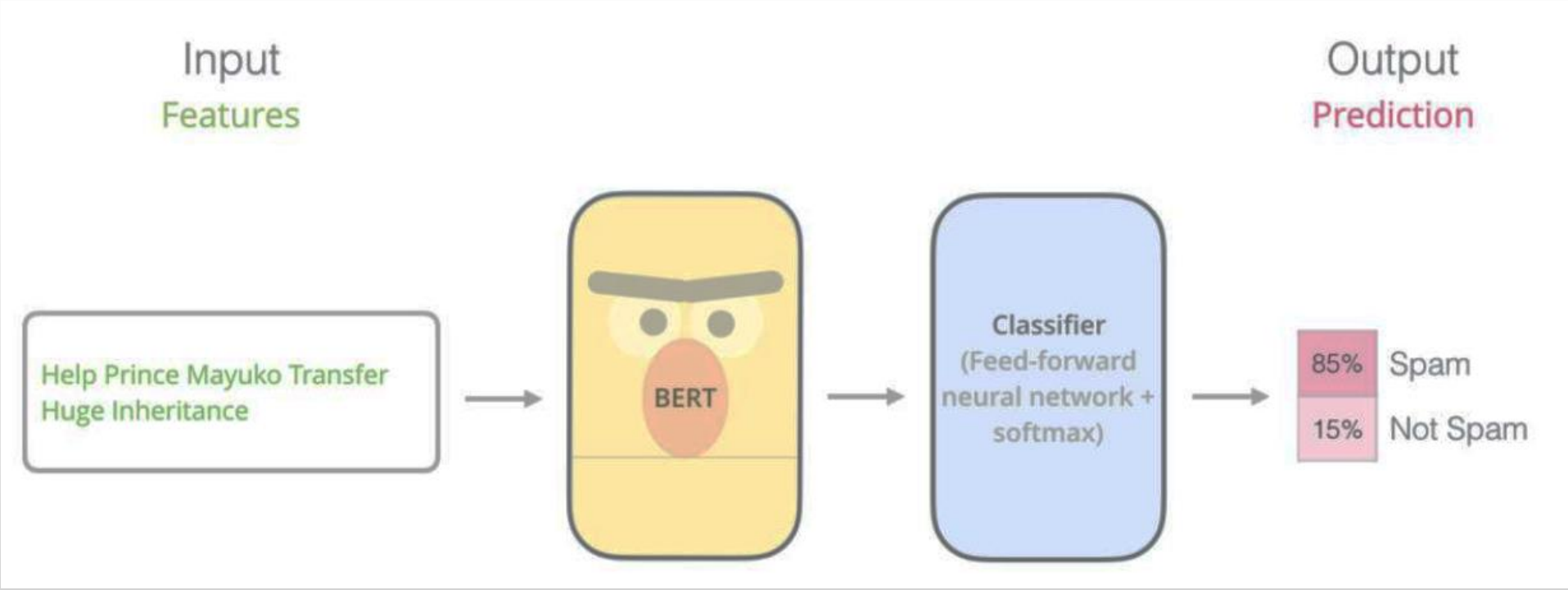


CLS = Classification, which is used for NSP (Next Sentence Prediction) during this phase
SEP = a special separator token (e.g. separating questions/answers).

https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

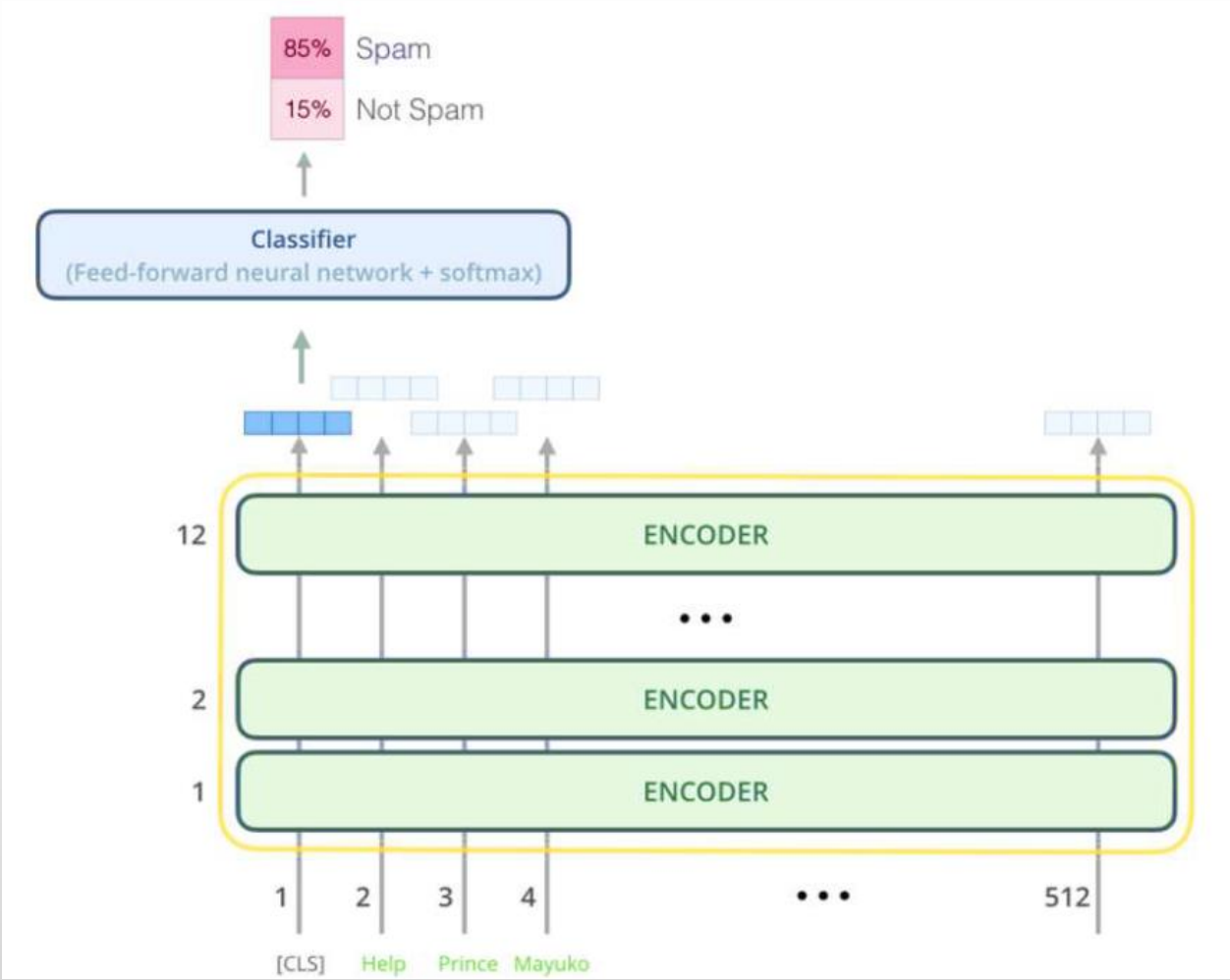
3) Pre-trained language modeling: BERT

- **Phase2: Supervised Phase**
- Supervised training (e.g. Email sentence classification)
  - you mainly have to train the classifier, with minimal changes happening to the pre-trained model during the training phase (fine-tuning approach).



3) Pre-trained language modeling: BERT

- E.g.,Spam Email Classification
  - BERT is basically a trained Transformer Encoder stack.
  - The first input token is supplied with a special (classification embedding) [CLS] token for classification task to represent the entire sentence.
  - The output is generated from only this special [CLS] token.



