AdvNLE Lecture 9

Transfer Learning with Pretrained Large Language Models

Dr Julie Weeds, Spring 2024

Previously …

- Paraphrase and semantic matching
  - applications
- Simple text matching
- Distributional representations of meaning
- Composition
- Contextualised word embeddings
  - ELMo
  - transformers
  - BERT

The Distributional Hypothesis: for words

"Words that occur in the same contexts tend to have similar meanings."

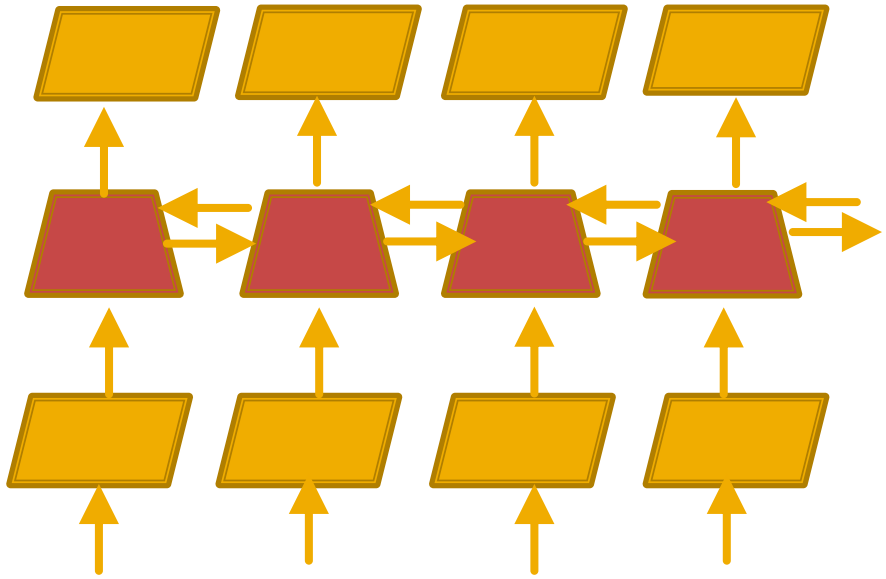
Harris, 1954

Words represented by real-valued vectors, where dimensions explicitly or implicitly represent contexts in which the word occurs:

chicken	0.3	0.1	**0.8**	….	0.5
beef	0.05	0.9	**0.75**	….	0.01

Contextualised Word Embeddings

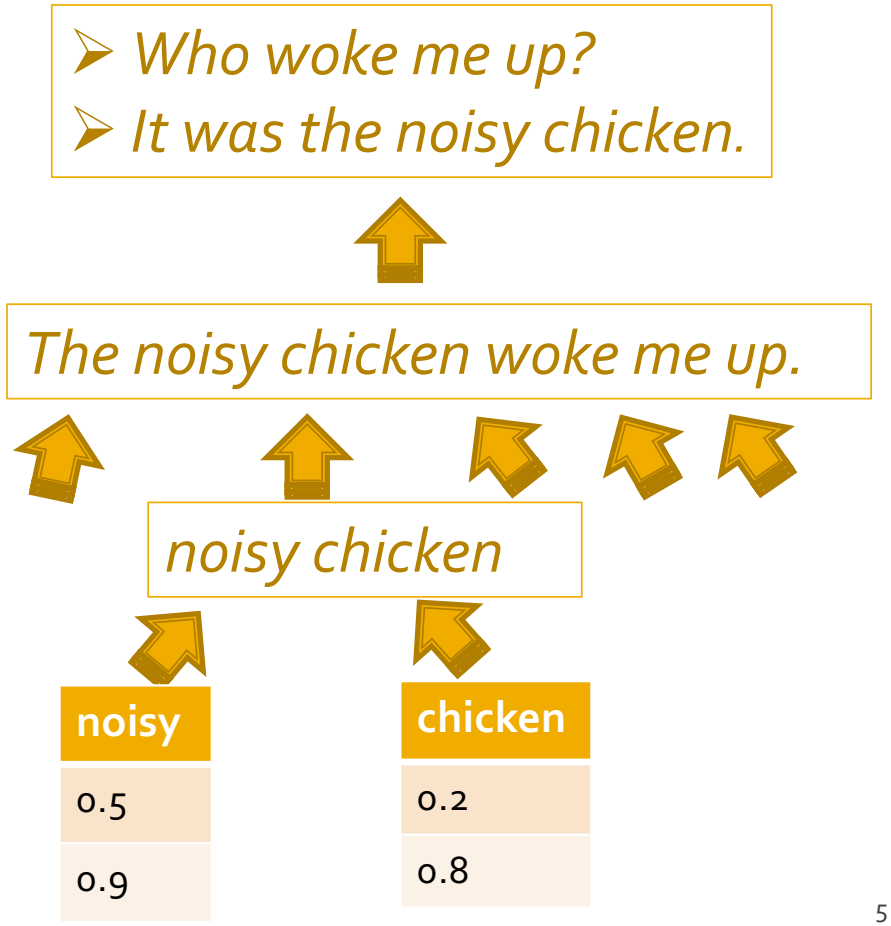
- ELMo (Peters et al. 2018)
- BERT (Devlin et al. 2018/2019)
- combine word embeddings and bi-directional language models to provide (deep) contextualised word embeddings



