Transfer Learning

Peerapon Vateekul, Ph.D
Credit: Can Udomcharoenchaikit

CHULA ENGINEERING
Foundation toward Innovation
COMPUTER

Outline

1. Introduction of transfer learning

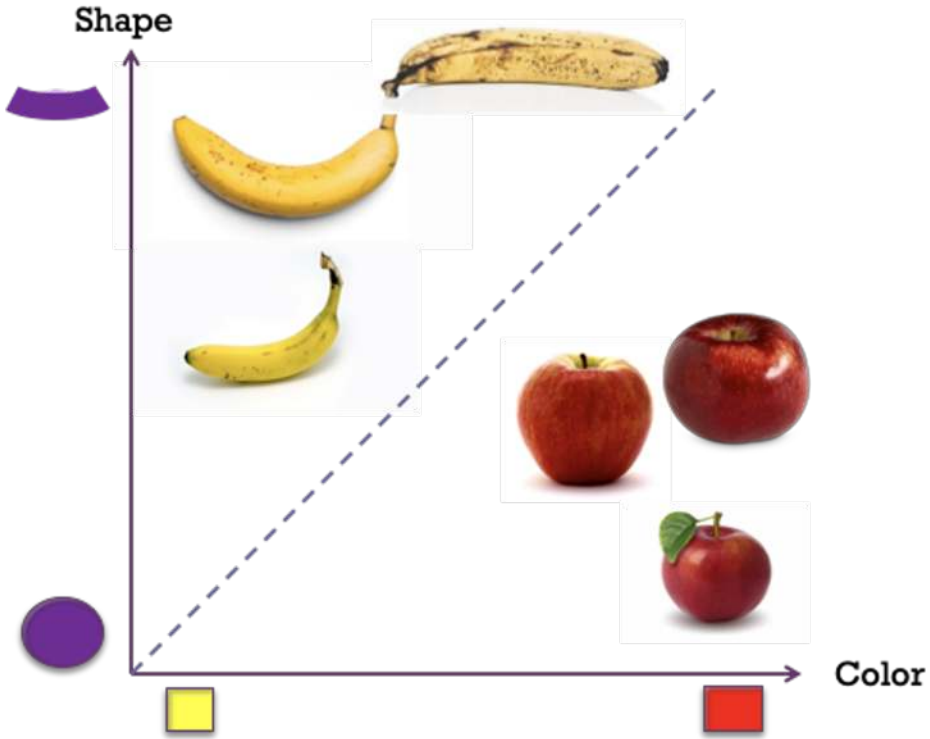
1. Transfer learning for NLP

1. Conclusion

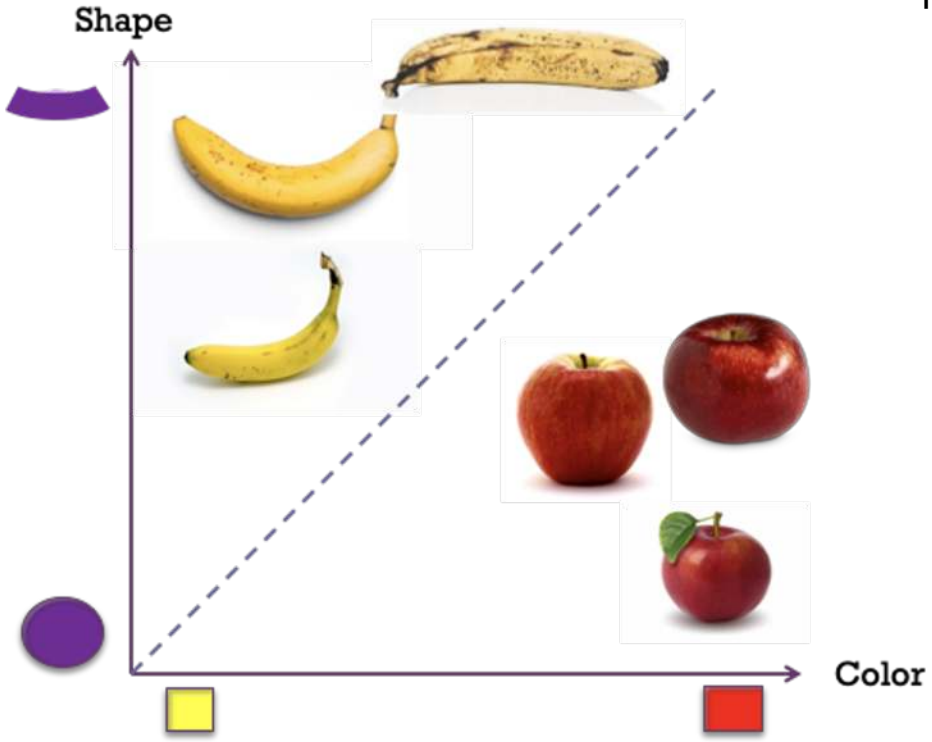
1

Introduction of Transfer learning

We need enough training data to infer the data pattern and to create model



We need enough training data to infer the data pattern and to create model (cont.)



A simple problem requires less training data.

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

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

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: idea

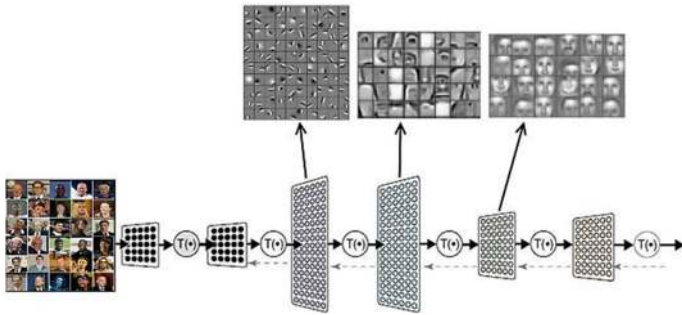
Instead of training a deep network from scratch for your task, you can

- **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)

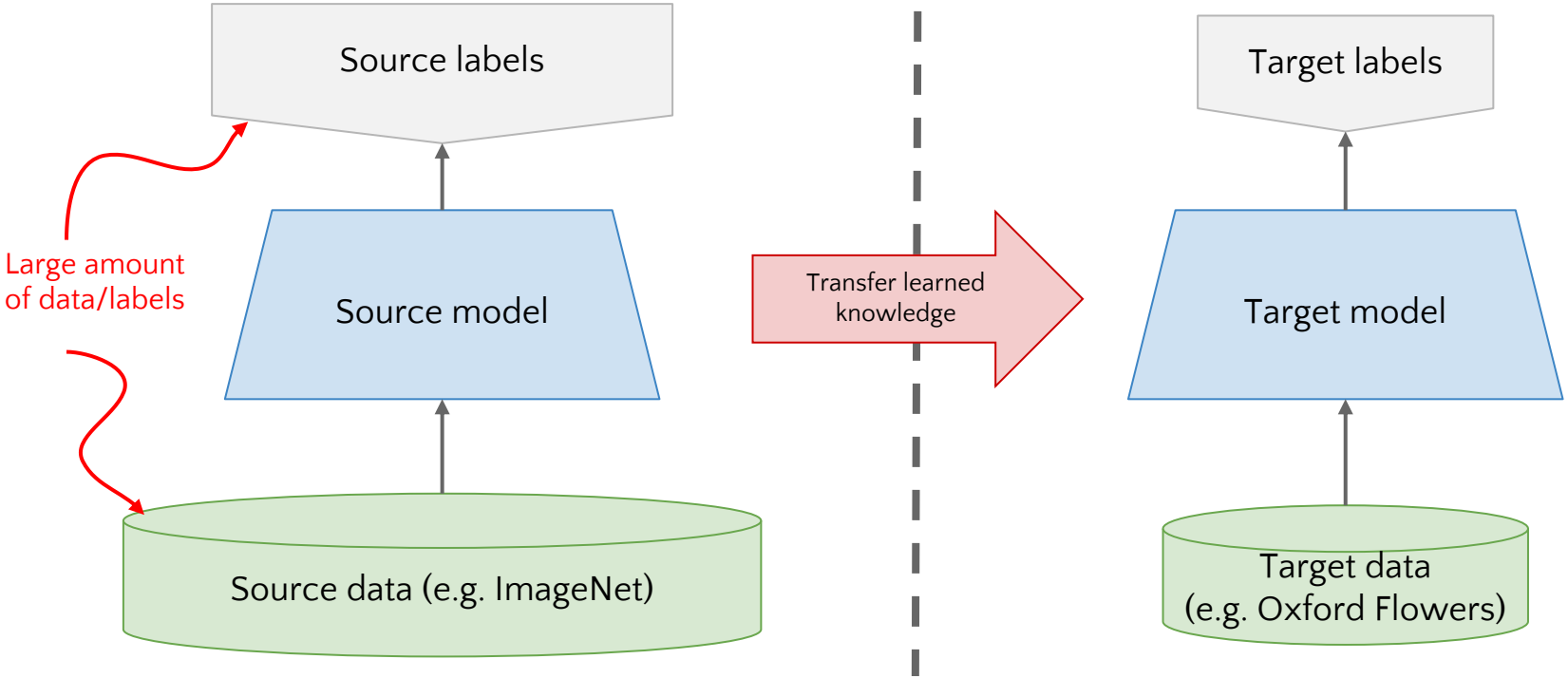
This lecture will talk about how to do this.

Variations:

- Different domain (data), same task
- Different domain (data), different task



Transfer learning: idea



http://www.image-net.org/

IMAGENET

14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications CoolStuff About

Not logged in. Login | Signup

Object recognition: 14M++ images on 20K++ categories

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. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

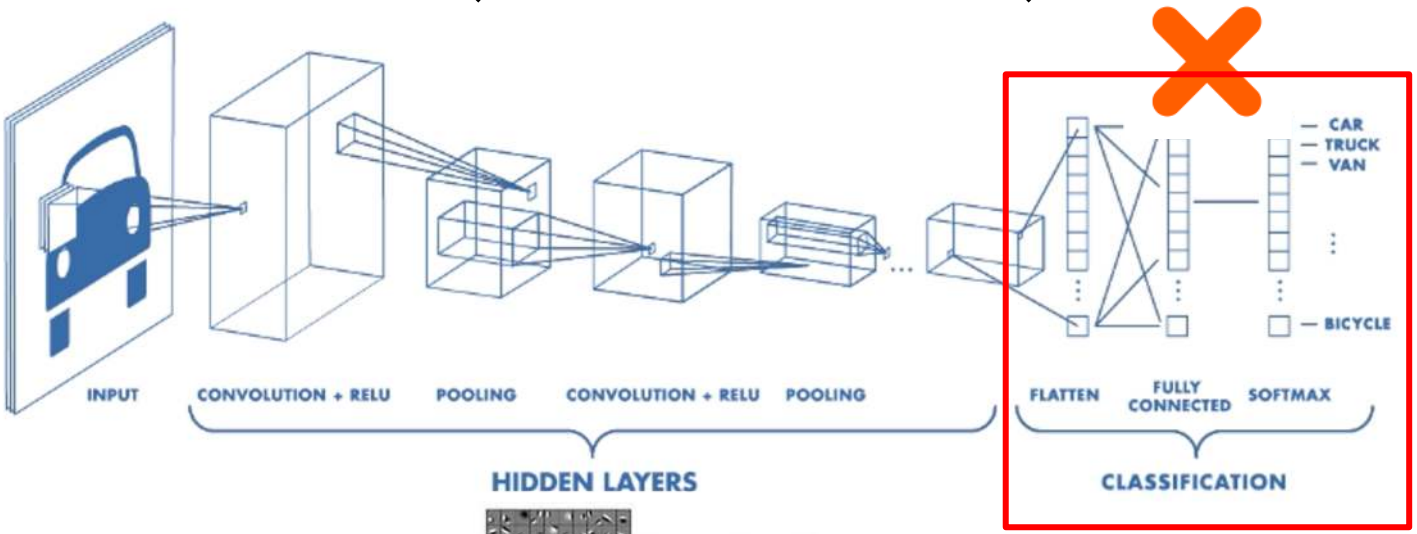
Click here to learn more about ImageNet, Click here to join the ImageNet mailing list.