3) Pre-trained language modeling: BERT

- E.g., other tasks

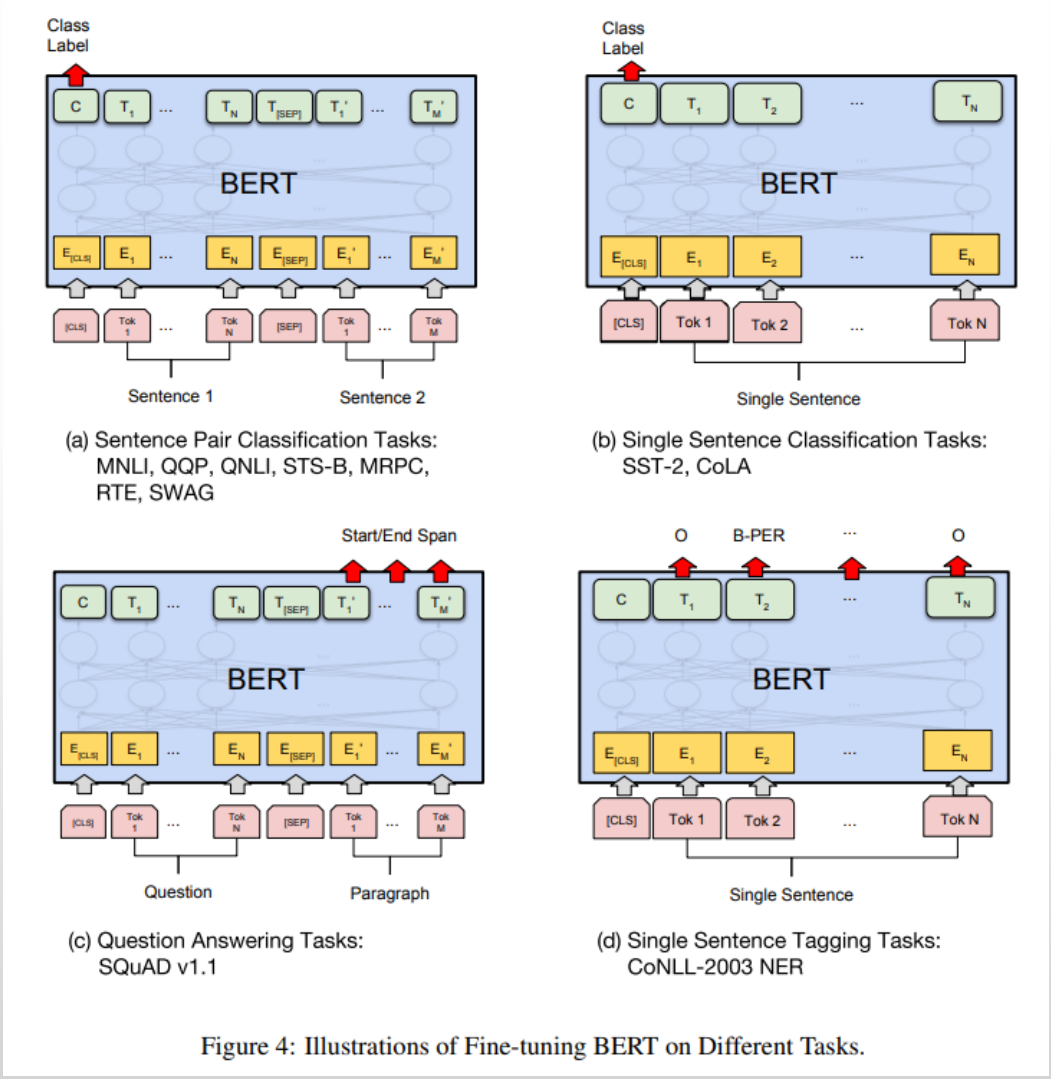


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

3) Pre-trained language modeling: BERT (English)

BERT

***** New March 11th, 2020: Smaller BERT Models *****

This is a release of 24 smaller BERT models (English only, uncased, trained with WordPiece masking) referenced in Well-Read Students Learn Better: On the Importance of Pre-training Compact Models.

We have shown that the standard BERT recipe (including model architecture and training objective) is effective on a wide range of model sizes, beyond BERT-Base and BERT-Large. The smaller BERT models are intended for environments with restricted computational resources. They can be fine-tuned in the same manner as the original BERT models. However, they are most effective in the context of knowledge distillation, where the fine-tuning labels are produced by a larger and more accurate teacher.

Our goal is to enable research in institutions with fewer computational resources and encourage the community to seek directions of innovation alternative to increasing model capacity.

You can download all 24 from here, or individually from the table below:

	H=128	H=256	H=512	H=768
L=2	**2/128 (BERT-Tiny)**	2/256	2/512	2/768
L=4	4/128	**4/256 (BERT-Mini)**	**4/512 (BERT-Small)**	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	**8/512 (BERT-Medium)**	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	**12/768 (BERT-Base)**

https://github.com/google-research/bert

3) Pre-trained language modeling: BERT (Multilingual)

Models

There are two multilingual models currently available. We do not plan to release more single-language models, but we may release `BERT-Large` versions of these two in the future:

- `BERT-Base, Multilingual Cased (New, recommended)`: 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)`: 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese`: Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

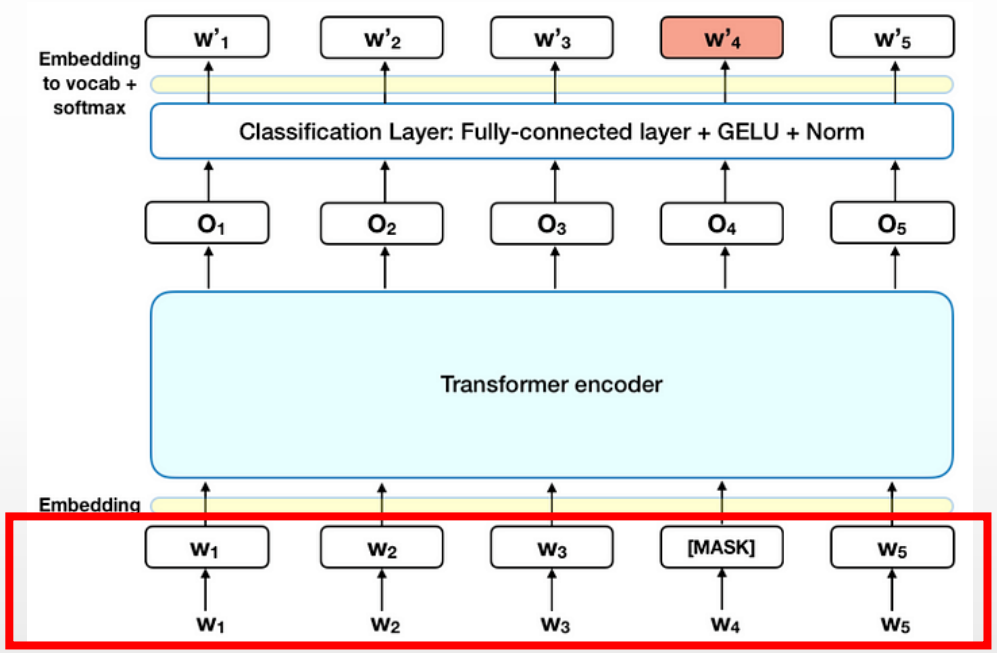
The `Multilingual Cased (New)` model also fixes normalization issues in many languages, so it is recommended in languages with non-Latin alphabets (and is often better for most languages with Latin alphabets). When using this model, make sure to pass `--do_lower_case=false` to `run_pretraining.py` and other scripts.