Beyond words

We have a model for **words**, how does this scale up to

- phrases
- sentences
- utterances
- documents
- discourse



Principle of compositionality

"*The meaning of a complex expression is determined by the meaning of its constituent parts and the rules used to combine them.*"

[Boole; Frege and others]

Can we take distributional word representations and combine them to make distributional sentence representations?

The distributional hypothesis: for sentences?

*"**Words** that occur in the same contexts tend to have similar meanings."*

*"**Phrases** that occur in the same contexts tend to have similar meanings."*

*"**Sentences** that occur in the same contexts tend to have similar meanings."*

Sentential contexts

- Contexts of sentences are other sentences
- But most composition methods assume that the possible contexts of a sentence is some function of the possible contexts of all of the constituent words
- Word representations are often **simply added or averaged** to get *composed* sentential representations (Mitchell and Lapata, 2010)

SBert (Reimers and Gurevych 2019)

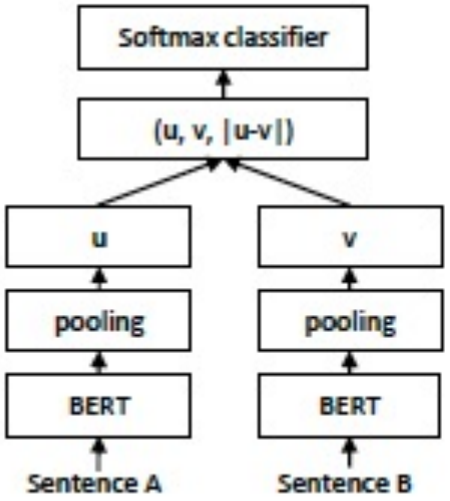


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

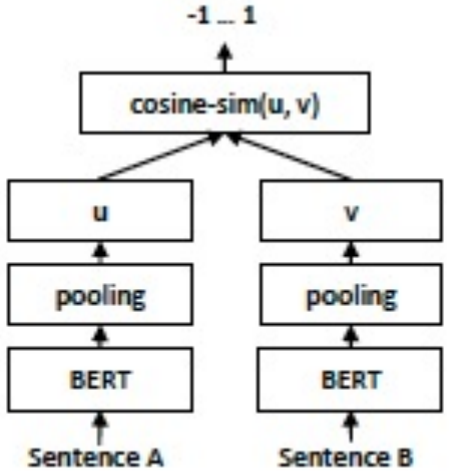


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Today

- Transfer learning
- Fine-tuning BERT-based models for
  - text classification
  - sequence labelling
- The BERT family
- More distant relatives
- Schick and Schütze (2021)

Transfer Learning through Fine-tuning

- **Transfer learning:** acquiring knowledge from one task or domain and then applying (transferring) it to solve a new task
- BERT-based models acquire knowledge about language through **pre-training** (masked language model prediction and next sentence prediction) on large unannotated corpora
- **Fine-tuning** is the process of transferring this knowledge to a specific task

Fine-tuning

- Fine-tuning **uses labelled data** from the application to train **additional application-specific parameters**
- Application-specific parameters could be a single layer neural network on top of the BERT architecture
- Fine-tuning might
  - freeze the pre-trained language model parameters
  - allow updates to be made to some or all of the pre-trained language model parameters