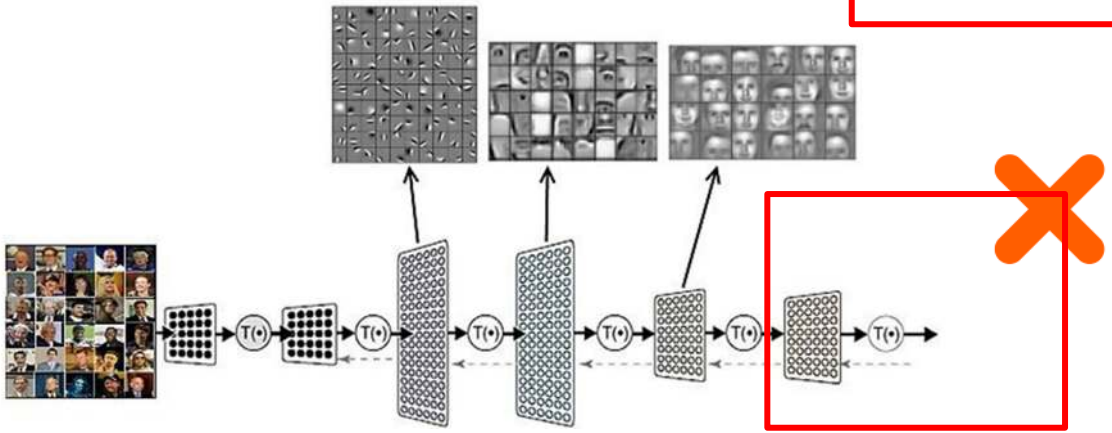
ImageNet

From Wikipedia, the free encyclopedia

The **ImageNet** project is a large visual database designed for use in visual object recognition software research. More than 14 million[1][2] images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.[3] ImageNet contains more than 20,000 categories[2] with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.[4] The database of annotations of third-party

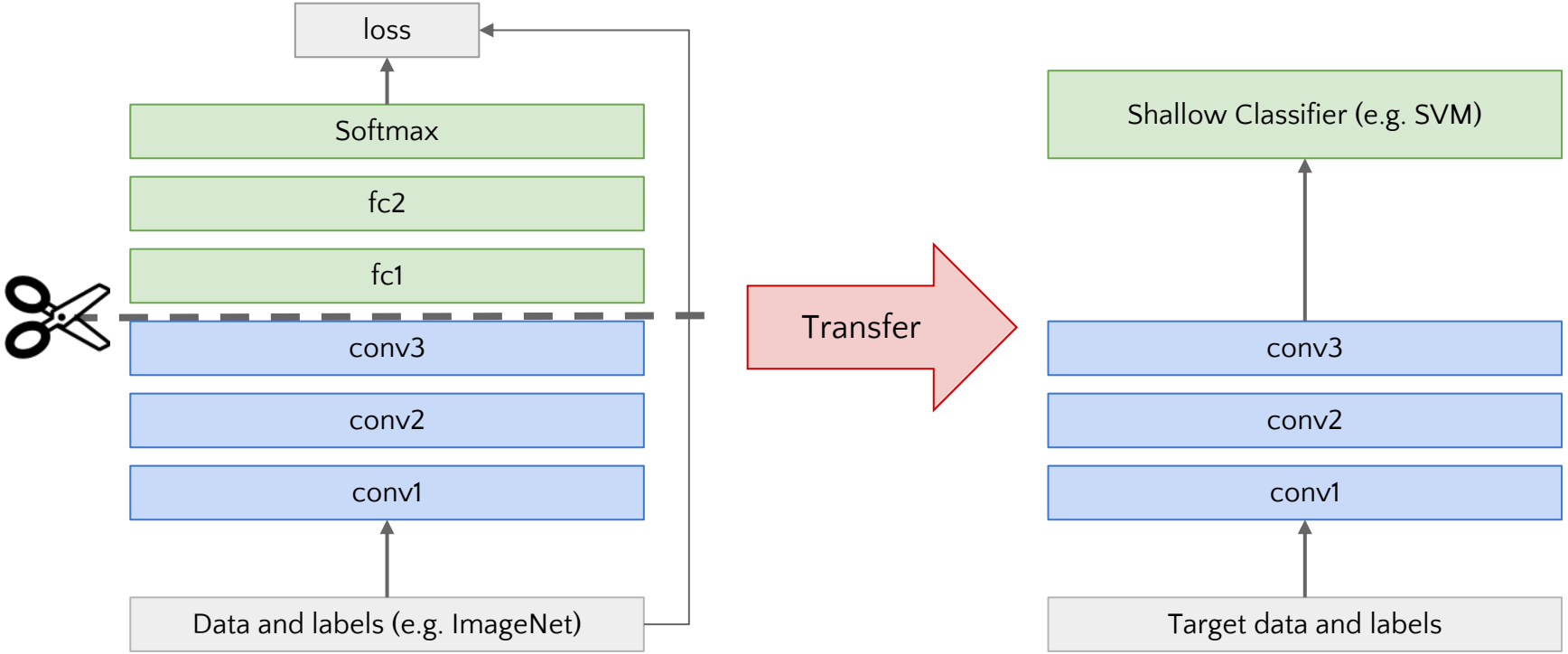
“Off-the-shelf (Feature Extractor)”





“Off-the-shelf (Feature Extractor)” (cont.)

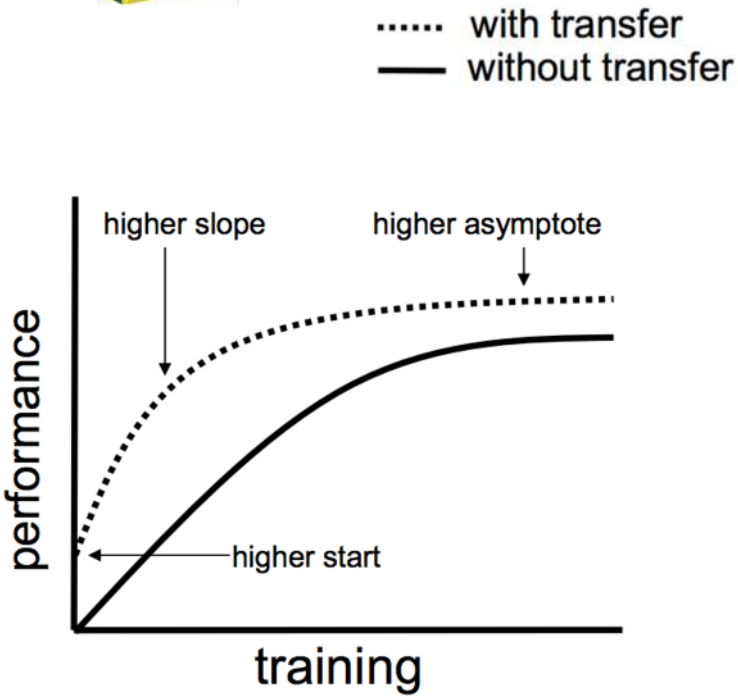
Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors.
Train a new shallow model on these features.



Transfer learning: 3 benefits

Lisa Torrey and Jude Shavlik in their chapter on transfer learning describe **three possible benefits** to look for when using transfer learning:

1. **Higher start.** The initial skill (before refining the model) on the source model is higher than it otherwise would be.
2. **Higher slope.** The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
3. **Higher asymptote.** The converged skill of the trained model is better than it otherwise would be.



Chapter 11: Transfer Learning, Handbook of Research on Machine Learning Applications, 2009.

Do you remember what you did for HW5 ?

Homework: Word Embedding

In this exercise, you will work on the skip-gram neural network architecture for Word2Vec. You will be using Keras to train your model.

The sample code for skip-gram model is given. Your job is to incorporate the tokenizer model that you created in HomeWork-1 to tokenize raw text and turn it into word vectors.

You must complete the following tasks:

1. Read/clean text files
2. Indexing (Assign a number to each word)
3. Create skip-grams (inputs for your model)
4. Create the skip-gram neural network model
5. Visualization
6. Evaluation (Using pre-trained, not using pre-trained) (classify topic from 4 categories)

Do you remember what you did for HW5 ? (cont.)

Ans.6 ผลลัพธ์ของการใช้ pre-trained weight และแบบไม่ใช้

```
normal_cls_precision = evaluate(test_input, test_target, cls_model)
pretrained_cls_precision = evaluate(test_input, test_target, cls_model_pretrained)

print("Precision without pretrained weight",normal_cls_precision)
print("Precision with pretrained weight",pretrained_cls_precision)
```

```
Precision without pretrained weight 0.47058823529411764
Precision with pretrained weight 0.8627450980392157
```

Can we do better than off the shelf features ?

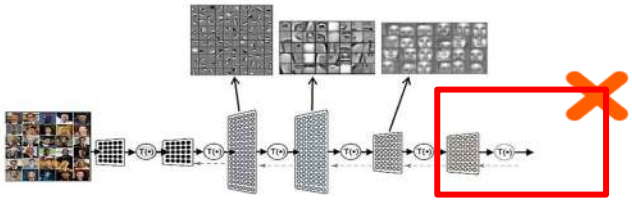
Fine-tuning: supervised task adaptation

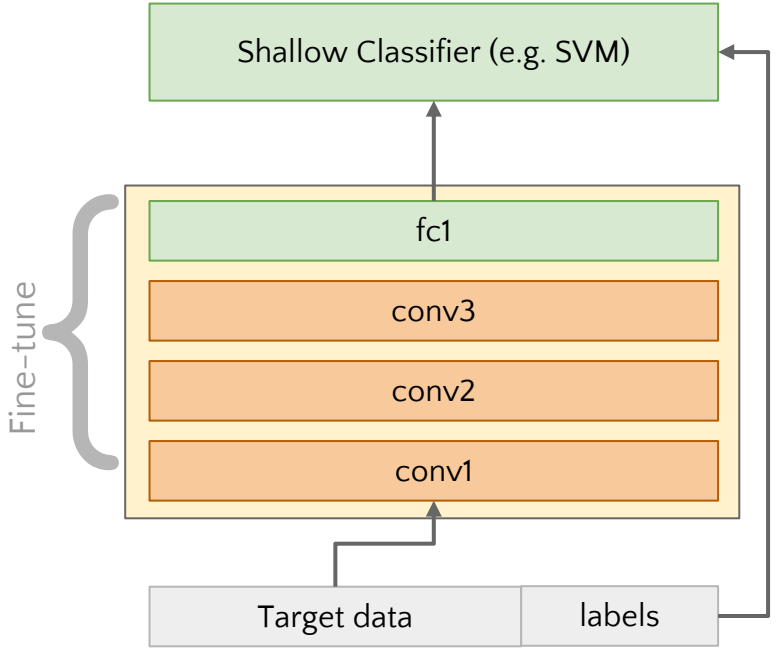
Train deep net on **nearby task** for which it is **easy to get labels** using standard backprop.

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

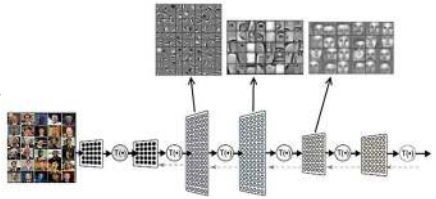
Cut off top layer(s) of network and **replace with supervised objective for target domain**.

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.

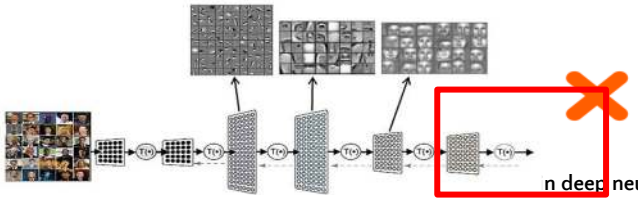
Fine-tuning improves generalization when sufficient examples are available.

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!