See the list of languages that the Multilingual model supports. The Multilingual model does include Chinese (and English), but if your fine-tuning data is Chinese-only, then the Chinese model will likely produce better results.

https://github.com/google-research/bert/blob/master/multilingual.md

3) Pre-trained language modeling: Roberta
(Robustly optimized BERT approach)

- A trick and tuning study
- Dynamic masking > static
- Next sentence prediction is not optimal
- Larger batch + higher learning rate

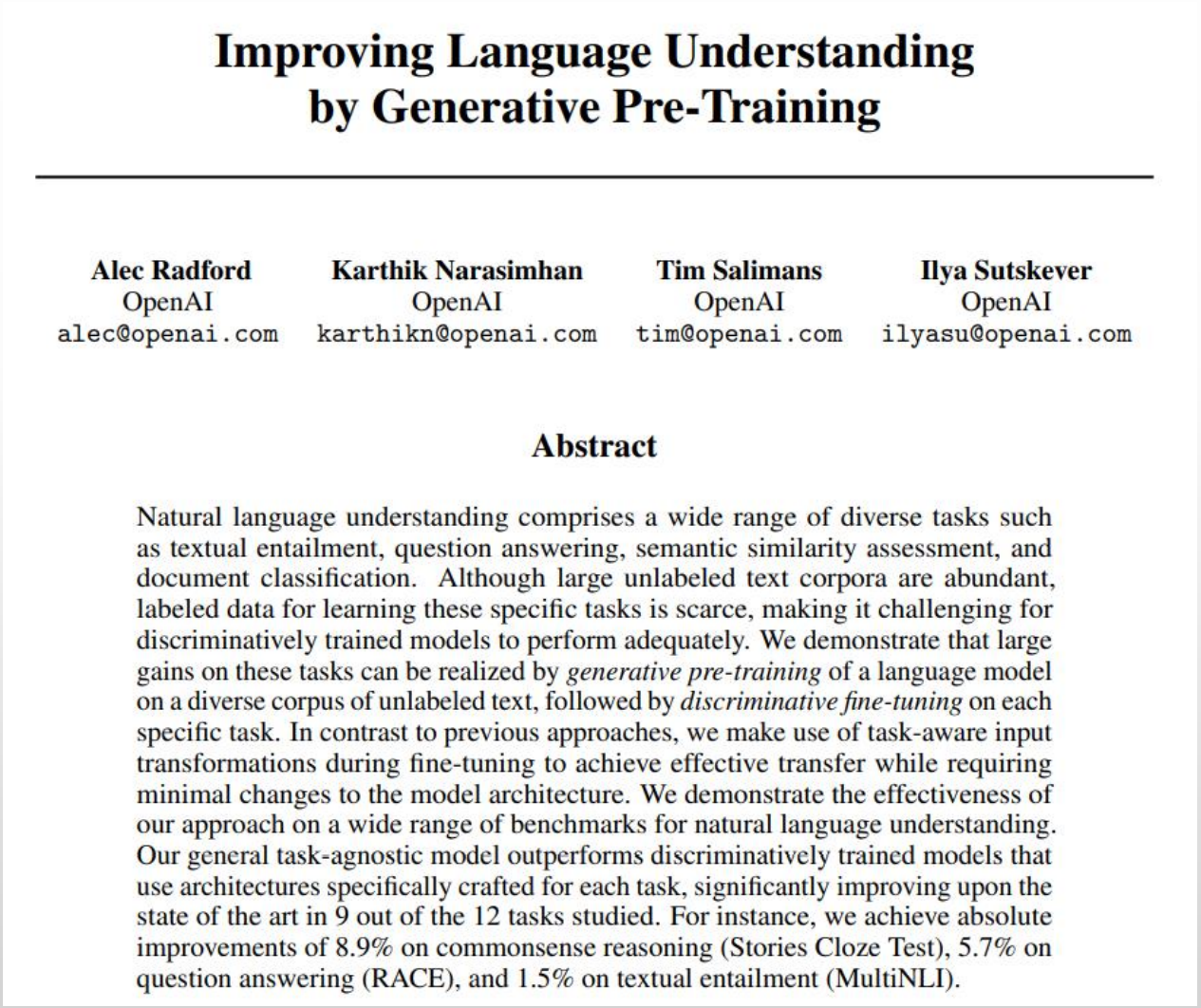


static

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

4) Pre-trained language modeling: GPT (Generative Pre-Training)



Improving Language Understanding by Generative Pre-Training

Alec Radford OpenAI alec@openai.com

Karthik Narasimhan OpenAI karthikn@openai.com

Tim Salimans OpenAI tim@openai.com

Ilya Sutskever OpenAI ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf

4) Pre-trained language modeling: GPT

- GPT training procedure consists of 2 steps:
  - 1) Unsupervised pre-training: (Unidirectional LM using Transformer)
    - Given an unsupervised corpus of tokens $U = \{u_1, \ldots, u_n\}$, use a standard language modeling objective to maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

    - where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ.
  - 2) Supervised fine-tuning:
    - We assume a labeled dataset C, where each instance consists of a sequence of input tokens, $x_1, \ldots, x_m$, along with a label y. This gives us the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y | x^1, \ldots, x^m).$$

    - Include language modeling as an auxiliary objective to the fine-tuning (L2 + L1):

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

Auxiliary task (LM)
Multitask; add this loss

4) Pre-trained language modeling: GPT

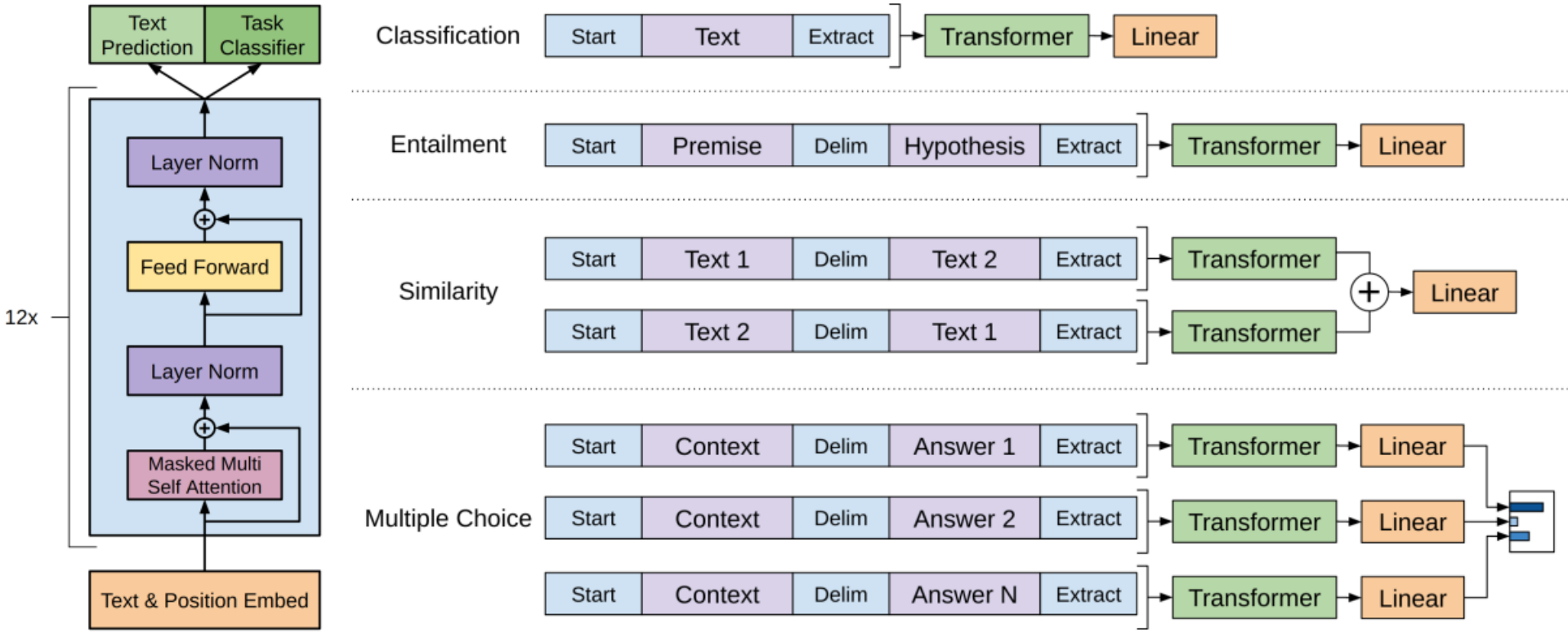
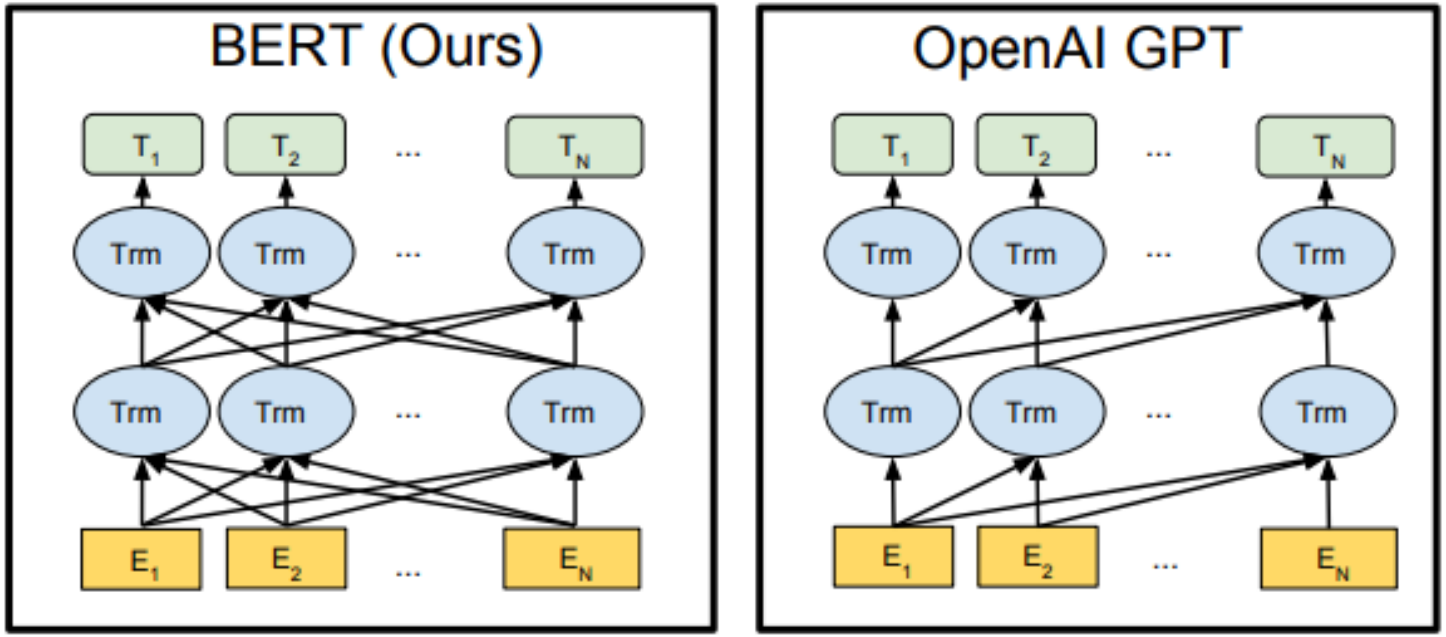