Text classification with BERT

- E.g., Is a review **positive** or **negative** in sentiment?
- [CLS] token is used to stand for the entire sequence
- [CLS] is part of the vocabulary and must be pre-pended to all input sequences during pre-training and fine-tuning
- [CLS] is the input to a classifier head, e.g., logistic regression, which makes relevant decision

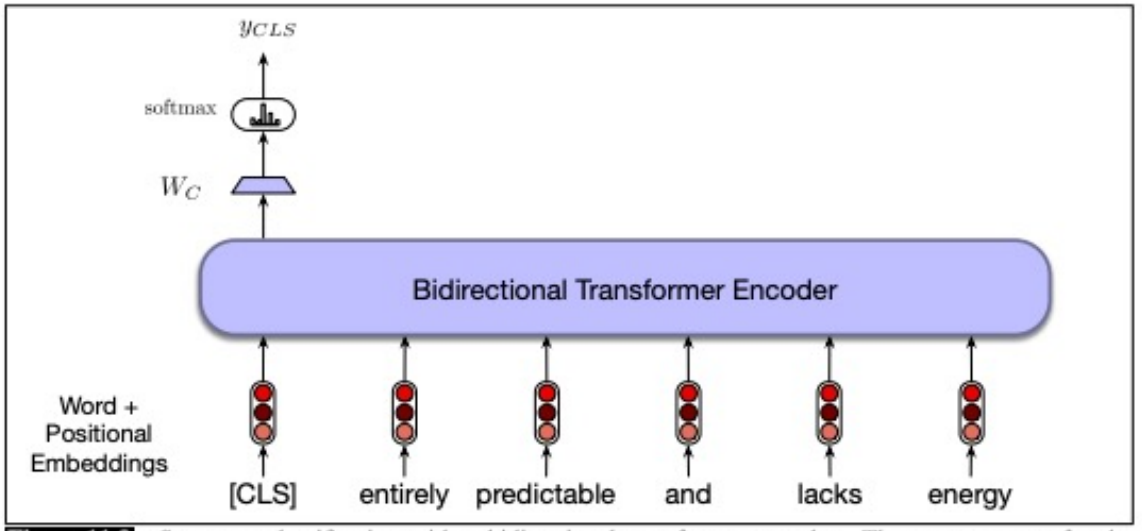


Figure 11.8 Sequence classification with a bidirectional transformer encoder. The output vector for the [CLS] token serves as input to a simple classifier.

Figure taken from Jurafsky and Martin (Section 11.3.1)

Text classification with BERT (2)

- learn a set of weights W_C which maps output vector for [CLS] to a set of scores for the possible sentiment classes
- Pass input text through pre-trained language model to generate $\mathbf{y_{CLS}}$, multiply it by $\mathbf{W_C}$ and then pass this through softmax

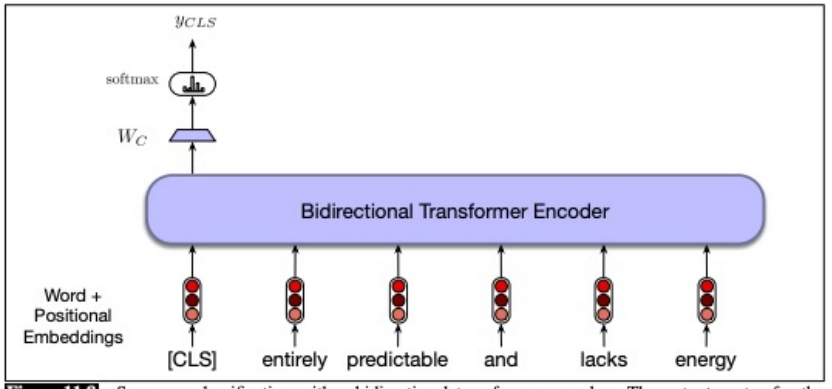


Figure 11.8 Sequence classification with a bidirectional transformer encoder. The output vector for the [CLS] token serves as input to a simple classifier.

$$y = \text{softmax}(\boldsymbol{W_C y_{CLS}})$$

- Fine-tuning requires input sequences labelled with appropriate class
- Cross-entropy loss between softmax output and correct answer drives backpropagation
- Weights can be updated just for $\mathbf{W_C}$ (pre-trained language model is frozen) or in the pre-trained language model as well

