Transfer Learning in NLP

Dr. Ahmad El Sallab
AI Senior Expert

Agenda

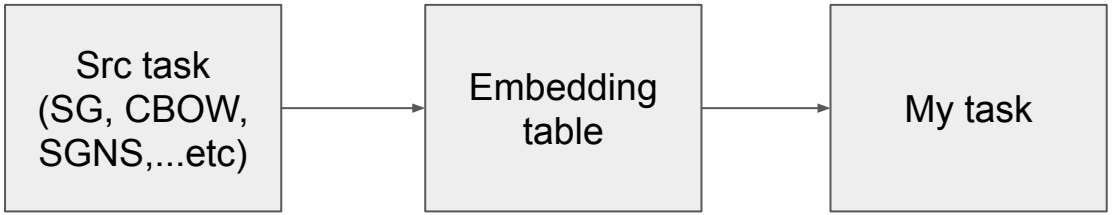
- TL with Pre-trained Word Embeddings
- TL from LMs: NLP ImageNet moment
- ULMFiT
- BERT
- GPT (1-2-3)
- XLTrans and XLNet
- RoBERTa

Word level TL

How to learn Embeddings?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?

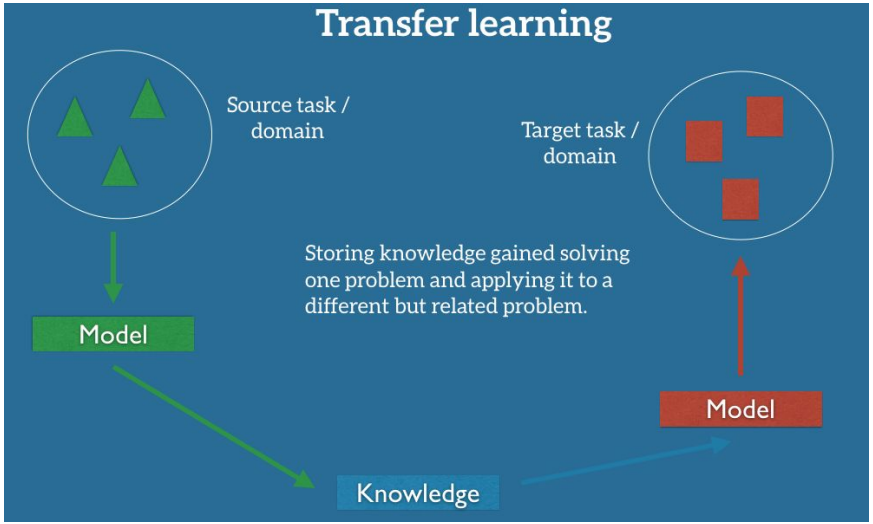
Pre-trained Word Embeddings



- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible

How to choose the src task?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?
- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible



How to learn Embeddings?

- End-to-end from current task
- Pre-trained → transfer to current task → Like what?
- Respect TL rules
  - Needs to be generic task, related to most NLP tasks
  - Needs to be trained on as much data as possible

Similarity between both domains

New dataset is small and similar to original dataset	***New dataset is large and similar to the original dataset.***
Max Reuse, No Fine-tune	**Max Reuse, Fine-tune**
New dataset is small but very different from the original dataset	***New dataset is large and very different from the original dataset***
Min Reuse, No Fine-tune	**NoReuse, Fine-tune No need to transfer**

New dataset size

ELMo

Step towards Sentence TL

ELMo

ELMO provides contextualized vectors.

The word2vec provides the same vector regardless the context of the word.

Contextualized vectors, learned by mixing the current word vector and its neighbors.

If we're using the GloVe representation, then the word "stick" would be represented by this vector no-matter what the context was, but "stick"" has multiple meanings depending on where it's used.

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 bi-directional LSTM trained on a specific task to be able to create those embeddings.

ELMO can work at char level, which reduces the OOV rate.