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

4) Pre-trained language modeling: GPT

- GPT vs BERT



Different points:

- BERT (unsupervised phase)
- Masked LM vs Unidirect LM
- Next sentence prediction
- GPT (supervised learning)
- Auxiliary task

4) Pre-trained language modeling: GPT

- GPT-2 (2019) & GPT-3 (2020)
- GPT-2 is a large transformer-based language model with 1.5 billion parameters
  - GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data.
  - Trained on 8 million webpages (scraped from outbound links from Reddit with at least 3 karma points)
  - State-of-the-art LM
  - On other language tasks like question answering, reading comprehension, summarization, and translation, GPT-2 learns these tasks from the raw text, using no task-specific training data. (zero-shot setting)
- The architecture of GPT-3 is same as GPT-2. Few major differences from GPT-2 are:
  - GPT-3 has 96 layers with each layer having 96 attention heads.
  - Size of word embeddings was increased to 12,888 for GPT-3 from 1600 for GPT-2.
  - Context window size was increased from 1024 for GPT-2 to 2048 tokens for GPT-3.
  - Adam optimizer was used with $\beta_1 = 0.9, \beta_2 = 0.95$ and $\varepsilon = 10^{-8}$.
  - Alternating dense and locally banded sparse attention patterns were used.

https://openai.com/research/better-language-models
https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2#:~:text=Model%20and%20Implementation%20details%3A%20The

Transformers by HuggingFace

🤗 Transformers

State-of-the-art Machine Learning for PyTorch, TensorFlow, and JAX.

🤗 Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

📝 **Natural Language Processing**: text classification, named entity recognition, question answering, language modeling, summarization, translation, multiple choice, and text generation.
🖼️ **Computer Vision**: image classification, object detection, and segmentation.
🗣️ **Audio**: automatic speech recognition and audio classification.
🐙 **Multimodal**: table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

https://huggingface.co/docs/transformers/index#supported-models

7. BARTpho (from VinAI Research) released with the paper BARTpho: Pre-trained Sequence-to-Sequence Models for Vietnamese by Nguyen Luong Tran, Duong Minh Le and Dat Quoc Nguyen.
8. BEiT (from Microsoft) released with the paper BEiT: BERT Pre-Training of Image Transformers by Hangbo Bao, Li Dong, Furu Wei.
9. 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.
10. BERT For Sequence Generation (from Google) released with the paper Leveraging Pre-trained Checkpoints for Sequence Generation Tasks by Sascha Rothe, Shashi Narayan, Aliaksei Severyn.
11. BERTweet (from VinAI Research) released with the paper BERTweet: A pre-trained language model for English Tweets by Dat Quoc Nguyen, Thanh Vu and Anh Tuan Nguyen.
12. BigBird-Pegasus (from Google Research) released with the paper Big Bird: Transformers for Longer Sequences by Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, Amr Ahmed.

Lijuan Wang.

69. GLPN (from KAIST) released with the paper Global-Local Path Networks for Monocular Depth Estimation with Vertical CutDepth by Doyeon Kim, Woonghyun Ga, Pyungwhan Ahn, Donggyu Joo, Sehwan Chun, Junmo Kim.
70. GPT (from OpenAI) released with the paper Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever.
71. GPT Neo (from EleutherAI) released in the repository EleutherAI/gpt-neo by Sid Black, Stella Biderman, Leo Gao, Phil Wang and Connor Leahy.
72. GPT NeoX (from EleutherAI) released with the paper GPT-NeoX-20B: An Open-Source Autoregressive Language Model by Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, Samuel Weinbach
73. GPT NeoX Japanese (from ABEJA) released by Shinya Otani, Takayoshi Makabe, Anuj Arora, and Kyo Hattori.
74. 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**.

5) Pre-trained language modeling: Flair
(Contextual string embedding for sequence labeling)

Contextual String Embeddings for Sequence Labeling

Alan Akbik
Zalando Research
Mühlenstraße 25
10243 Berlin

Duncan Blythe
Zalando Research
Mühlenstraße 25
10243 Berlin

Roland Vollgraf
Zalando Research
Mühlenstraße 25
10243 Berlin

{firstname.lastname}@zalando.de

Abstract

Recent advances in language modeling using recurrent neural networks have made it viable to model language as distributions over characters. By learning to predict the next character on the basis of previous characters, such models have been shown to automatically internalize linguistic concepts such as words, sentences, subclauses and even sentiment. In this paper, we propose to leverage the internal states of a trained character language model to produce a novel type of word embedding which we refer to as *contextual string embeddings*. Our proposed embeddings have the distinct properties that they (a) are trained without any explicit notion of words and thus fundamentally model words as sequences of characters, and (b) are *contextualized* by their surrounding text, meaning that the same word will have different embeddings depending on its contextual use. We conduct a comparative evaluation against previous embeddings and find that our embeddings are highly useful for downstream tasks: across four classic sequence labeling tasks we consistently outperform the previous state-of-the-art. In particular, we significantly outperform previous work on English and German named entity recognition (NER), allowing us to report new state-of-the-art F1-scores on the CoNLL03 shared task.