In practice … with Huggingface transformers library

- transformers.BertForSequenceClassification
  - BERT transformer with a sequence classification / regression head on top (linear layer on top of **pooled output)**
  - inherits from PreTrainedModel

```
:
from transformers import BertTokenizerFast, BertForSequenceClassification
from transformers import Trainer, TrainingArguments

# check text classification models here: https://huggingface.co/models?filter=text-classification
model_name = "bert-base-uncased"

# load the tokenizer
tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lower_case=True)

# load the model and pass to CUDA
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=len(target_names)).to("cuda")
```

Downloading: 100%  466k/466k [00:00<00:00, 756kB/s]

Training the Sequence Classification Model

- See https://www.thepythoncode.com/article/finetuning-bert-using-huggingface-transformers-python
- Need to load, tokenize and encode the inputs
  - training and validation set
- Set up training_args e.g., number of epochs, batch size
- Set up metrics for evaluation e.g., accuracy

```
trainer = Trainer(
    model=model,                         # the instantiated Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=valid_dataset,          # evaluation dataset
    compute_metrics=compute_metrics,     # the callback that computes metrics of interest
)

# train the model
trainer.train()
```

Training arguments

```
args = TrainingArguments(
    f"{model_name}-finetuned-{task}",
    evaluation_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=5,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model=metric_name,
    push_to_hub=True,
)
```

e.g., “pearson”, “accuracy”, “f1” or “spearmanr”

The GLUE Benchmark tasks

The GLUE Benchmark is a group of nine classification tasks on sentences or pairs of sentences which are:

- CoLA (Corpus of Linguistic Acceptability) Determine if a sentence is grammatically correct or not.is a dataset containing sentences labeled grammatically correct or not.
- MNLI (Multi-Genre Natural Language Inference) Determine if a sentence entails, contradicts or is unrelated to a given hypothesis. (This dataset has two versions, one with the validation and test set coming from the same distribution, another called mismatched where the validation and test use out-of-domain data.)
- MRPC (Microsoft Research Paraphrase Corpus) Determine if two sentences are paraphrases from one another or not.
- QNLI (Question-answering Natural Language Inference) Determine if the answer to a question is in the second sentence or not. (This dataset is built from the SQuAD dataset.)
- QQP (Quora Question Pairs2) Determine if two questions are semantically equivalent or not.
- RTE (Recognizing Textual Entailment) Determine if a sentence entails a given hypothesis or not.
- SST-2 (Stanford Sentiment Treebank) Determine if the sentence has a positive or negative sentiment.
- STS-B (Semantic Textual Similarity Benchmark) Determine the similarity of two sentences with a score from 1 to 5.
- WNLI (Winograd Natural Language Inference) Determine if a sentence with an anonymous pronoun and a sentence with this pronoun replaced are entailed or not. (This dataset is built from the Winograd Schema Challenge dataset.)

See https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/text_classification.ipynb

Compute metrics

In [ ]:
```
import numpy as np
from datasets import load_metric
metric = load_metric("glue","sst-2")

fake_preds=np.random.randint(0,2,size=(64,))
fake_labels=np.random.randint(0,2,size=(64,))
metric.compute(predictions=fake_preds,references=fake_labels)
```

In [ ]:
```
def compute_metrics(eval_pred):
    predictions,labels=eval_pred
    # more code here if needed to pre-processs predictions /labels into lists e.g.,select appropriate column
    return metric.compute(predictions=predictions,references=labels)
```

Using the Sequence Classification Model

- Evaluate it using the trainer's evaluate() method
- Save the associated model and tokenizer for future use
- Make predictions on unseen data

```
:
def get_prediction(text):
    # prepare our text into tokenized sequence
    inputs = tokenizer(text, padding=True, truncation=True, max_length=max_length, return_tensors="pt").to("cuda")
    # perform inference to our model
    outputs = model(**inputs)
    # get output probabilities by doing softmax
    probs = outputs[0].softmax(1)
    # executing argmax function to get the candidate label
    return target_names[probs.argmax()]

print(get_prediction(text))
```