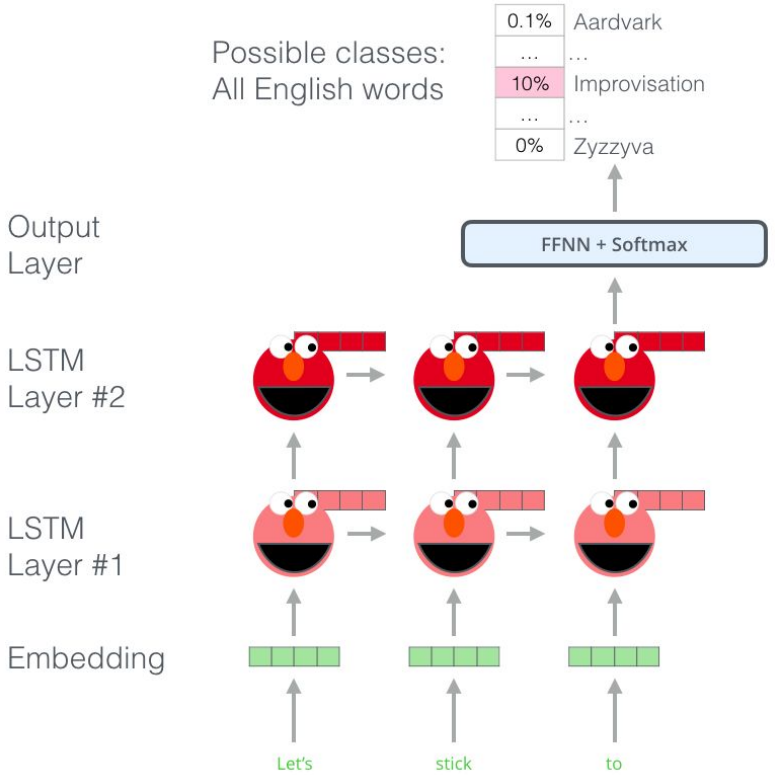
http://jalammar.github.io/images/elmo-embedding-robin-williams.png

What's ELMo's secret?

ELMo = Embeddings from Language Models

ELMo gained its language understanding from being trained to predict the next word in a sequence of words - a task called **Language Modeling**.

Self-supervised: This is convenient because we have vast amounts of text data that such a model can learn from without needing labels.



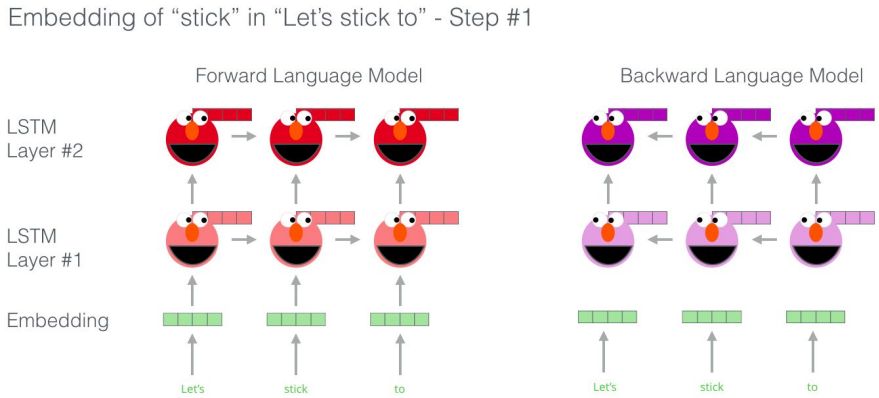
http://jalammar.github.io/images/Bert-language-modeling.png

What's ELMo's secret?

ELMo = Embeddings from Language Models

ELMo gained its language understanding from being trained to predict the next word in a sequence of words - a task called **Language Modeling**.

Bi-directional: ELMo actually goes a step further and trains a bi-directional LSTM – so that its language model doesn't only have a sense of the next word, but also the previous word.

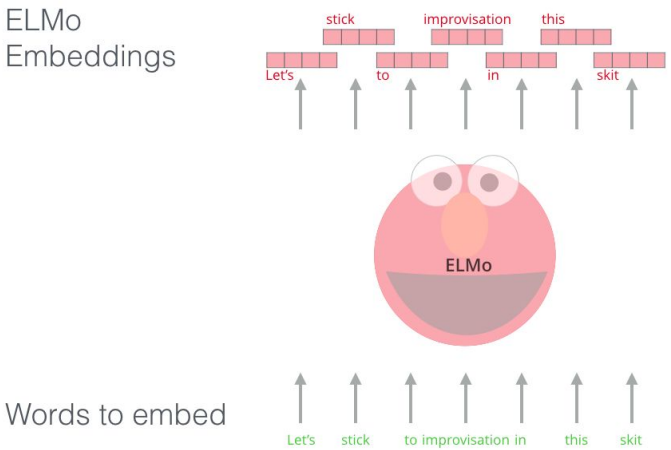


http://jalammar.github.io/images/elmo-forward-backward-language-model-embedding.png