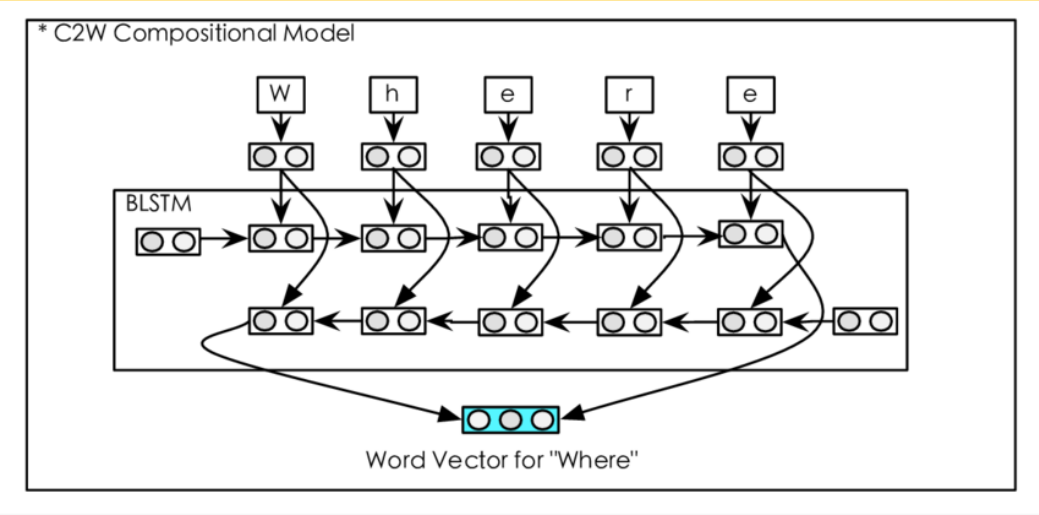
Akbik, A., Blythe, D., & Vollgraf, R. (2018, August). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th international conference on computational linguistics* (pp. 1638-1649).

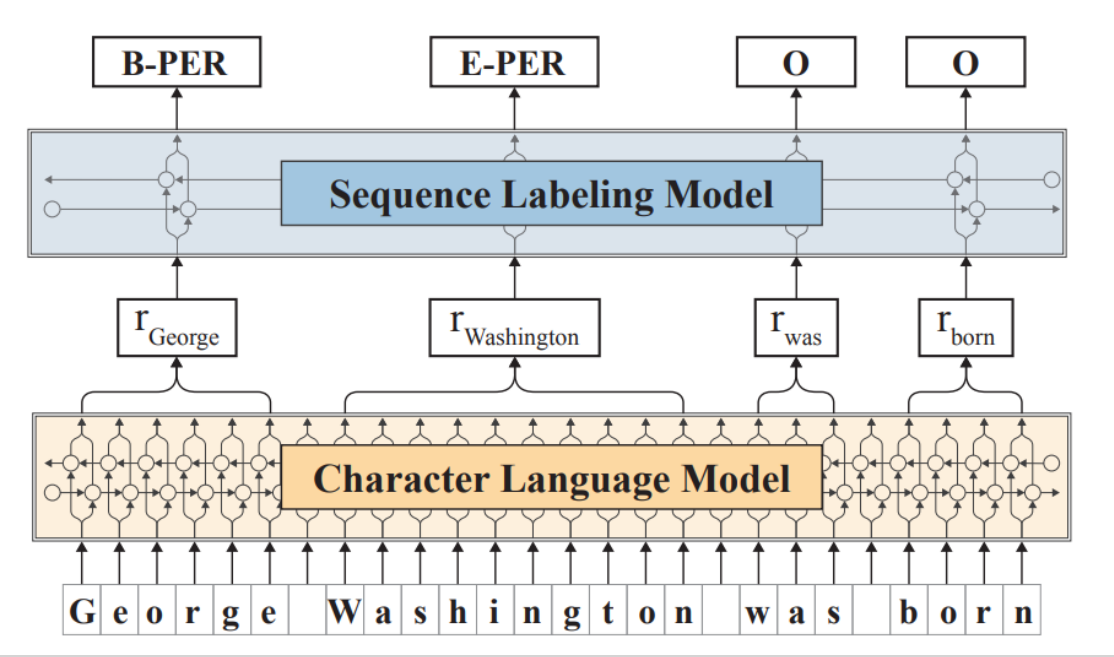
https://aclanthology.org/C18-1139.pdf

5) Pre-trained language modeling: Flair

- Contextual String Embeddings for Sequence Labeling
  - Problem: Previously, each word (string) is embedded using its own characters independent from other words in the sentence.
  - Propose: String Embeddings can be contextualized by their surrounding text.

Standard String Embeddings
Embed only that word regardless of other words in the sentence.





5) Pre-trained language modeling: Flair

Task	PROPOSED	Previous best
NER English	**93.09**±0.12	*92.22*±0.1 (Peters et al., 2018)
NER German	**88.32**±0.2	*78.76* (Lample et al., 2016)
Chunking	**96.72**±0.05	*96.37*±0.05 (Peters et al., 2017)
PoS tagging	**97.85**±0.01	*97.64* (Choi, 2016)

Table 1: Summary of evaluation results for best configuration of proposed architecture, and current best publ
The proposed approach significantly outperforms previous work on the CoNLL03 NER task for German and
slightly outperforms previous works on CoNLL2000 chunking and Penn treebank PoS tagging.

Named entity recognition

Named entity recognition (NER) is the task of tagging entities in text with their corresponding type. Approaches typically use BIO notation, which differentiates the beginning (B) and the inside (I) of entities. O is used for non-entity tokens.

Model	F1	Paper / Source	Code
CNN Large + fine-tune (Baevski et al., 2019)	93.5	Cloze-driven Pretraining of Self-attention Networks	
RNN-CRF+Flair	93.47	Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition	
CrossWeigh + Flair (Wang et al., 2019)♦	93.43	CrossWeigh: Training Named Entity Tagger from Imperfect Annotations	Official
LSTM-CRF+ELMo+BERT+Flair	93.38	Neural Architectures for Nested NER through Linearization	Official
Flair embeddings (Akbik et al., 2018)♦	93.09	Contextual String Embeddings for Sequence Labeling	Flair framework
BERT Large (Devlin et al., 2018)	92.8	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding	

http://nlpprogress.com/english/named_entity_recognition.html

5) Pre-trained language modeling: Flair

README.md

A very simple framework for **state-of-the-art NLP**. Developed by Humboldt University of Berlin and friends.

Flair is:

- **A powerful NLP library.** Flair allows you to apply our state-of-the-art natural language processing (NLP) models to your text, such as named entity recognition (NER), sentiment analysis, part-of-speech tagging (PoS), special support for biomedical data, sense disambiguation and classification, with support for a rapidly growing number of languages.
- **A text embedding library.** Flair has simple interfaces that allow you to use and combine different word and document embeddings, including our proposed Flair embeddings and various transformers.
- **A PyTorch NLP framework.** Our framework builds directly on PyTorch, making it easy to train your own models and experiment with new approaches using Flair embeddings and classes.

Now at version 0.12.2!