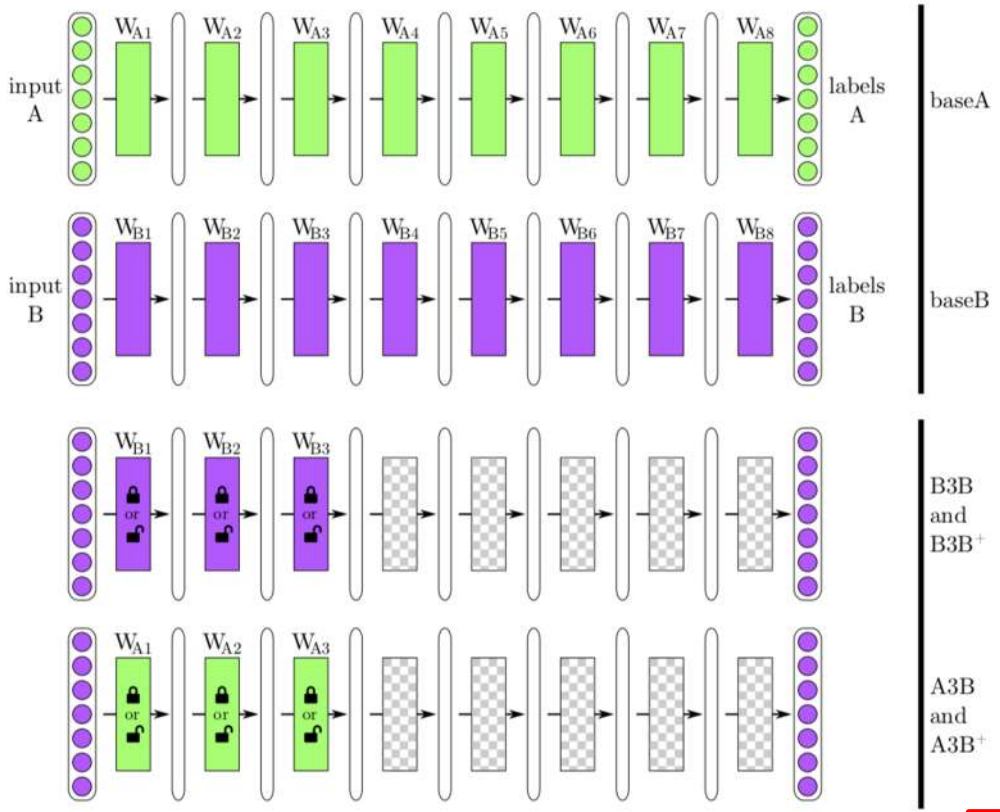
More specific

More generic



n deep neural networks, NIPS 2014 https://arxiv.org/abs/1411.1792

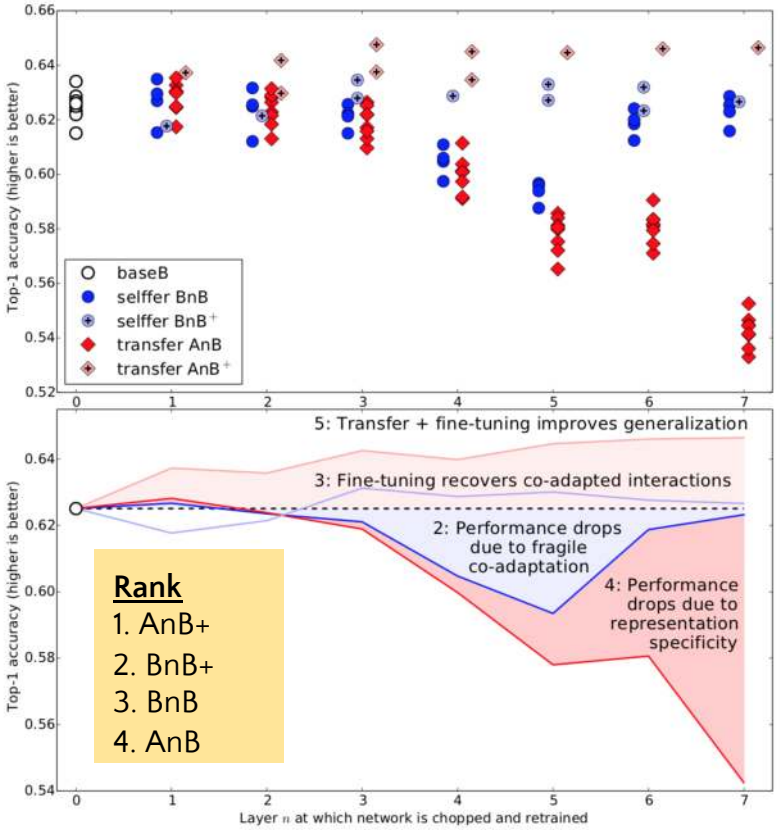
How transferable are features



- **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 baseB, 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***.

Yosinki et al., **How transferable are features in deep neural networks,** NIPS 2014 https://arxiv.org/abs/1411.1792

How transferable are features



Co-adaptation issue is a problem that layers interact with each other.

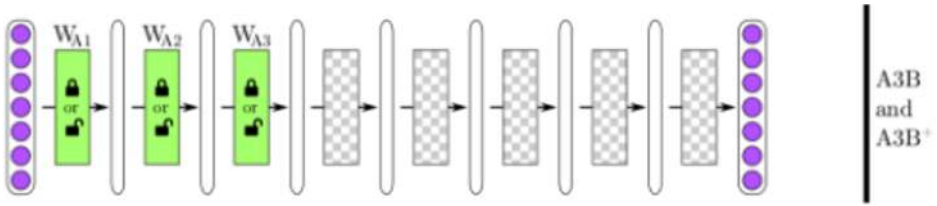
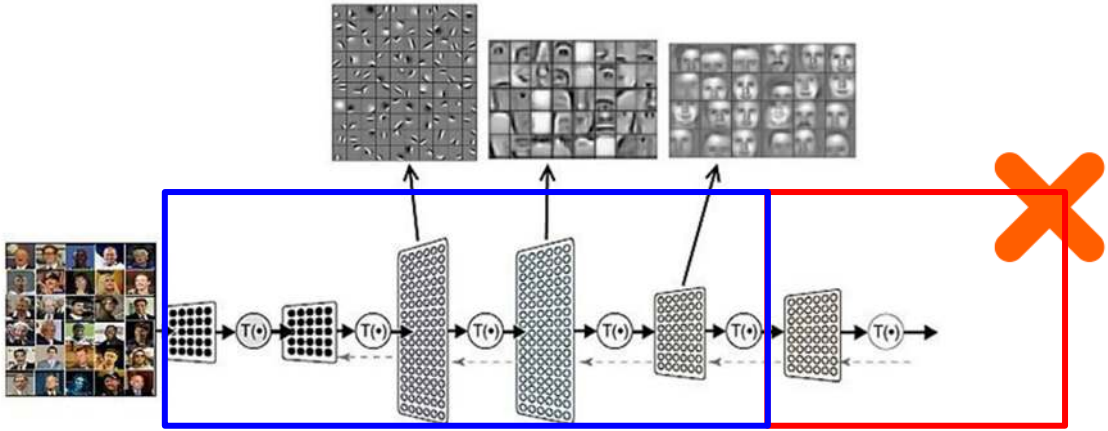


Figure 2: The results from this paper's main experiment. *Top*: Each marker in the figure represents the average accuracy over the validation set for a trained network. The white circles above $n = 0$ represent the accuracy of baseB. There are eight points, because we tested on four separate random A/B 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.

Yosinki et al., **How transferable are features in deep neural networks,** NIPS 2014 https://arxiv.org/abs/1411.1792

2

Transfer learning for NLP



Transfer learning for NLP vs CV

NLP is more difficult than image domain

Differences between transfer learning for NLP and for Computer vision fields:

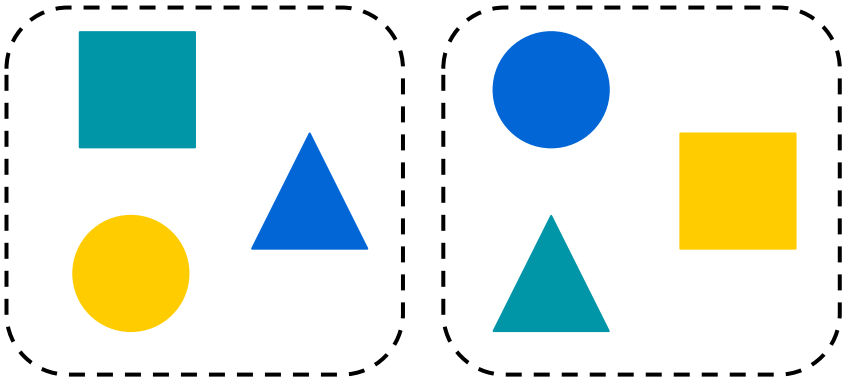
- More variety in types of target tasks.
  - E.g. entity extraction, classification, sequence labeling.

- 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



Source models

Target tasks

Color: Task type

Shape: Vocabulary

1) What source tasks?

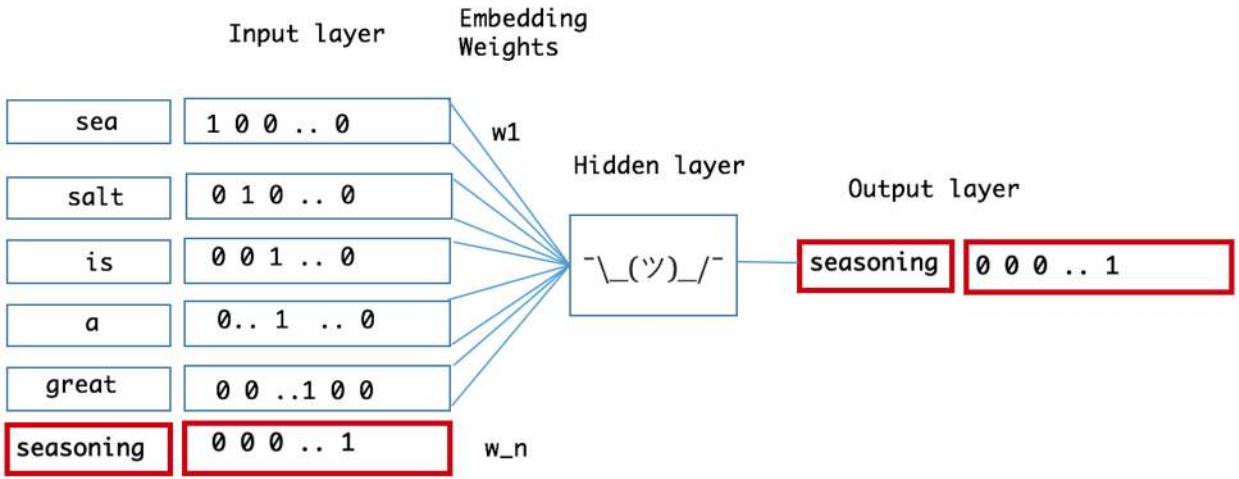
We expect the source tasks to be able to produce the good and general representations

- 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)

1) Language modeling: (cont.)
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



2) 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:

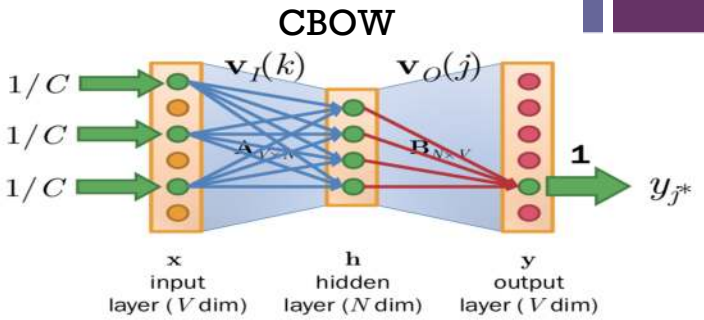
- BooksCorpus[1] (980M words)
- English Wikipedia[2] (2,300M words)
- WMT News crawl [3] (3,600M words, only from 2008-2012)

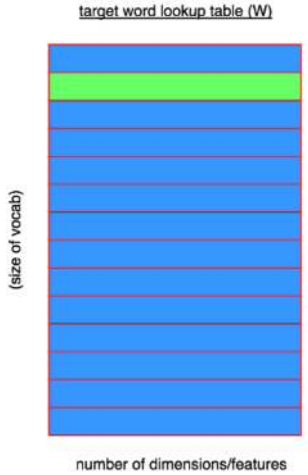
All of these corpora are unlabeled. Since it focuses on LM (predicting next word), it doesn't need any labeling effort.

[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 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.

1. ULMFiT (Universal Language Model Fine-tuning) [1]
1. ELMo (Embeddings from Language Models) [2]
1. BERT (Bidirectional Encoder Representations from Transformers) [3]
1. GPT (Generative Pre-Training)

The code of the models and their weights that already pre-trained on massive datasets are available for download.

[1] Jeremy Howard and Sebastian Ruder, **Universal Language Model Fine-tuning for Text Classification,** ACL 2018, https://arxiv.org/pdf/1801.06146.pdf
[2] Peter et al., **Deep contextualized word representations**, NAACL 2018, https://arxiv.org/abs/1802.05365.pdf
[3] Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

1) ULMFiT [Howard, et al, 2018]

Universal Language Model Fine-tuning for Text Classification

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. We propose Universal Language Model Fine-tuning (ULMFiT), an effective trans-

ay 2018

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-trained word embeddings (Mikolov et al., 2013),

1) ULMFiT [Howard, et al, 2018]

Universal Language Model Fine-tuning for Text Classification

- ULMFiT introduced methods to effectively utilize a lot of what the model learns during pre-training.