Drawbacks

- BertForSequenceClassification is quite **opaque**
  - Dig through lots of code to find out the architecture of the classification head
  - What is the "pooled output"?
  - It is the CLS representation but this is not clear in the documentation and you might want to use a different pooling strategy
- You need to be signed up to huggingface-hub to be able to use all of the functionality (e.g., load_metric) which increases the burden on getting started
- **ALTERNATIVE:** build own classification head on top of pre-trained BERT model
  - **See lab**

Code for BERTclassifier (from lab)

```
]: #now we need to put a simple classification layer on top of BERT

from torch import nn
from transformers import BertModel

class BertClassifier(nn.Module):

    def __init__(self,dropout=0.5,num_classes=2):

        super(BertClassifier,self).__init__()

        self.bert=BertModel.from_pretrained('bert-base-uncased')
        self.dropout=nn.Dropout(dropout)
        self.linear=nn.Linear(768,num_classes)
        self.relu=nn.ReLU()

    def forward(self,input_id,mask):

        last_hidden_layer,pooled_output = self.bert(input_ids=input_id,attention_mask=mask,return_dict=False)
        dropout_output=self.dropout(pooled_output)
        linear_output=self.linear(dropout_output)
        final_layer=self.relu(linear_output)

        return final_layer
```

Freezing layers

In [ ]:
```
model=BERTClassifier(num_classes=len(labels.keys()))

#this will freeze the pre-trained BERT model and just make the classification head trainable
#can speed things up and avoid "catastrophic forgetting" / overfitting on task-specific data
model.bert.requires_grad_(False)
```

Pairwise Sequence Classification

- Do two sentences entail or contradict each other?
  - "I'm confused"
  - "It is not completely clear to me"
- Concatenate the sentences, separated by the [SEP] token
- Proceed as for single sequence classification
- OR use something like SBERT. What's the advantages of each approach?

Sequence Labelling with BERT

- E.g., POS tagging or NER
- Simplest approach is to pass each output from BERT to a simple classifier
- Or pass it to a CRF which considers tag-level transitions as well

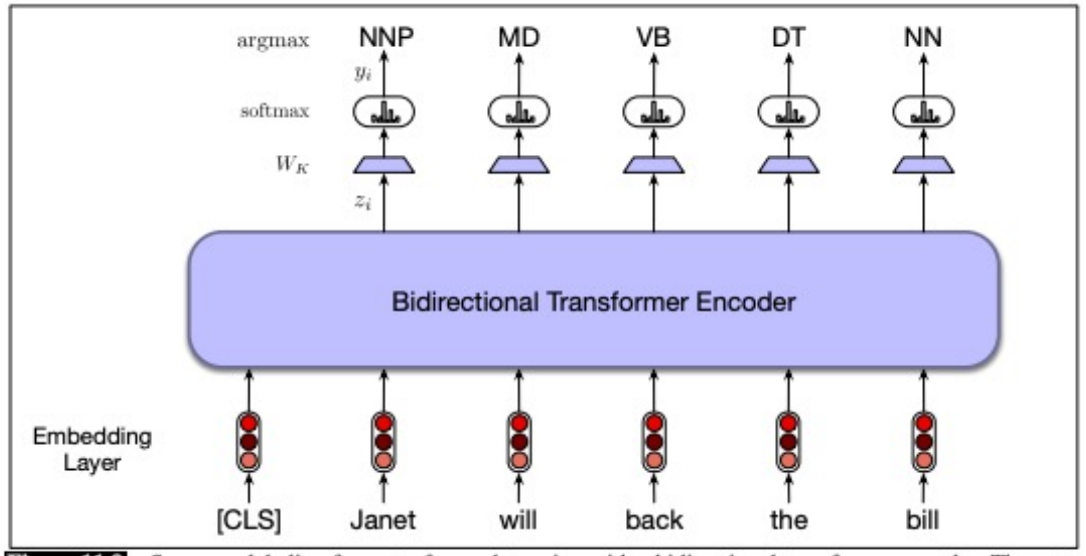


Figure 11.9 Sequence labeling for part-of-speech tagging with a bidirectional transformer encoder. The output vector for each input token is passed to a simple k-way classifier.