ELMo = Sentence training ⇒ Word embeddings

ELMo is trained on LM task → So it models the **whole sentence** using BiLSTM

However, when it is used to get Embeddings, only the specific word vector is retained | given its context



http://jalammar.github.io/images/elmo-word-embedding.png

Char level Embeddings

- Word level Embeddings suffer an issue called Out-Of-Vocabulary (OOV).
- Representing the word as sequence of chars might solve the issue, but also might produce invalid words
- Another advantage of ELMo is using a mix of char level and word level models, called the sub-word level, which reduces the OOV.
- Since ELMo uses a sequence model (BiLSTM), it's no issue to encode the word as a sequence of chars.
-

Let’s code

https://www.kaggle.com/ahmadelsallab/elmo-keras

TFHub

TensorFlow Hub is a repository of reusable TensorFlow machine learning modules.

Catalogue: https://tfhub.dev/

```
import tensorflow as tf
import pandas as pd
import tensorflow_hub as hub
```

```
elmo_path = '../input/elmo-tfhub/2/'
```

```
self.elmo = hub.Module(elmo_path,
                       trainable=self.trainable,
                       name="{}_module".format(self.name))
```

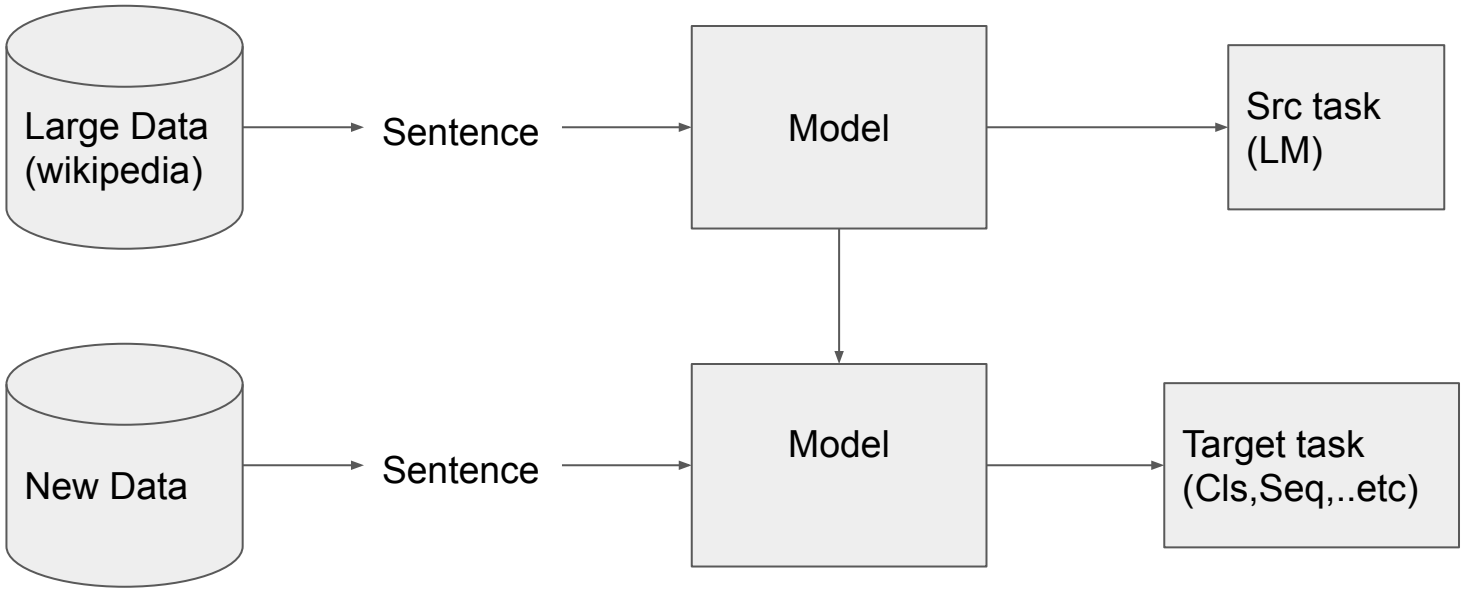
```
class ElmoEmbeddingLayer(Layer):
    def __init__(self, **kwargs):
        self.dimensions = 1024
        self.trainable=True
        super(ElmoEmbeddingLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.elmo = hub.Module(elmo_path,
                               trainable=self.trainable,
                               name="{}_module".format(self.name))

        self.trainable_weights += K.tf.trainable_variables(scope="^{}_module/.*".format(se
lf.name))
        super(ElmoEmbeddingLayer, self).build(input_shape)

    def call(self, x, mask=None):
        result = self.elmo(K.squeeze(K.cast(x, tf.string), axis=1),
                           as_dict=True,
                           signature='default',
                           )['default']
        return result
```

```
# Function to build model
def build_model():
    input_text = layers.Input(shape=(1,), dtype="string")
    embedding = ElmoEmbeddingLayer()(input_text)
    dense = layers.Dense(256, activation='relu')(embedding)
    pred = layers.Dense(1, activation='sigmoid')(dense)

    model = Model(inputs=[input_text], outputs=pred)

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()

    return model
```