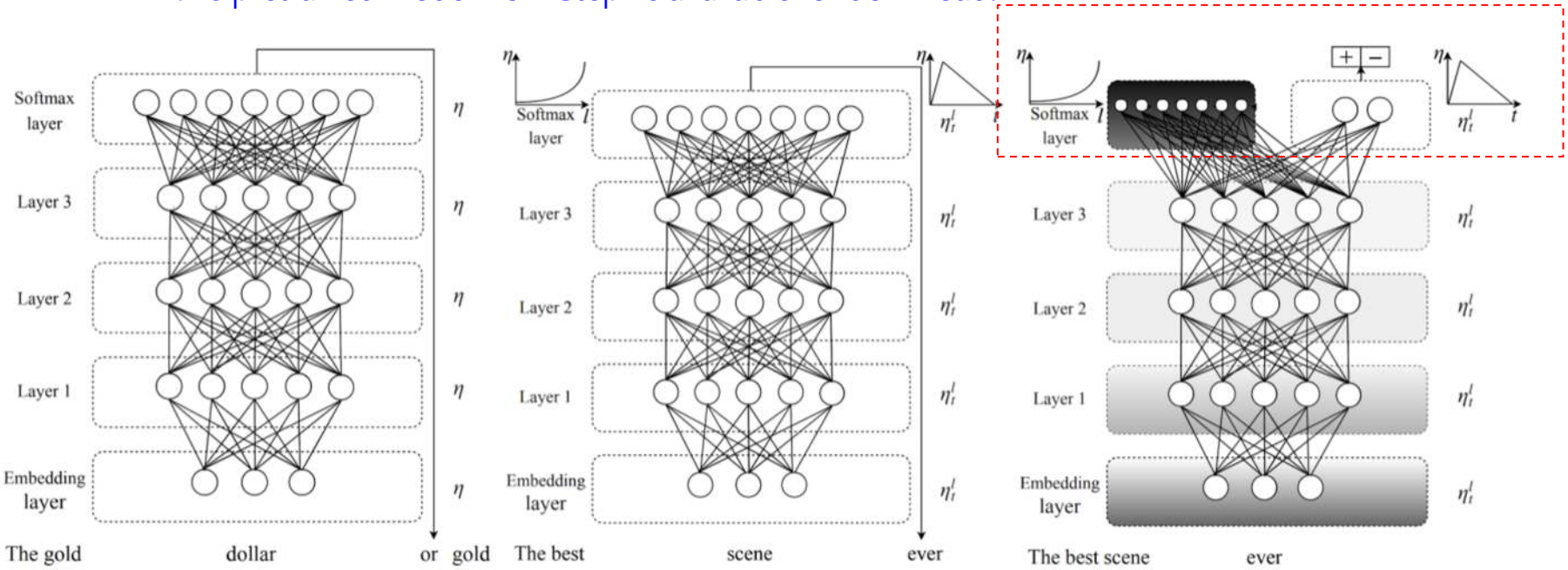
- More than just a single embedding layer, ULMFiT introduced a language model which contains many layers and a process to effectively fine-tune that language model for various tasks

- Unlike BERT, this language model is unidirectional LSTM which means that each word is only contextualized using the words to its left.

Jeremy Howard and Sebastian Ruder, **Universal Language Model Fine-tuning for Text Classification,** ACL 2018 https://arxiv.org/pdf/1801.06146.pdf

ULMFiT (cont.)

The pretrained model from Step1 is available for download.



(a) Train LM using large unlabeled corpus (wiki corpus)

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

(c) Train classifier using target corpus (target data with label)

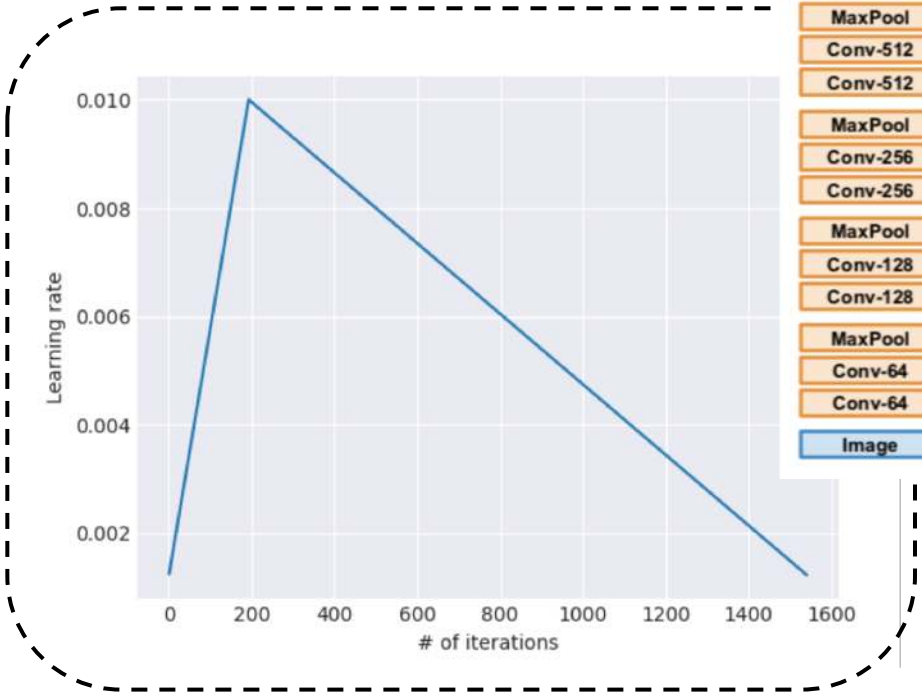
ULMFiT (cont.)

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

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

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

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

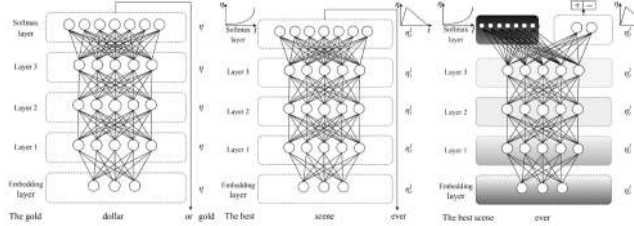


Discriminative fine-tuning

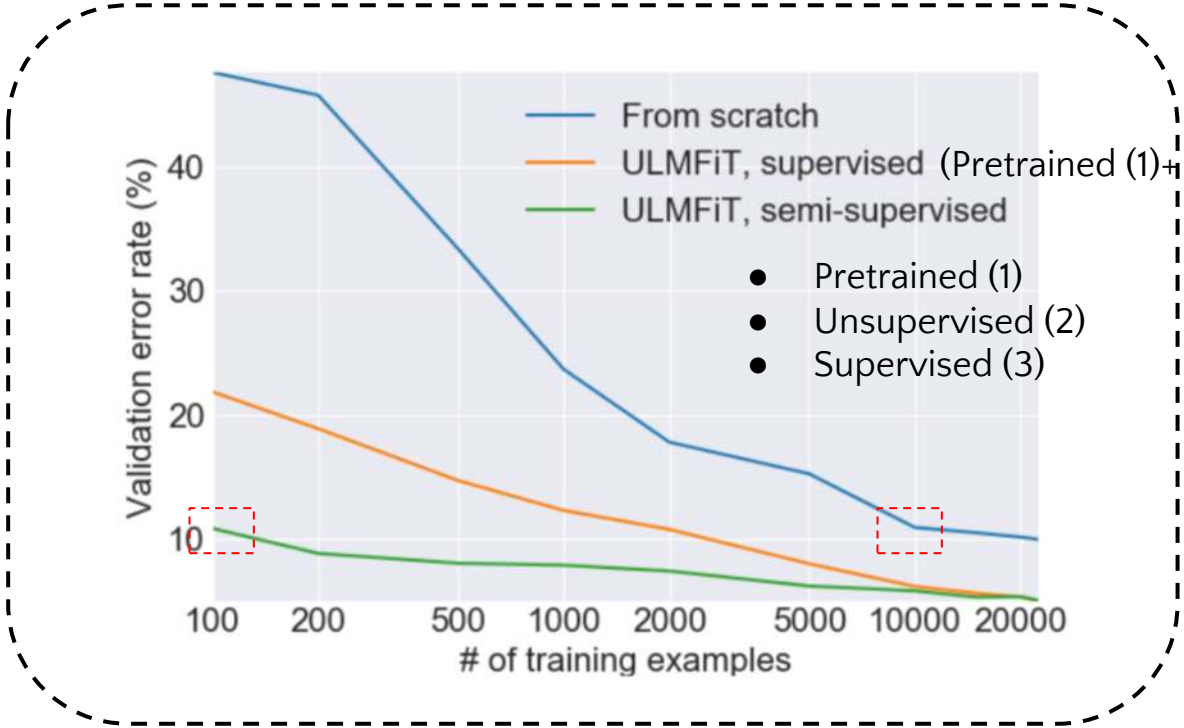
Instead of using the same learning rate for all layers of the model, discriminative fine-tuning allows us to tune each layer with different learning rates.

- Layers that are closer to inputs → large learning rate
- Layers that are closer to outputs → small learning rate

ULMFiT (cont.)



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



Pretrained ULMFiT (English)

fast.ai NLP

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

- 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/

Efficient multi-lingual language model fine-tuning

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

With ULMFiT, we can make training tex
than English a lot easier as all we need
available for 301 languages, a small nu
annotated by hand, and optionally add
even easier, we will soon launch a mod
many languages.

Pretrained ULMFiT (Thai)

https://github.com/cstorm125/thai2fit

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.

2) ELMo (Embeddings from Language Models) [Peter, et al, 2018]

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

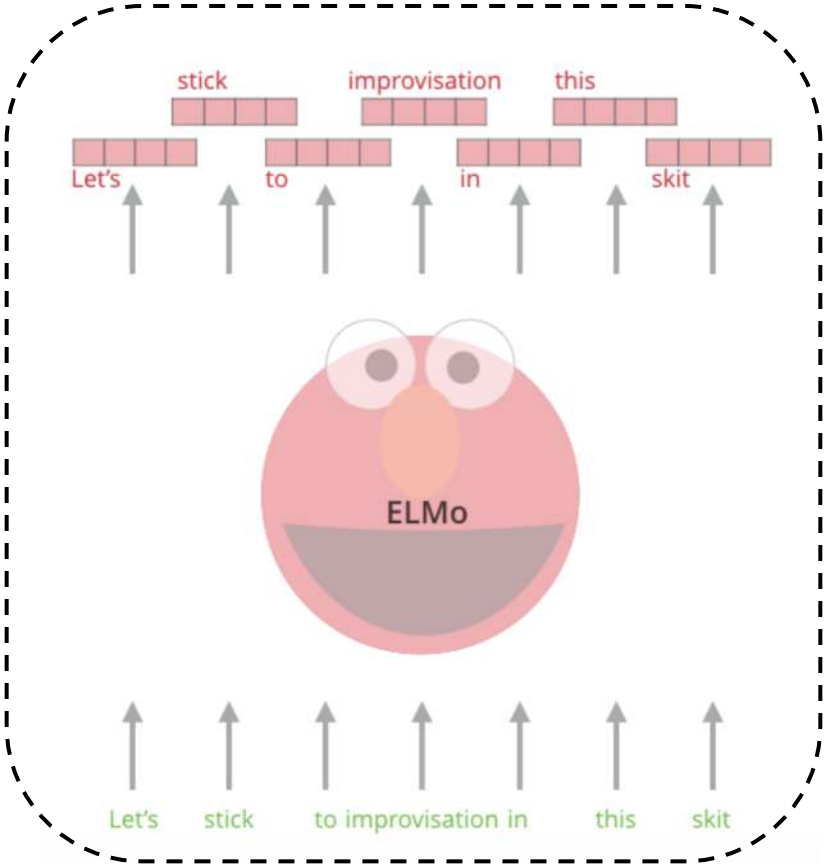
We introduce a new type of *deep contextualized* word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). Our word vectors are learned functions of the internal states of a deep bidirectional language model (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

2) ELMo (Embeddings from Language Models) [Peter, et al, 2018]

Peter et al., **Deep contextualized word representations**, NAACL 2018, https://arxiv.org/abs/1802.05365.pdf

ELMo (cont.): Contextualized

- 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.



ELMo (cont.): Architecture

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.

- 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 **2 separated** LSTM-based bidirectional language models (biLMs) 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.

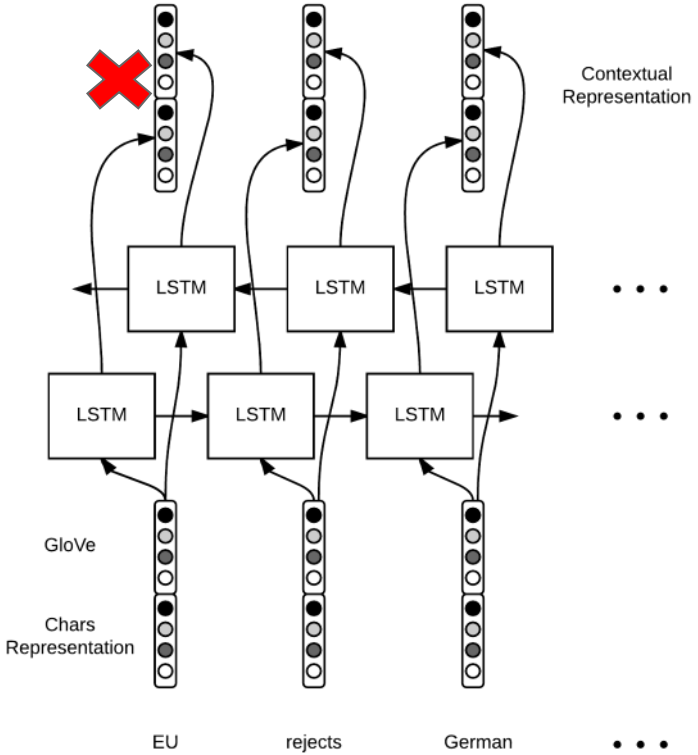


Figure reference : Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

ELMo (cont.): Architecture

- 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 **2 separated** LSTM-based bidirectional language models (biLMs) 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.