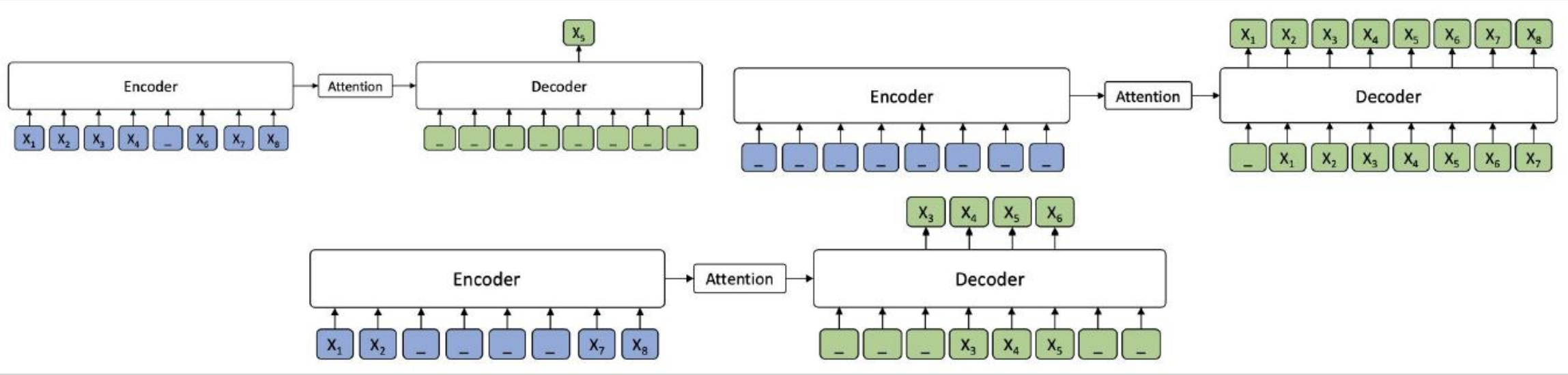
https://github.com/flairNLP/flair

Pre-trained transformer types (by training method)

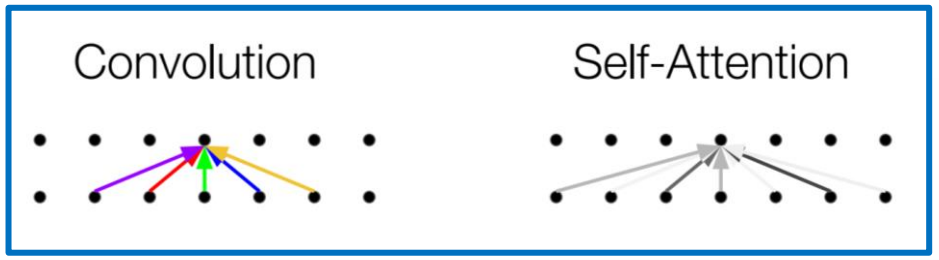
Encoder only (autoencoder)
- BERT, ALBERT, RoBERTa
- Seq classification, token classification
- Masked words and predict

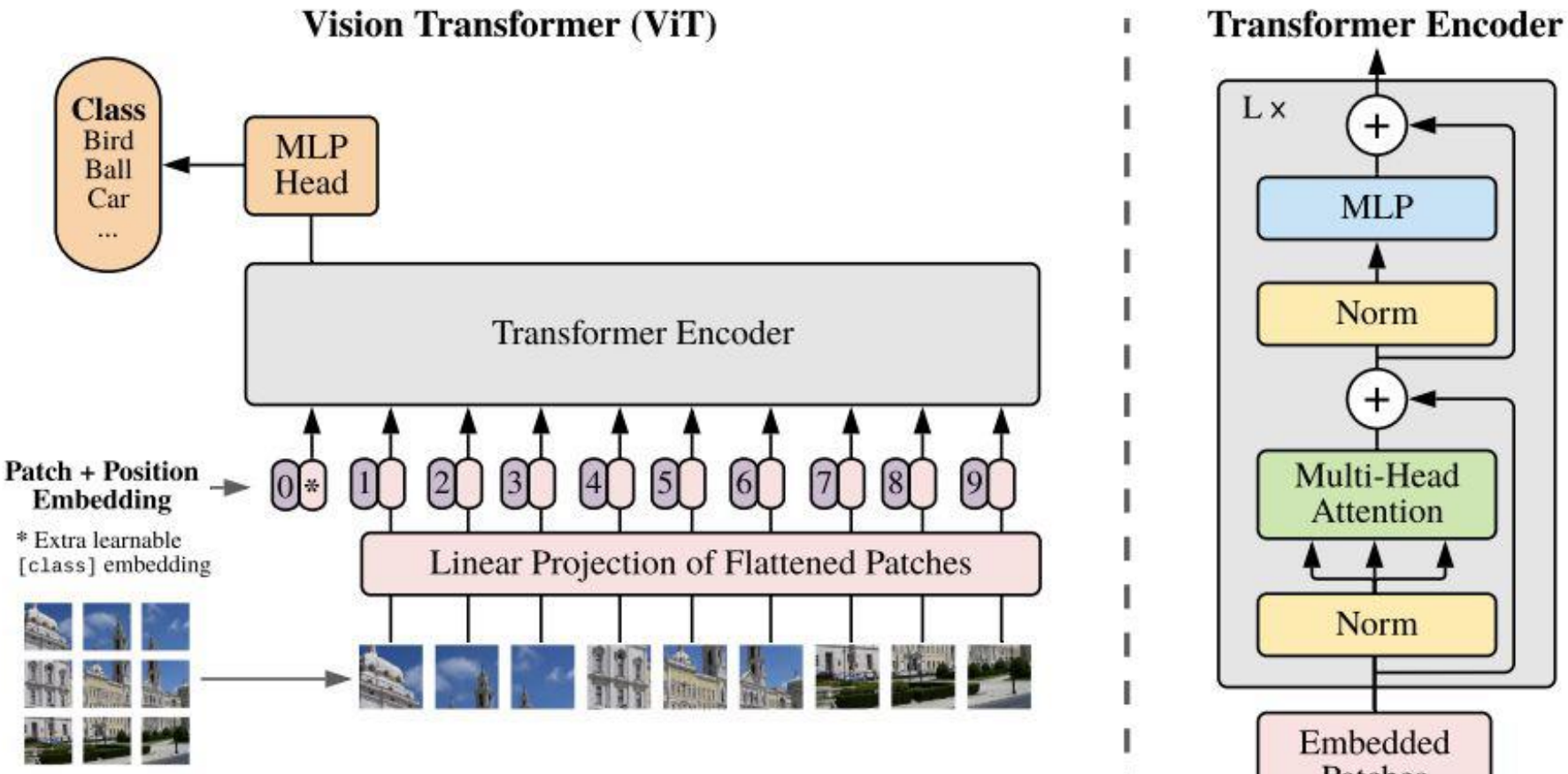
Encoder-decoder (seq2seq)
- MASS, BART, T5
- Machine translation, text summary
- Mask phrases

Decoder only (autoregressive)
- GPT
- Text generation
- Predict next word



Vision transformer





https://arxiv.org/pdf/2010.11929v1.pdf

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Demo1: “true-voice” Classification using ULMfit

https://drive.google.com/file/d/1dVVKjKuTwf2qDGAfnDsN4qtPn36gdTNn/view?usp=share_link

Demo2: BERT finetuning

https://drive.google.com/file/d/1jPD1DaGwxewLB8SThUcKLe4QqU44WZDh/view?usp=share_link

Grading (100%)

- Assignments (40%): 4 times
- Project (60%): Final presentation 15%, Report 30%, Progress 5% and Q&A participation 10% (at least 5 Questions)

Project assigned

- Group of 3-5 people (Due date 28/04/2023)
  - https://docs.google.com/spreadsheets/d/1kQUZqgbU6oG-Y1RZAo6B_PaTbfWH-mM0jnzKn7s9aC4/edit?usp=share_link
- Anything text/NLP related
  - A component of application should be required.
  - Should not be purely basic NLP task.
- Dataset guideline
  - LST20 (https://aiat.or.th/lst20-corpus/) → NER, Text classification
  - PyThaiNLP (https://github.com/PyThaiNLP/spelling-check/tree/master/dataset)→ Thai Spelling Check Dataset
  - IMDB ratings (https://ai.stanford.edu/~amaas/data/sentiment/)→ Large Movie Review Dataset
  - https://github.com/wannaphong/MyList_ThaiNLP_Group
  - https://nlpforthai.com/tasks/
  - https://huggingface.co/datasets
  - **Self-made!!!**