- Complications can arise from subword tokenizations
  - simplest approach is to use the first subword token from each word

Sequence Labelling … with Huggingface Transformers

- https://huggingface.co/docs/transformers/tasks/token_classification

```
>>> from transformers import AutoModelForTokenClassification, TrainingArguments, Trainer

>>> model = AutoModelForTokenClassification.from_pretrained("distilbert-base-uncased", num_labels=2)
```

BERT Family

- RoBERTa
- AlBERT
- DistilBERT
- SciBERT
- MBert
- and more
  - StructBERT
  - AraBERT
  - DeBERTa

RoBERTa

- Alternative to BERT from Facebook (Liu et al (2019)): a **r**obustly **o**ptimized **BERT** pretraining **a**pproach
- Pre-trained on even larger corpus
- Ditches next sentence prediction and just uses MLM for pre-training
  - dynamic masking (different masks each time sentence is used)
  - Full-sentences without NSP loss
  - large mini-batches (8K sequences)
  - larger byte-level BPE (50K subword units, 15M-20M additional parameters)

AlBERT

- **A** **L**ite **B**ERT for Self-supervised learning of language representations (Lan et al. 2019)
- Incorporates 2 parameter reduction techniques with a view to making pre-trained models more scalable
  - factorization of the vocabulary embedding matrix
  - cross-layer parameter sharing
- Performance also improved with sentence order prediction task
- 18x fewer parameters than BERT and trained about 1.7x faster

DistilBERT

- a distilled version of BERT: smaller, faster, cheaper and lighter (Sanh et al. 2019)
- knowledge distillation during pre-training
- a compact model (the student) is trained to reproduce the behaviour of a larger model (the teacher) or ensemble of models
  - reduces size by 50%
  - retains 97% of language understanding
  - 60% faster

SciBERT

- SciBERT: a pretrained language model for scientific text (Beltagy et al. 2019)
- basically BERT pre-trained on large multi-domain corpus of scientific publications
- Improved in-domain results

MBert

- Multilingual BERT, model released by Devlin et al. at the same time as BERT
- See Pires et al (2019): How multilingual is Multilingual BERT
- trained on text from 104 languages
- does not contain any explicit translation information
- Intuition is that embeddings for words in different languages will be embedded in the same space due to commonalities in the vocabularies for the languages (e.g., names, numbers and other shared vocab)

More Distant Relatives of BERT

- Other Pretrained Large Language Models, generally still based on transformers e.g.,
  - GPT,
  - Turing-NLG,
  - T5,
  - XLNet,
  - Electra

Generative Pre-trained Transformer 3 (GPT-3)

- Brown et al. 2020: *Language Models are Few Shot Learners*
- autoregressive language model
  - this means it predicts the next token rather than masked tokens
- variable length inputs but uni-directional in nature
- largest non-sparse language model: 175 billion parameters, 10x bigger than competitors
- Trained on Common Crawl, WebText2, Books1, Books2 and Wikipedia
- No fine-tuning. Used to generate answers using a few-shot training / prompting paradigm

In-context learning / Prompting

- Zero-shot: model predicts the answer given only a natural language description of the task. No parameter updates.
  - "Translate English to French: cheese => _____"
- One-shot: model sees the description and one example of the task. No parameter updates.
  - "Translate English to French: sea otter => loutre de mer, cheese => _____"
- Few-shot: model sees the description and a few examples of the task. No parameter updates
  - "Translate English to French: sea otter => loutre de mer, peppermint => menth poivree, plush girafe => girafe peluche, cheese => _____"

Text-to-Text Transfer Transformer (T5)