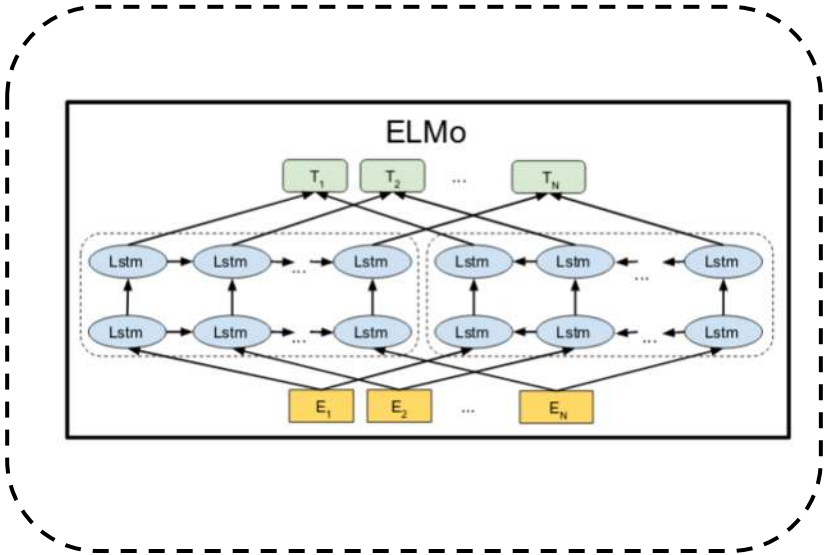
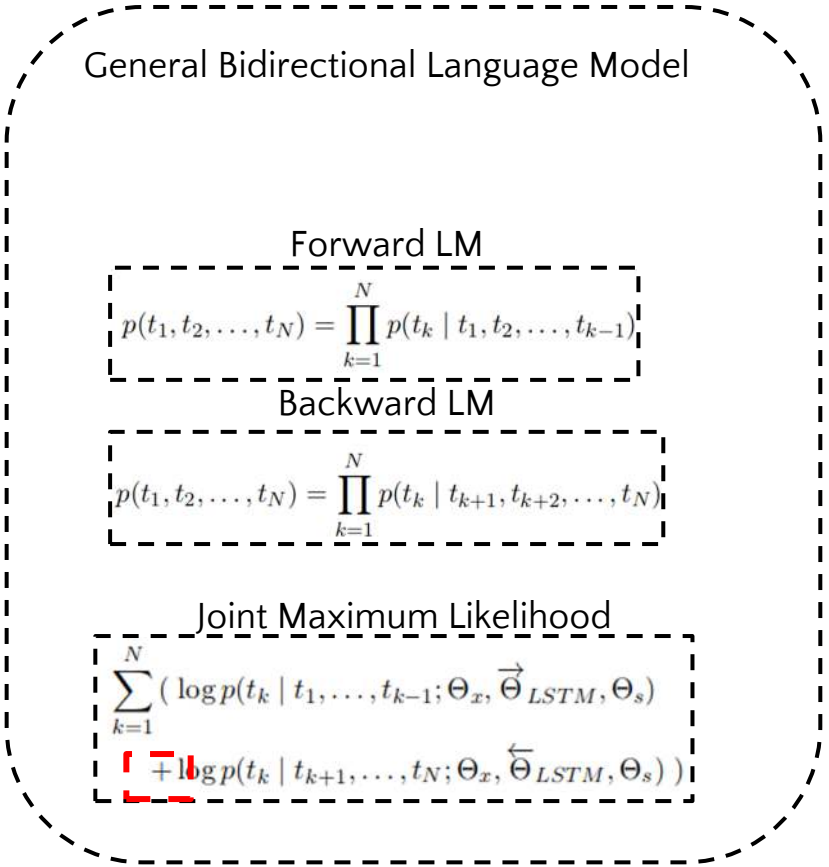


Figure reference : Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

ELMo (cont.)

- 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 **2 separated** LSTM-based bidirectional language models (biLMs) 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.



ELMo (cont.): Character-level

- 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.

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).

ELMo (cont.): Character-level

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

A downstream model collapses all layers in R into a single vector: **ELMo**$_k$

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

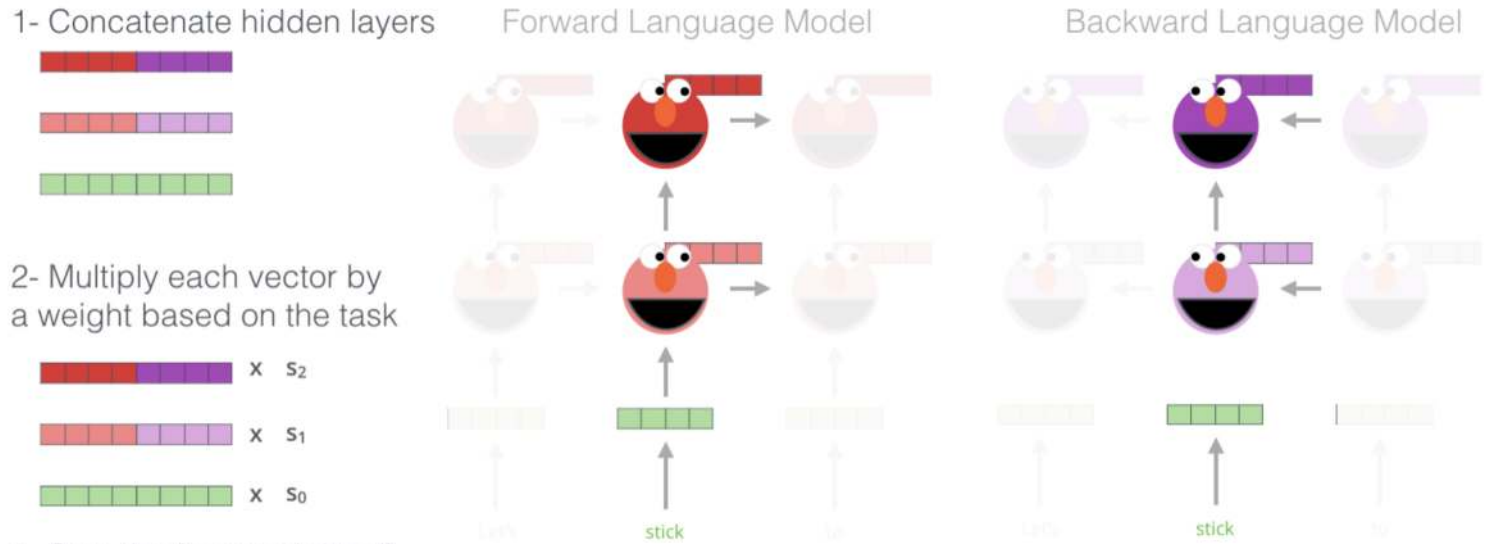
$\mathbf{s}^{task}$ are softmax-normalized weights (layer-wise weight), a summation of all layers' weight is one.

γ^{task} is a scalar parameters.

Both $\mathbf{s}^{task}$ and γ^{task} are **learnable** parameters.

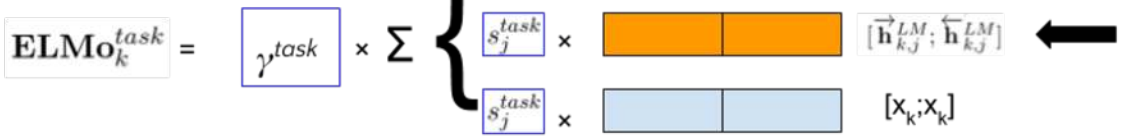
ELMo (cont.):
Downstream task.

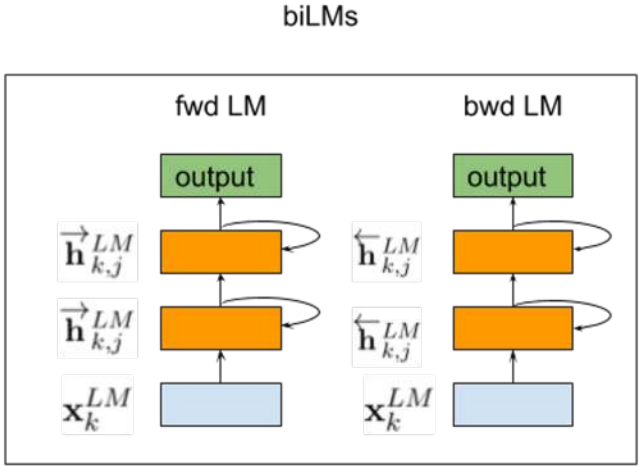
Embedding of “stick” in “Let’s stick to” (word-level task)



ELMo (cont.)

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





Pretrained ELMO (English, no Thai)

Allen Institute for AI

https://allennlp.org/elmo

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

3) BERT [Devlin, et al, 2018]:
Bidirectional **E**ncoder **R**epresentation from **T**ransformers

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

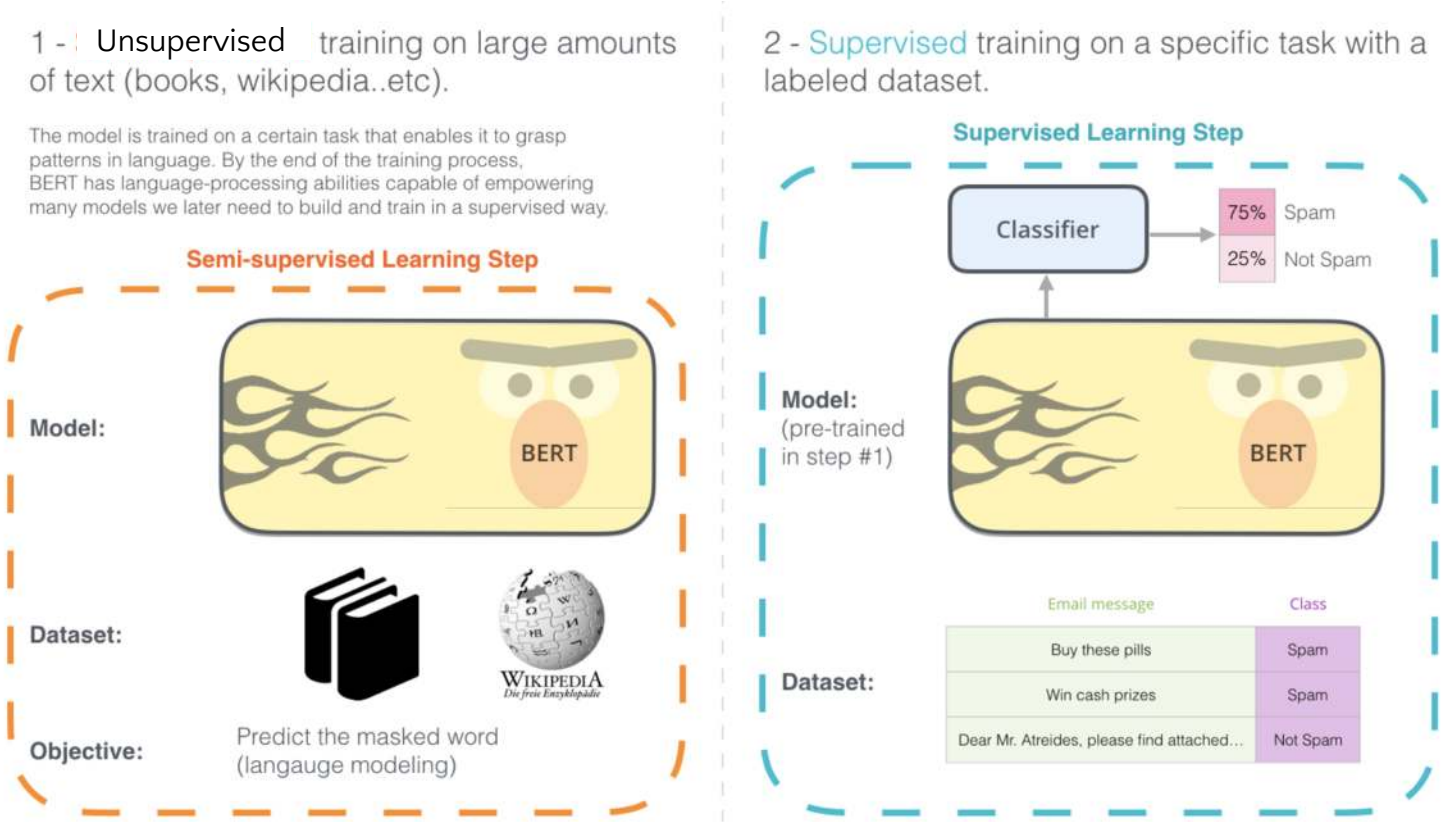
Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

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-

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-

3) BERT [Devlin, et al, 2018]:
Bidirectional Encoder Representation from Transformers



Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

BERT (cont.)