14	21/04/2023	Aj Paisit	Recent Research in NLP (Special Topic: BERTs & GPT) and Project or Paper Announcement	Lecture Project assigned
15	28/04/2023	Aj Paisit	Break	HW4 submitted
16	05/05/2023	Aj Paisit	Progress	Presentation (5-10 min)
17	12/05/2023	Aj Paisit	Break & Project Consulting	Consults
18	19/05/2023	Aj Paisit	Break & Project Consulting	Consults
19	26/05/2023	Aj Paisit	Final (Project Presentation due)	Presentation (15-20 min) Report Submitted

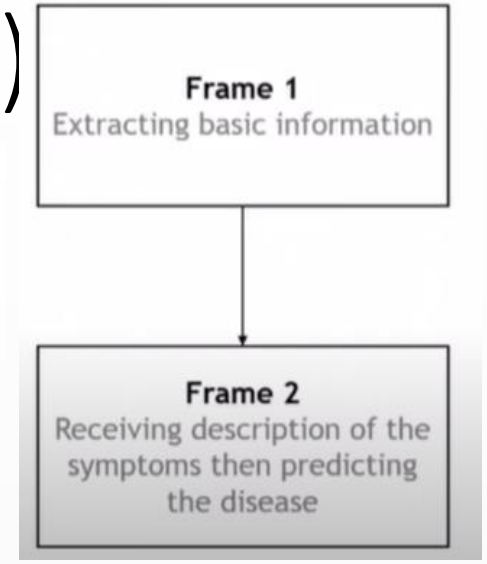
Before 13.30 pm

You must provide

- **Progress Presentation (5%)** → Define objectives, scopes, a few experiments.
- **Final Presentation (15%)**→ Project Description, Importance of the Project, Analysis Process, Analysis Results, Question and Answer, Demo
- **Report (30%)** **[Follows from parts of a scientific article]**
  - **Introduction and Literature review [10%]** (at least TWO - THREE articles) → Your goal for this task is to guess a number of the performance of the model before coding and try to find models/concepts related your works.
  - **Building models [10%]**
    - → Create models and evaluate the performance, compare the performance at least TWO difference types of models/concepts.
    - → Describe your models, inputs/features, loss function, segmentation/tokenization used, network structure (Figures), your experimental setup (Machine learning pipeline), report the performance train/test
  - **Analysis[10%]** → Compare and contract the performance (may include runtime) between models
    - What kind of errors are specific to certain kinds of models?
    - What kind of errors are common?
    - Are they expected?
    - May suggest possible solution
- **Code (.ipynb)** that shows the entire pipeline from data loading to evaluation, including results (do not clear outputs)

Project guideline(1)

- Medical Helpdesk Chatbot
  - Use extracted basic information and symptoms (as sentence) to predict the disease.



Frame1

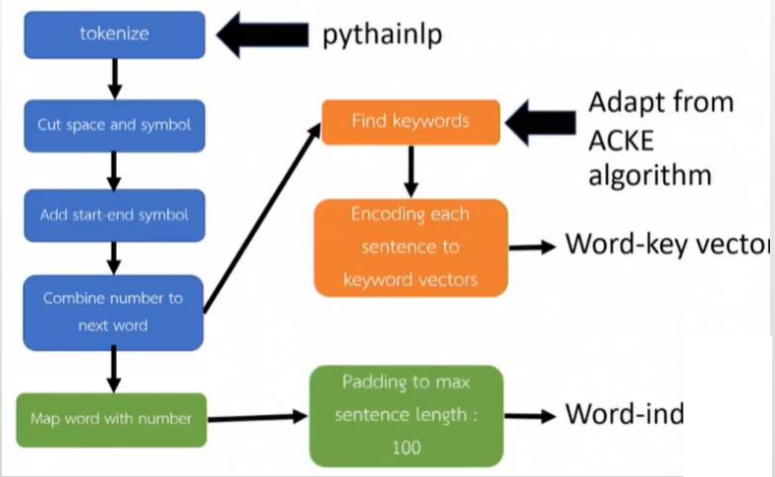
Previous Model

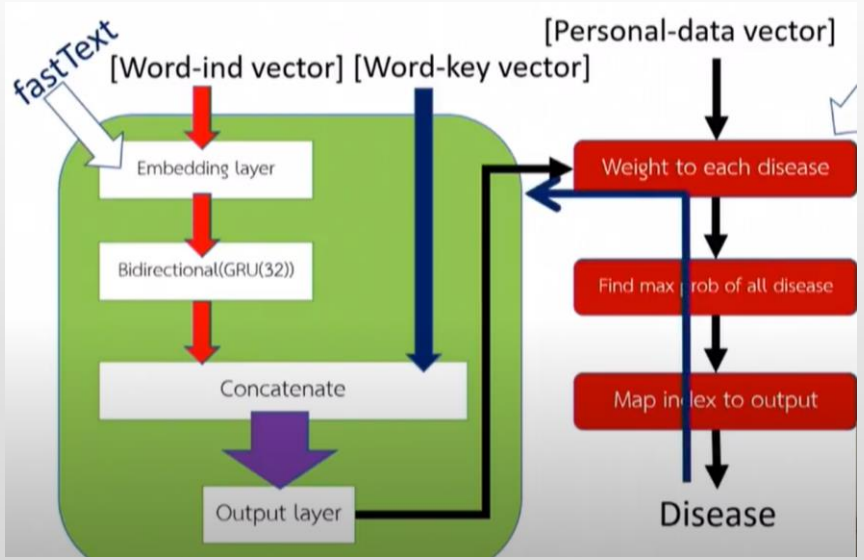
- *deepcut* tokenizer
- Replace numbers with special tokens
- Padding and appending with special tokens
- *fastText* word embedding
- Bi-GRU(12) -> Dense(64) -> Softmax Output
- Result: ~99.6% accuracy (baseline: ~95.6%) on ~120 training data

Evaluation

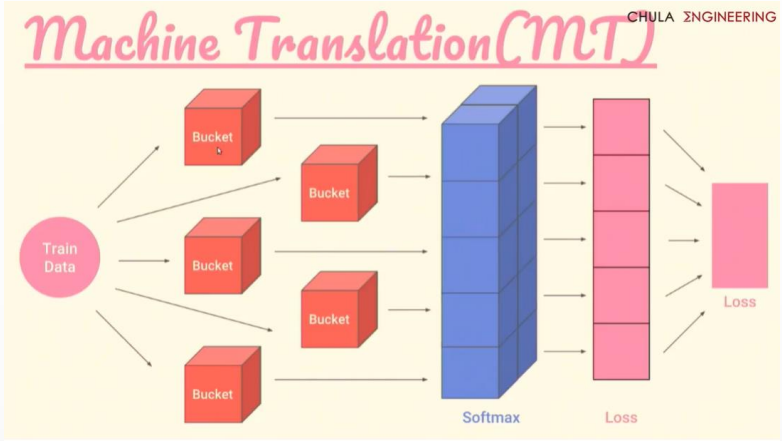
Baseline:	~95%
Rule-based:	~97.5%
Previous model:	99.64%
Pure *ICU* w/o *fastText* lib: (lol)	73.04%
Reading form w/o *fastText* lib:	86.64%*
ICU x *deepcut*:	99.68%
fastText lib:	99.71%
fastText lib x *deepcut*:	99.51%