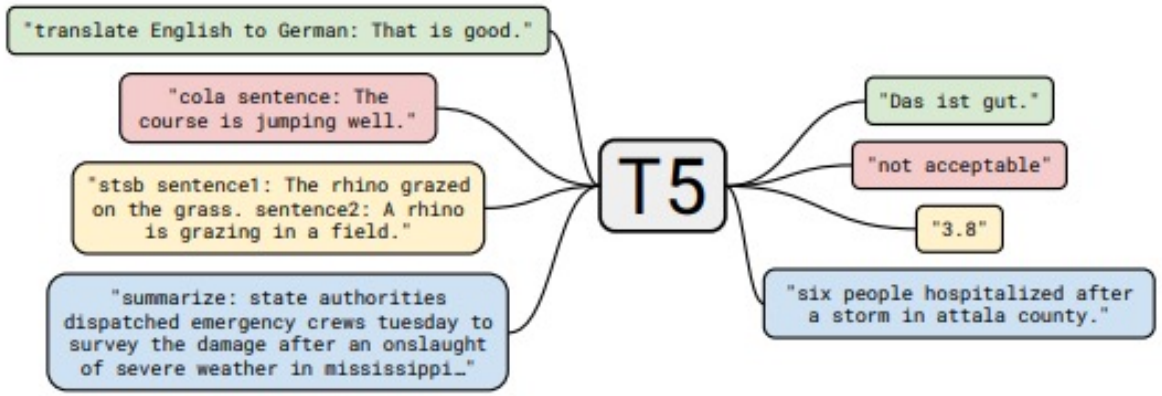
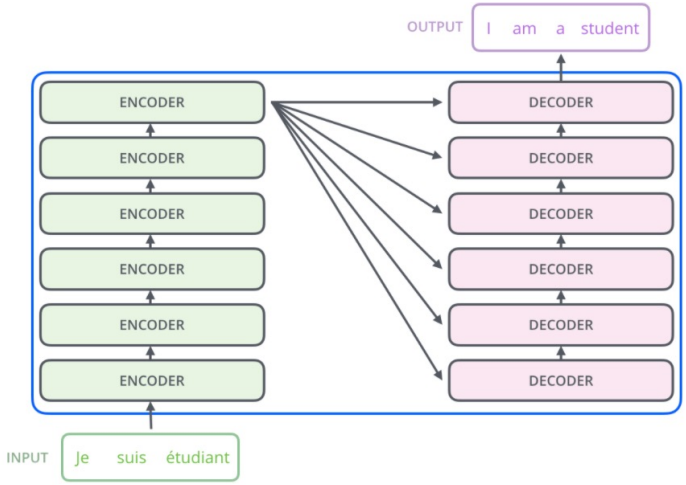


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".

- Raffel et al. 2020: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

T5 Architecture

- Encoder-decoder architecture closely following the original proposal by Vaswani et al. 2017
- Used for generation as well as encoding
- Every task is structured in such a way that the answer (be it a label or a translation) should be generated by the model i.e., prediction is done in an autoregressive way



T5 also uses MLM

- Pre-training involves unsupervised objectives which are similar to the MLM of BERT and "word dropout" regularization technique (Bowman et al. 2015)

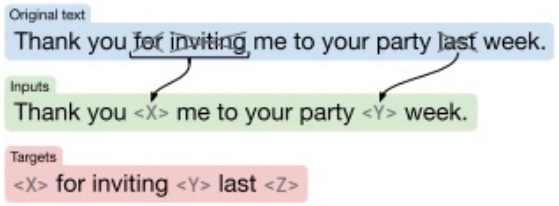


Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as **<X>** and **<Y>**) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel **<X>**. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token **<Z>**.

T5 Fine-Tuning

- Fine-tuned in a **supervised** fashion on a large number of tasks
- "unified" in the sense that all tasks can benefit from learning on other tasks
- To make best use of a pre-trained and fine-tuned T5 model, you need to know the prompt format which most closely matches your task e.g.,
  - input ="summarize: text."
  - output = "summary"

Next time

- ChatGPT
- Generative applications and LLMs
  - Machine translation
  - Summarisation
- Trustworthy and responsible LLMs / AI
- Environmental impact of LLMs / AI

Seminar Reading

- Schick and Schütze (2021): Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference

Further reading

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. in *NAACL 2019* https://www.aclweb.org/anthology/N19-1423/
- Erk, K. and Padó, S. (2008) A structured vector space model for word meaning in context. In *EMNLP* 2008
- Levy, O., and Goldberg, Y. (2014) Dependency-based word embeddings. *In ACL 2014*
- Marcheggiani, D. and Titov, I. (2017). Encoding Sentences with Graph Convolutional Networks for Semantic Role Labelling. In *Proceedings of EMNLP* 2017. https://www.aclweb.org/anthology/D17-1159/
- Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. In *Cognitive Science*, 34(8):1388–1429.
- Padó, S. and Lapata, M., (2007). Dependency-based construction of semantic space models. In *Computational Linguistics*, 33(2):161–199,
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *NAACL 2018*
- Weir, D., Weeds, J., Reffin, J. and Kober, T. (2016). Aligning packed dependency trees: a theory of composition for distributional semantics. In *Computational Linguistics*.