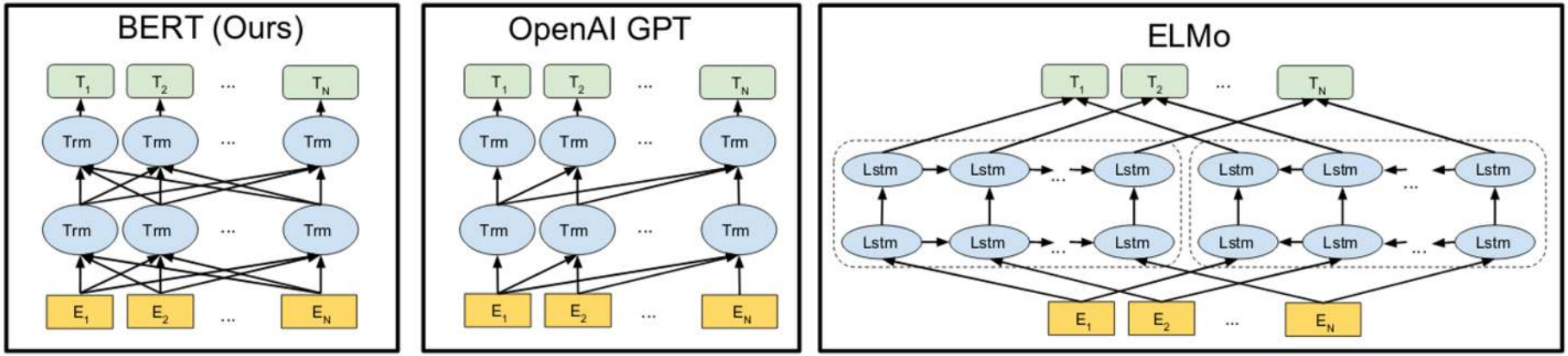


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

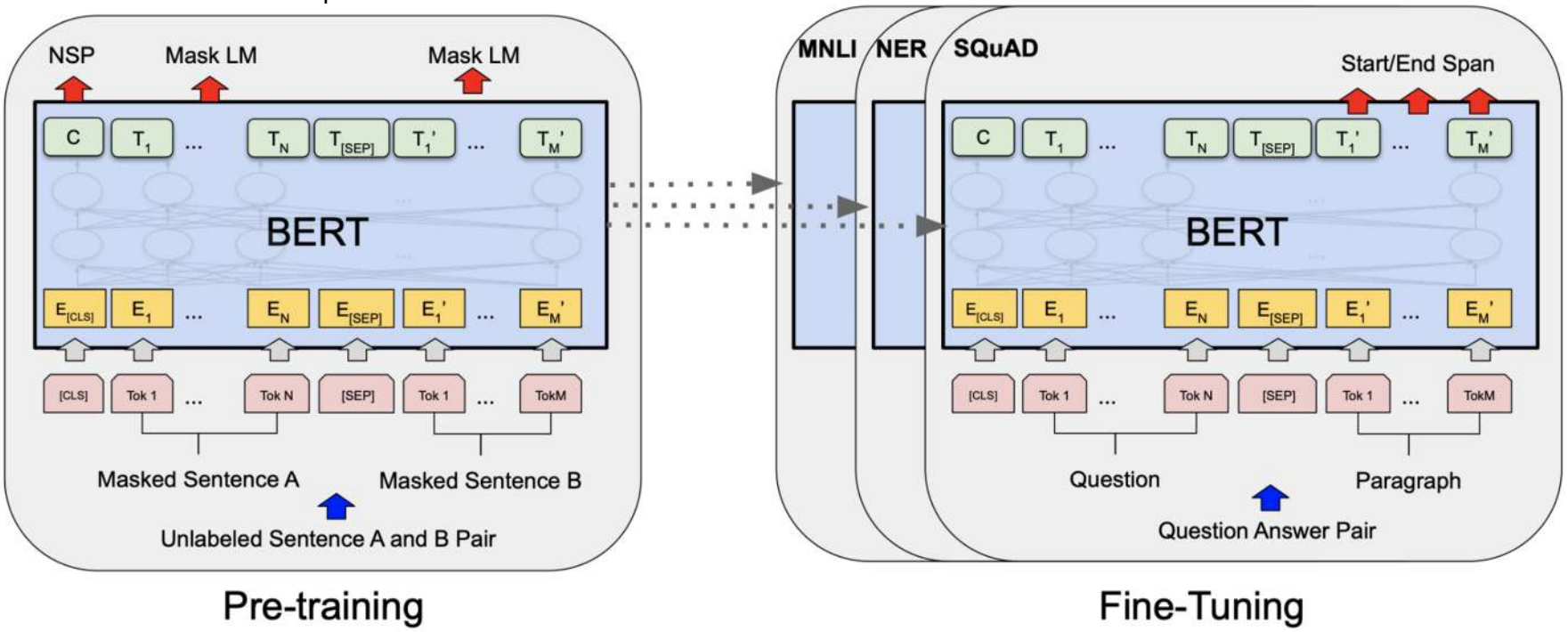
BERT (cont.): Overall idea

Phase1: Unsupervised Learning

- Mask LM
- Next sentence prediction

Phase2: Supervised learning

- **E.g.** Finetune on QA, Text classification, etc.



Phase1: Unsupervised Phase

Semi-supervised training, using 2 prediction tasks:

1. **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
```

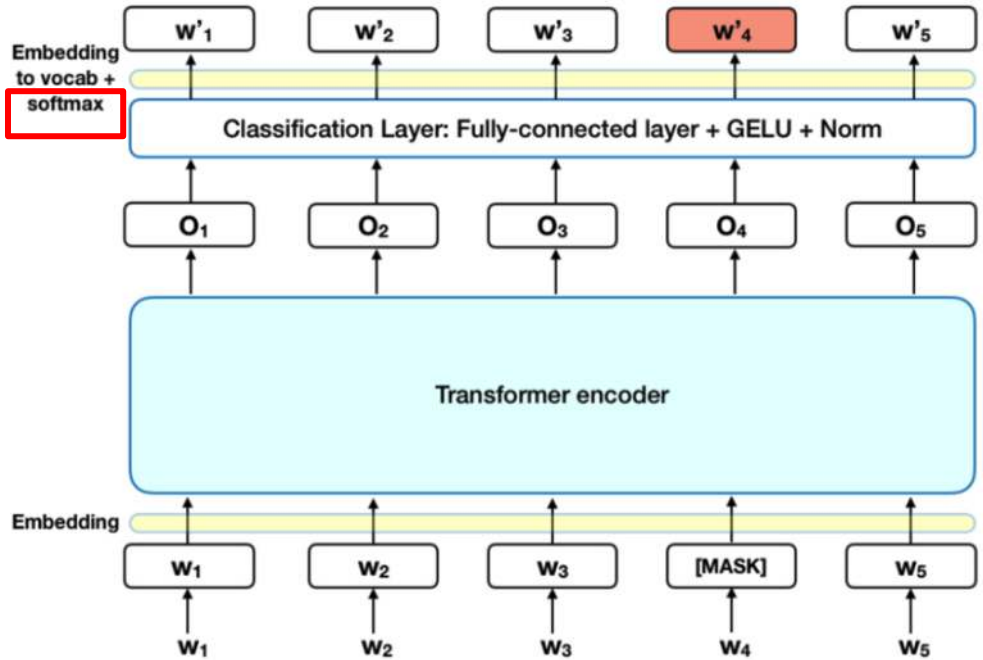
1. **Next sentence prediction (NSP)**
   - to learn relationships between sentences.