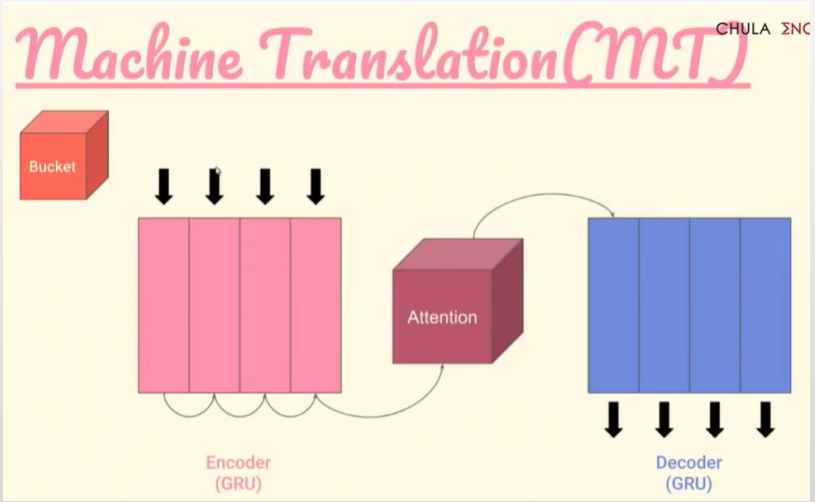
Frame2

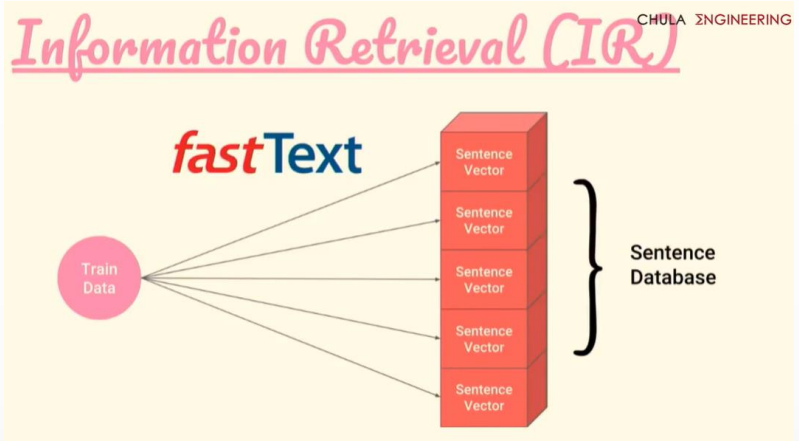


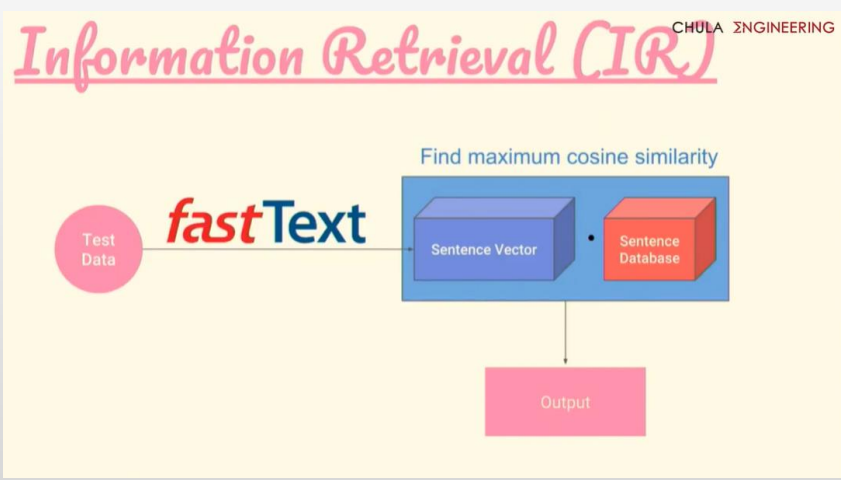


Project guideline(2)









Project guideline(3)

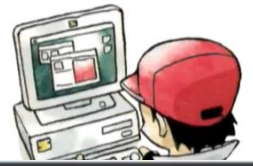
CHULA ΣNGINEERING

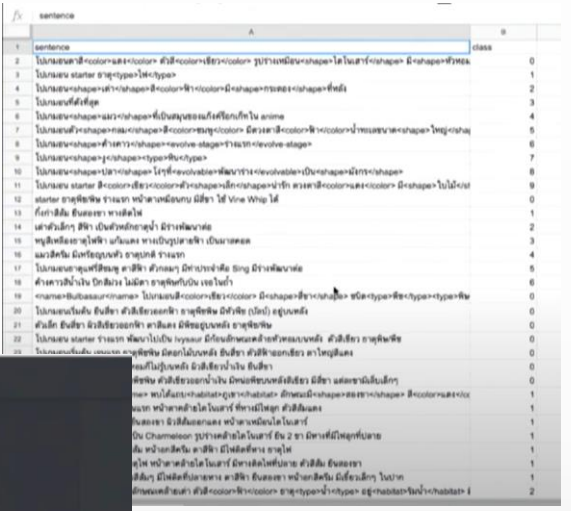
Rotom Pokédex

- User describes a Pokémon by its features e.g. appearance, shape, color, habitat, abilities.
- Tries to identify the Pokémon and responds with a link to Bulbapedia.
  Example : https://bulbapedia.bulbagarden.net/wiki/Rotom
- Runs on DISCORD

Data

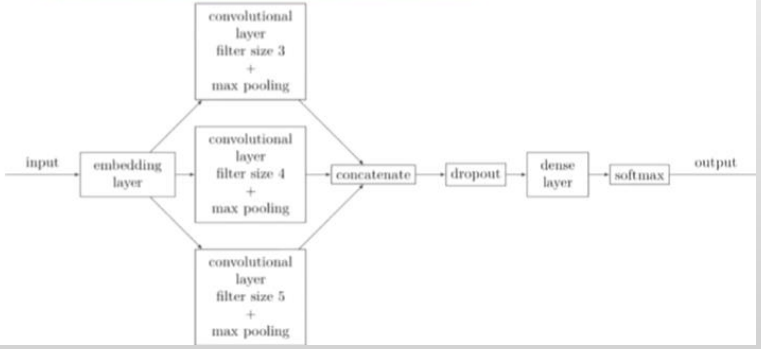
- Pokemon Data: pokedex (github.com/veekun/pokedex)
- Descriptions, images: Bulbapedia (bulbapedia.bulbagarden.net)
- Corpus: self-made

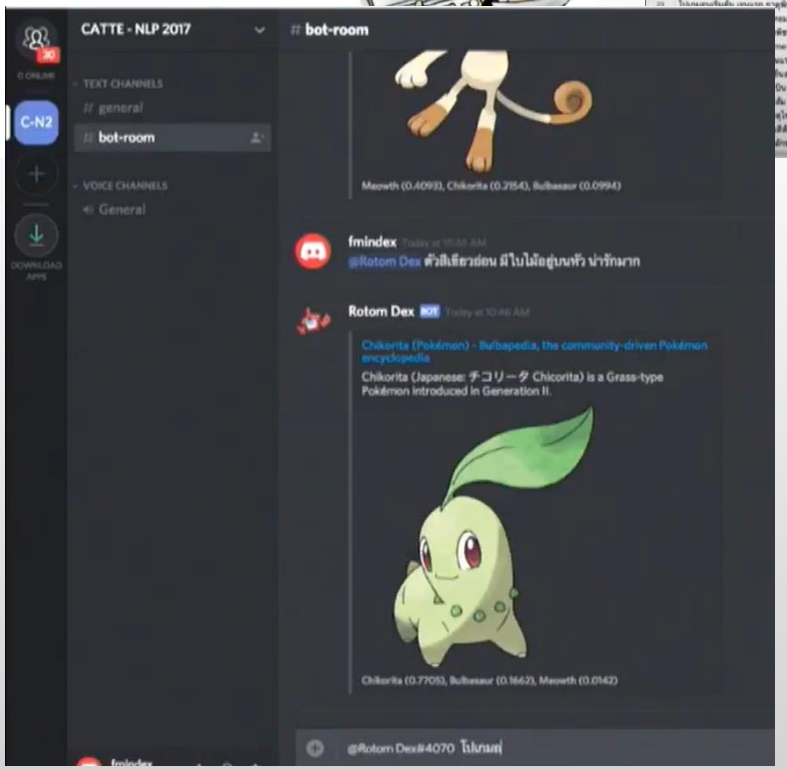




Model

- End-to-end
- Tokenization: Deepcut (github.com/rkcosmos/deepcut)
- Embeddings: thai2vec (github.com/cstorm125/thai2vec)





Project guideline(4)