```
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
```

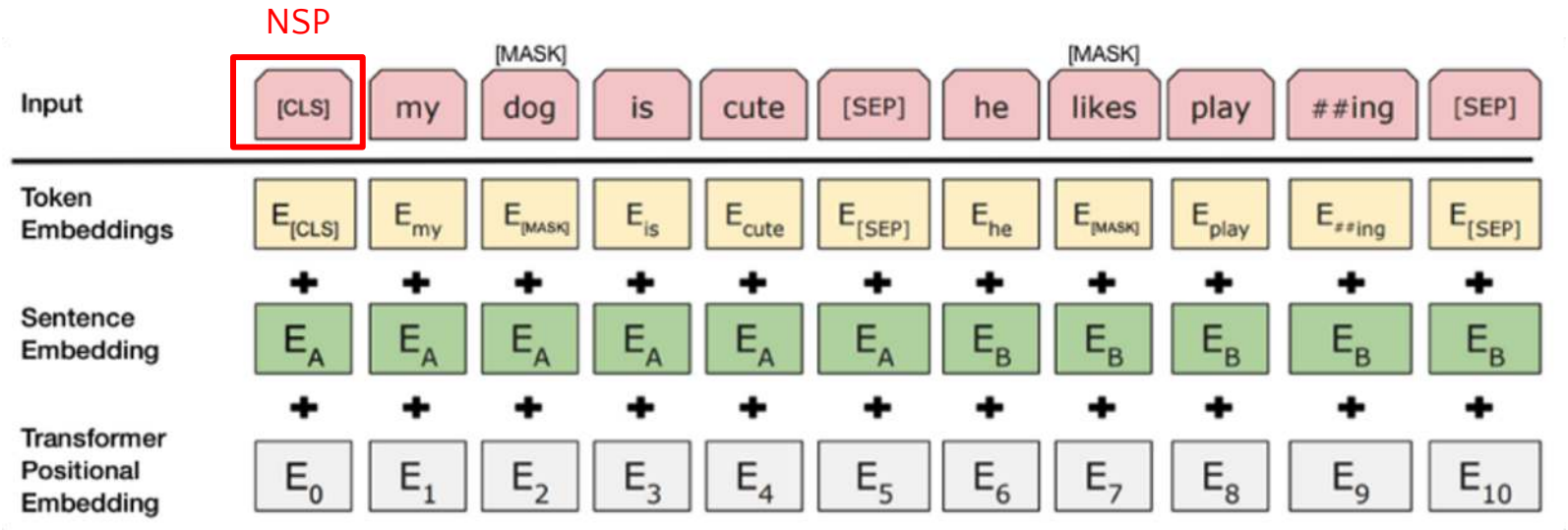
Phase1: Unsupervised Phase (1.1. Masked LM)

Mask language modeling:



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

Phase1: Unsupervised Phase (1.2. Next Sentence Prediction)



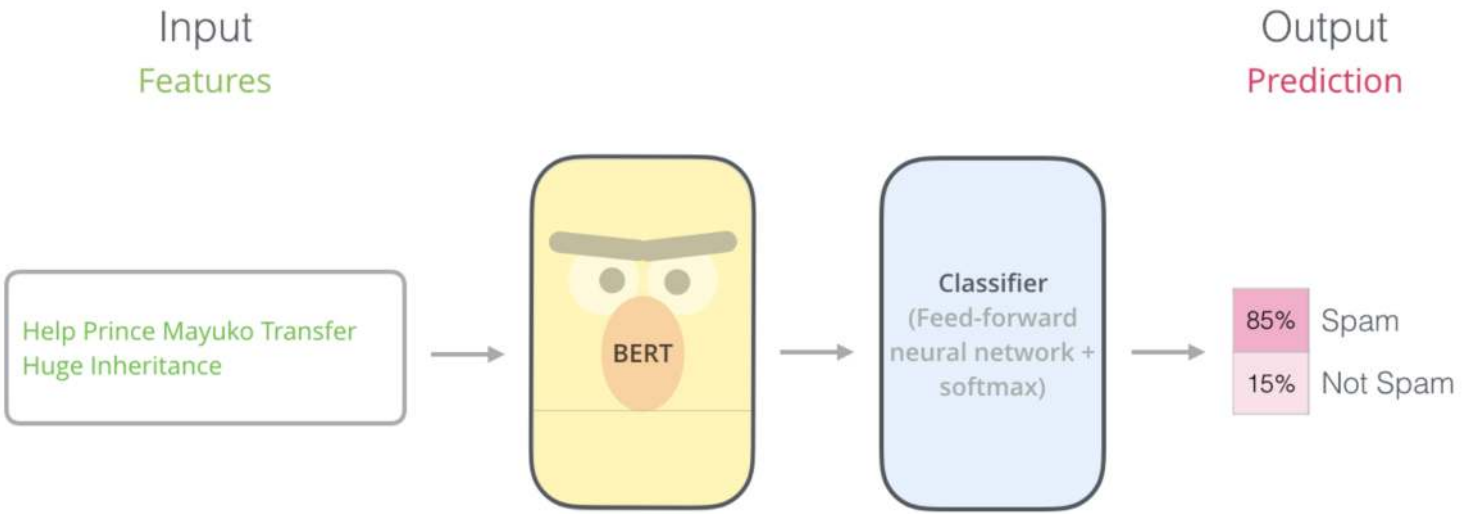
CLS = Classification, which is used for NSP (Next Sentence Prediction) during this phase
SEP = a special separator token (e.g. separating questions/answers).

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

Phase2: Supervised Phase

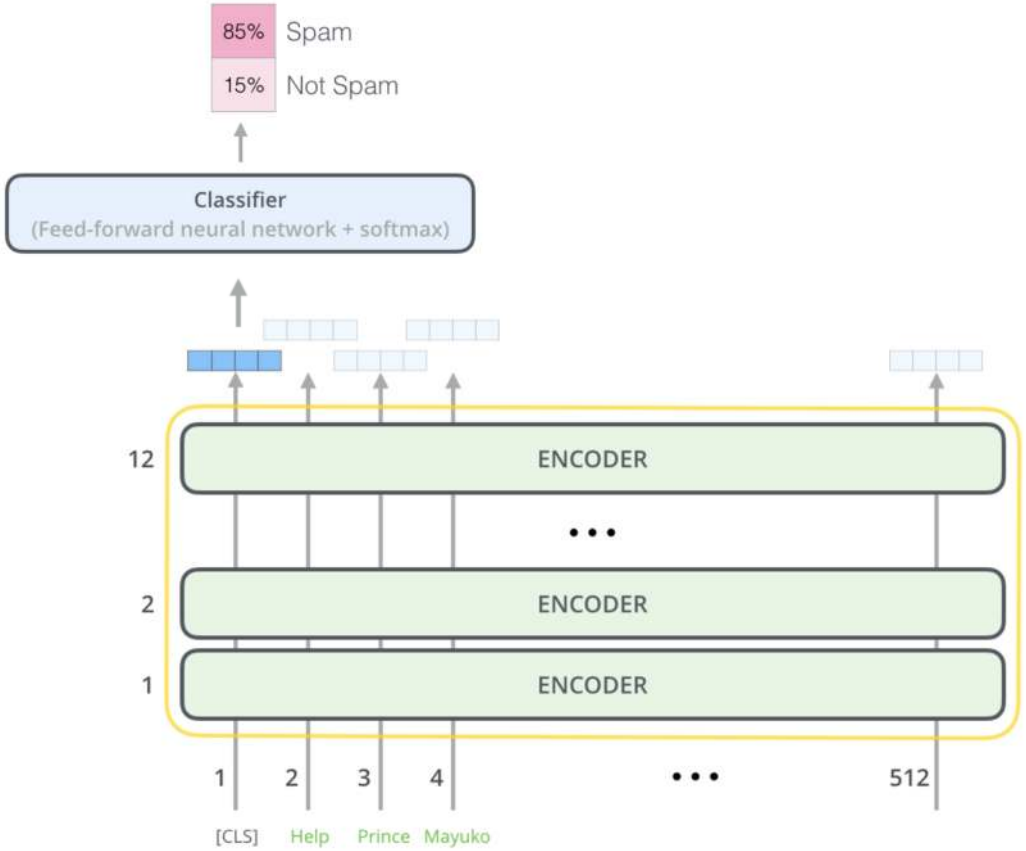
Supervised training (e.g. Email sentence classification)

- you mainly have to train the classifier, with minimal changes happening to the pre-trained model during the training phase (fine-tuning approach).



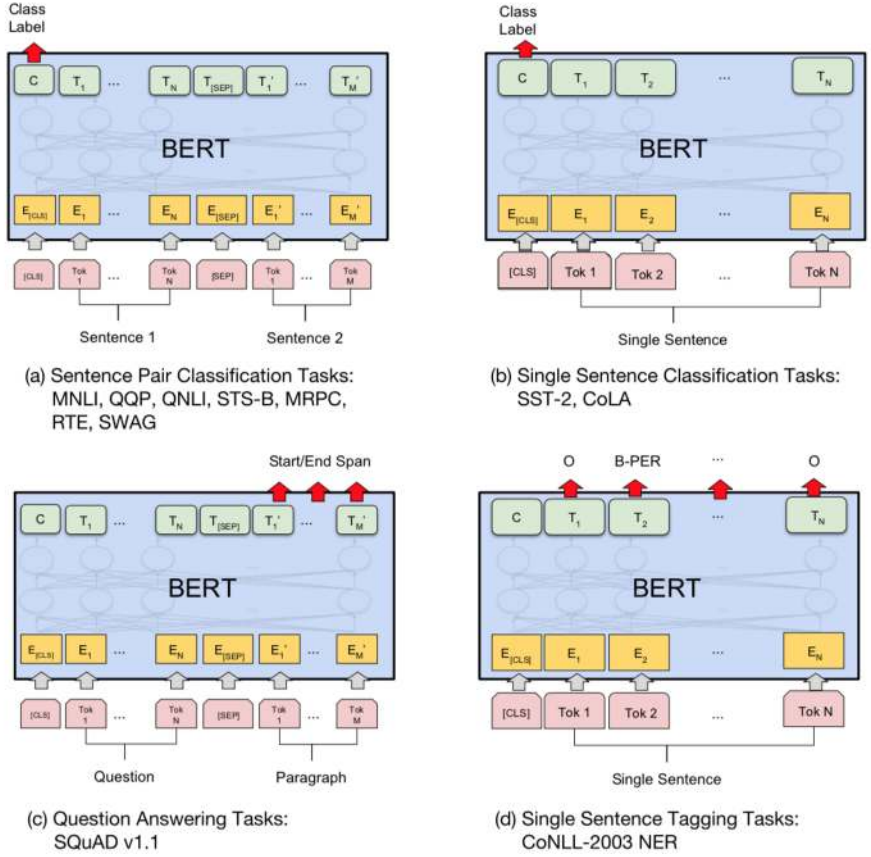
Phase2: Supervised Phase
E.g., Spam Email Classification

- BERT is basically a trained Transformer Encoder [1] 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.



[1] Vaswani et al., **Attention Is All You Need**, NIPS 2017, https://arxiv.org/pdf/1706.03762.pd

Phase2: Supervised Phase
E.g., other tasks



Devlin et al., **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**, arXiv 2018, https://arxiv.org/pdf/1810.04805.pdf

Pretrained BERT (English)

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

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)**

Pretrained BERT (Multilingual)

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

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

4) OpenAI GPT (Generative Pre-Training) [Radford, 2018]

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).

4) OpenAI GPT (Generative Pre-Training) [Radford, 2018]

GPT training procedure consists of 2 steps:

1. Unsupervised pre-training: (Unidirectional LM using Transformer)
   a. 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.

1. Supervised fine-tuning:
   a. 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:

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

   a. 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

OpenAI GPT (Generative Pre-Training)

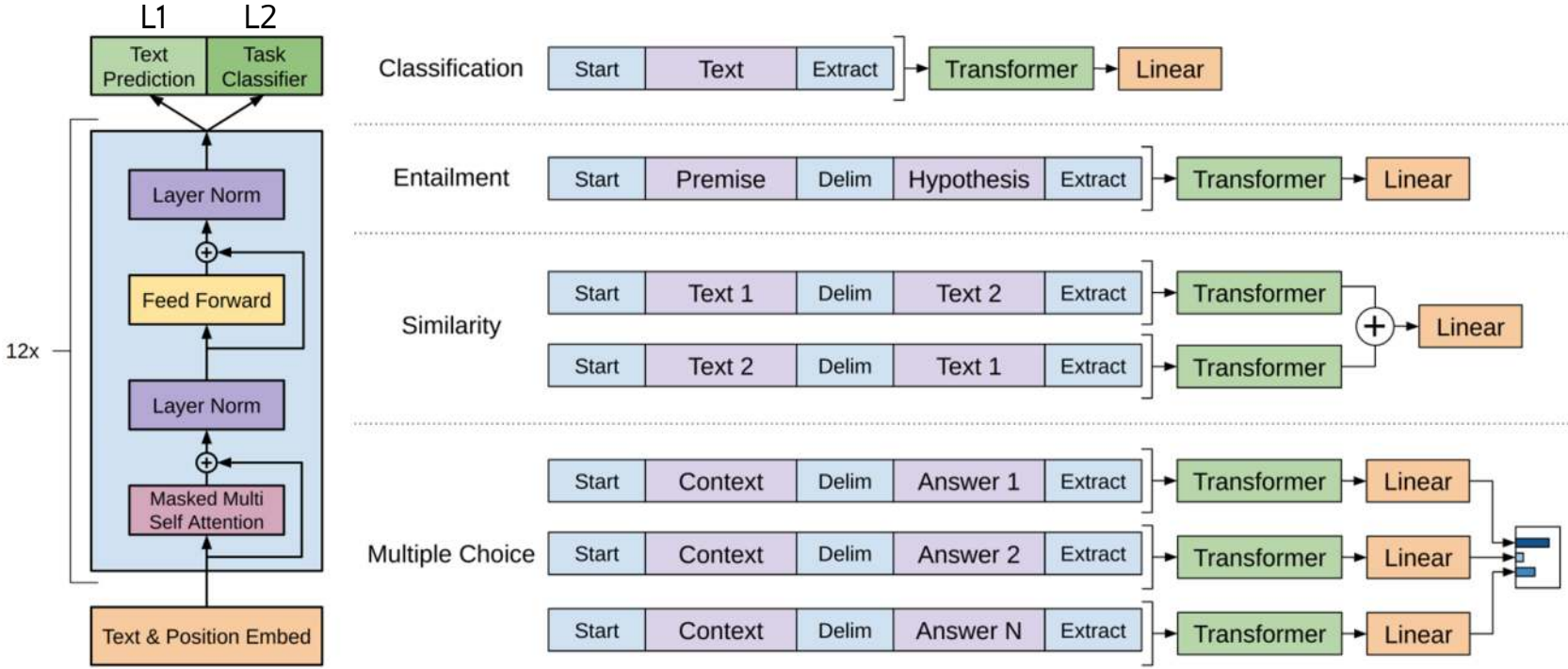
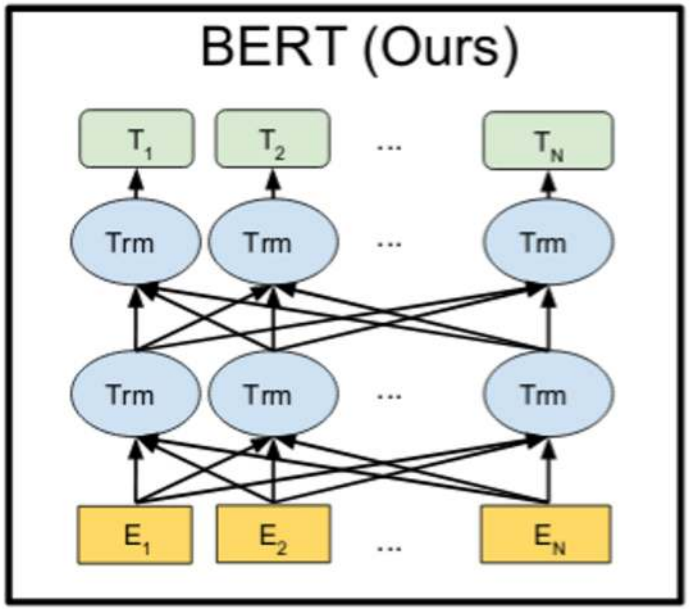
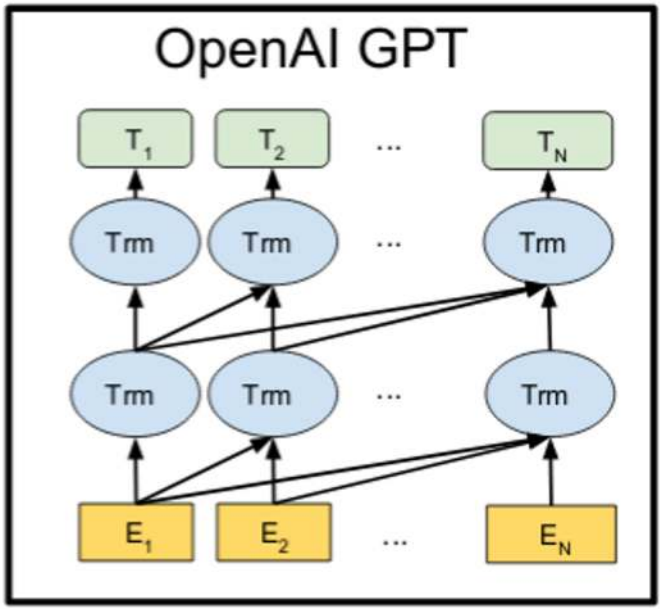


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT vs BERT





Different points:

- BERT (unsupervised phase)
- Masked LM vs Unidirect LM
- Next sentence prediction

- GPT (supervised learning)
- Auxiliary task

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 optimiser 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.

Reference: https://openai.com/blog/better-language-models/
https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2#:~:text=Model%20and%20Implementation%20details%3A%20The,from%201600%20for%20GPT%2D2.

Pretrained GPT-2 (English)

https://github.com/openai/gpt-2-output-dataset

gpt-2-output-dataset

This dataset contains:

- 250K documents from the WebText test set
- For each GPT-2 model (trained on the WebText training set), 250K random samples (temperature 1, no truncation) and 250K samples generated with Top-K 40 truncation

We look forward to the research produced using this data!

Download

For each model, we have a training split of 250K generated examples, as well as validation and test splits of 5K examples.

All data is located in Google Cloud Storage, under the directory `gs://gpt-2/output-dataset/v1`.

There, you will find files:

- `webtext.${split}.jsonl`
- `small-117M.${split}.jsonl`
- `small-117M-k40.${split}.jsonl`
- `medium-345M.${split}.jsonl`
- `medium-345M-k40.${split}.jsonl`
- `large-762M.${split}.jsonl`
- `large-762M-k40.${split}.jsonl`
- `xl-1542M.${split}.jsonl`
- `xl-1542M-k40.${split}.jsonl`

Pretrained GPT-3 (English)

README.md

GPT-3: Language Models are Few-Shot Learners

arXiv link

> Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and cloze tasks, as well as several tasks that require on-the-fly reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. At the same time, we also identify some datasets where GPT-3's few-shot learning still struggles, as well as some datasets where GPT-3 faces methodological issues related to training on large web corpora. Finally, we find that GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans. We discuss broader societal impacts of this finding and of GPT-3 in general.

Contents

https://github.com/openai/gpt-3

Transformers by HuggingFace

https://huggingface.co/transformers/index.html

Transformers

🤗 Transformers (formerly known as *pytorch-transformers* and *pytorch-pretrained-bert*) provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between TensorFlow 2.0 and PyTorch.

- Converting Tensorflow Checkpoints
  - BERT
  - OpenAI GPT
  - OpenAI GPT-2
  - Transformer-XL
  - XLNet
  - XLM

BERT	`bert-large-cased-whole-word-masking-fine`
	`bert-base-cased-finetuned-mrpc`
	`bert-base-german-dbmdz-cased`
	`bert-base-german-dbmdz-uncased`
	`bert-base-japanese`
	`bert-base-japanese-whole-word-masking`
	`bert-base-japanese-char`
	`bert-base-japanese-char-whole-word-mask`
	`bert-base-finnish-cased-v1`
	`bert-base-finnish-uncased-v1`
	`bert-base-dutch-cased`
GPT	`openai-gpt`
GPT-2	`gpt2`
	`gpt2-medium`
	`gpt2-large`
	`gpt2-xl`
Transformer-XL	`transfo-xl-wt103`
XLNet	`xlnet-base-cased`
	`xlnet-large-cased`

Proceedings of the 27th International Conference on Computational Linguistics, pages 1638–1649
Santa Fe, New Mexico, USA, August 20-26, 2018.

5) FLAIR

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.

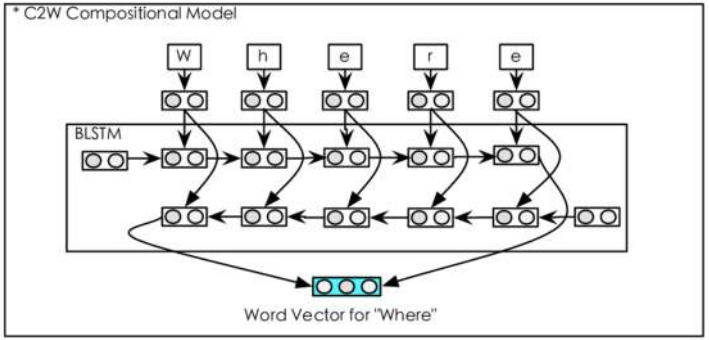
5) FLAIR

Contextual String Embeddings for Sequence Labeling

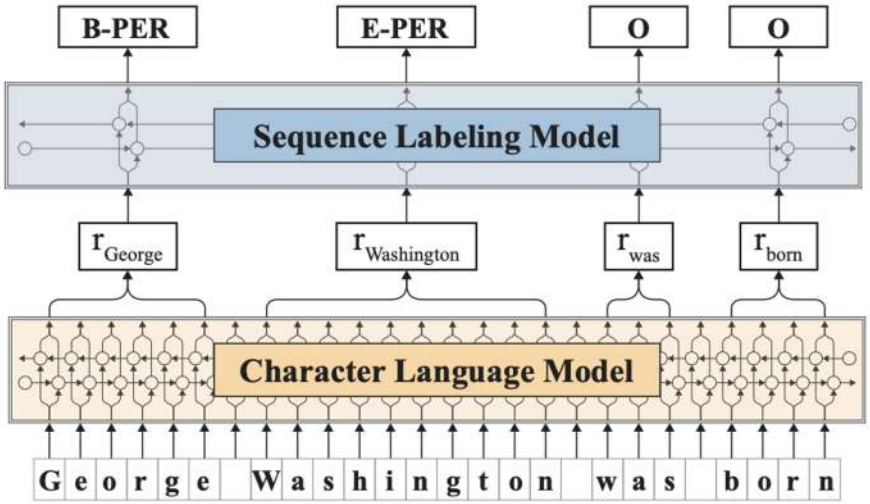
- Problem: Previously, each word (string) is embedded using its own characters **independent** from other words in the sentence.
- Propose: String Embeddings can be **contextualized** by their surrounding text.

Standard String Embeddings

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



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 published results. The proposed approach significantly outperforms previous work on the CoNLL03 NER task for German and English and slightly outperforms previous works on CoNLL2000 chunking and Penn treebank PoS tagging.

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

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	
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

- 1st is CNN+Attention
- 2nd, 3rd, 4th are based on Flair.

FLAIR (Multilingual)

https://github.com/flairNLP/flair

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), part-of-speech tagging (PoS), sense disambiguation and classification.
- **Multilingual.** Thanks to the Flair community, we support a rapidly growing number of languages. We also now include '*one model, many languages*' taggers, i.e. single models that predict PoS or NER tags for input text in various 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**, BERT embeddings and ELMo embeddings.
- **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.

Demo

https://github.com/cstorm125/thai2fit

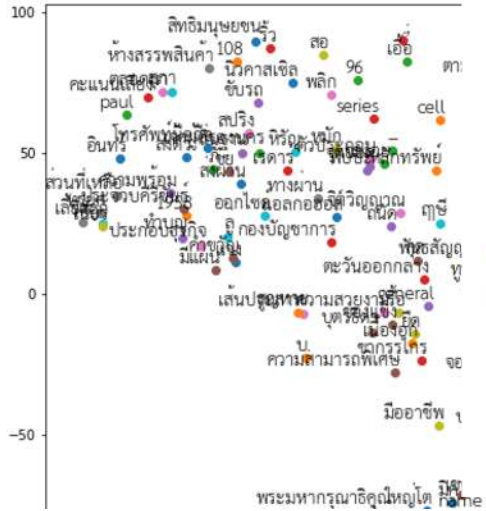
- Unsupervised LM on Thai Wikipedia Dump of 60,005 vocabs
- Original supervised task is a text classification on Wongnai Challenge

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.



v0.4 (In Progress)

- Replace AWD-LSTM/QRNN with tranformers-based models
- Named-entity recognition

Text Classification

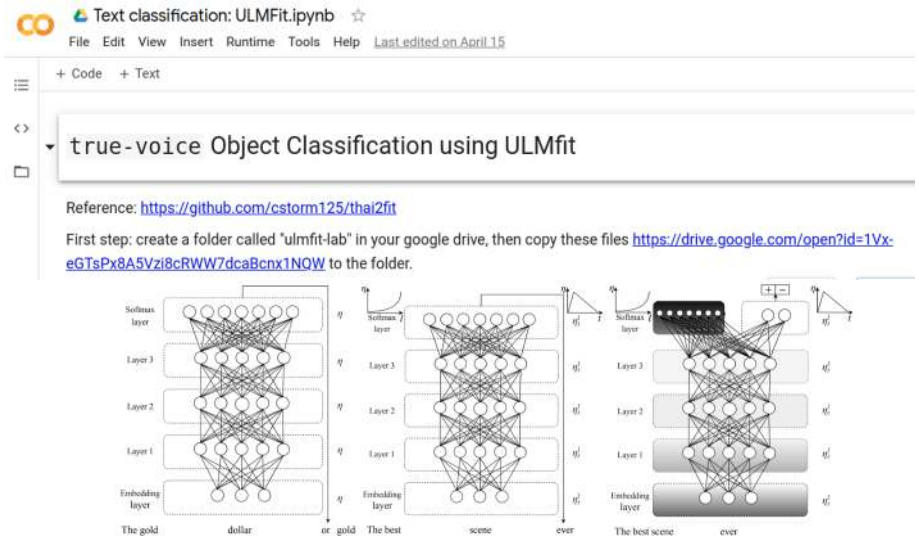
We trained the ULMFit model implemented by `thai2fit` for text classification. We use Wongnai Challenge: Review Rating Prediction as our benchmark as it is the only sizeable and publicly available text classification dataset at the time of writing (June 21, 2018). It has 39,999 reviews for training and validation, and 6,203 reviews for testing.

We achieved validation perplexity at 35.75113 and validation micro F1 score at 0.598 for five-label classification. Micro F1 scores for public and private leaderboards are 0.59313 and 0.60322 respectively, which are state-of-the-art as of the time of writing (February 27, 2019). FastText benchmark based on their own pretrained embeddings has the performance of 0.50483 and 0.49366 for public and private leaderboards respectively. See `ulmfit_wongnai.ipynb` for more details.

In-class demo: ULMFiT

https://drive.google.com/file/d/1iZvRpsw2HQvjOeOl3UBnxu5t0DMbzQgq/view?usp=sharing

Action/Object classification dataset from customer service phone calls transcribed by TrueVoice



Step1) load pretrained model on thai wikipedia corpus

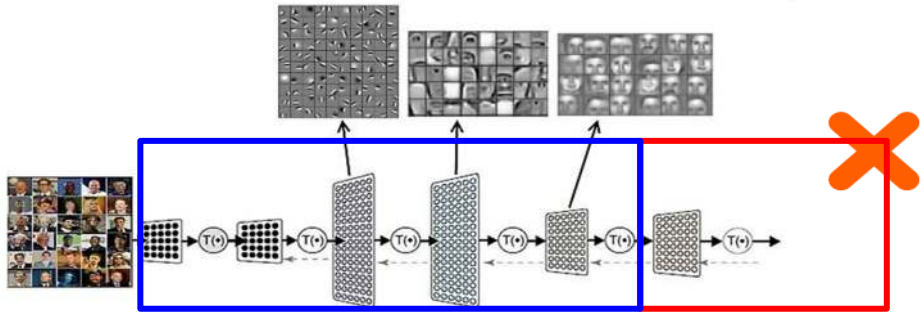
Step2) Fine-tune on LM task with TrueVoice data

Step3) Gradually unfreeze each layer to fine-tune on supervised task (object classification on TrueVoice data)

3 Conclusion

Transfer learning conclusion

- Low-resource/small training data is a common issue especially in NLP.
- Possible to train large models on small labeled data by using transfer learning and domain adaptation.
- Off the shelf features (fixed weights) work very well in various domains and tasks for smaller samples, but should be adapted to increase performance .
- Lower layers of network contain very generic features, higher layers more task specific features. Tasks that are very different are harder to transfer.
- **Supervised domain adaptation via fine tuning almost always improves performance.**



NLP conclusion

- In NLP, LM is a common unsupervised task, and Wiki corpus is a common corpus for unsupervised learning. Analogy, LM = image classification & Wiki = ImageNet.
- There are 5 recent pretrained LM's: ULMFiT, ELMO, BERT, GPT, FLAIR where BERT is the SoTA. There are two main phases: (1) unsupervised on LM task [pretrained and available for download] and (2) supervised on downstream task [transferred model for downstream task (fine-tune & inference)].
- 1) ULMFit comprises of one embedding layer and three LSTM layers, where each layer will be gradually unfreezed in a top-down fashion during the fine-tuning phrase,.
- 2) ELMo employs bidirectional LMs (separated forward and backward LMs) and a CNN-over-character embedding to create a word embedding vector -- this alleviates OOV issue.
- 3) BERT is based on Transformer (attention) with two unsupervised objectives: (1) mask language model (word-level embedding vector) & (2) next sentence prediction (sentence embedding vector).
- 4) GPT is an unidirectional transformer-based model. LM is also used as an auxiliary objective during supervised training phrase.
- 5) FLAIR is a pre-trained bidirectional character LM.

References

- K. McGuinness, Transfer Learning (D2L4 Insight@DCU Machine Learning Workshop 2017)
- https://medium.com/dair-ai/a-light-introduction-to-transfer-learning-for-nlp-3e2cb56b48c8
- http://jalammar.github.io/illustrated-bert/