Sentence level TL

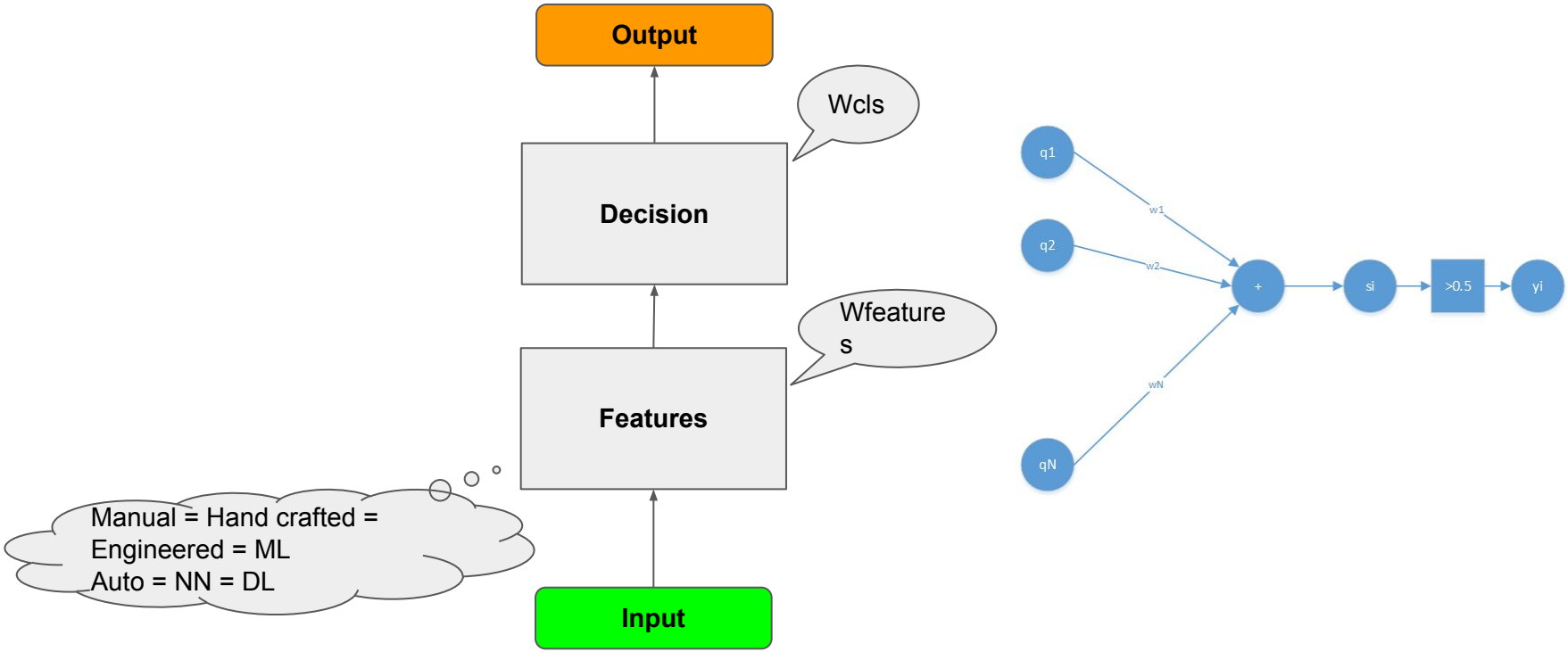
Sentence Embedding

But why we care about individual words vectors, while we can already model the whole sequence/sentence?



NLP ImageNet moment

Supervised Learning Model Design Pattern



Encoder-Decoder pattern

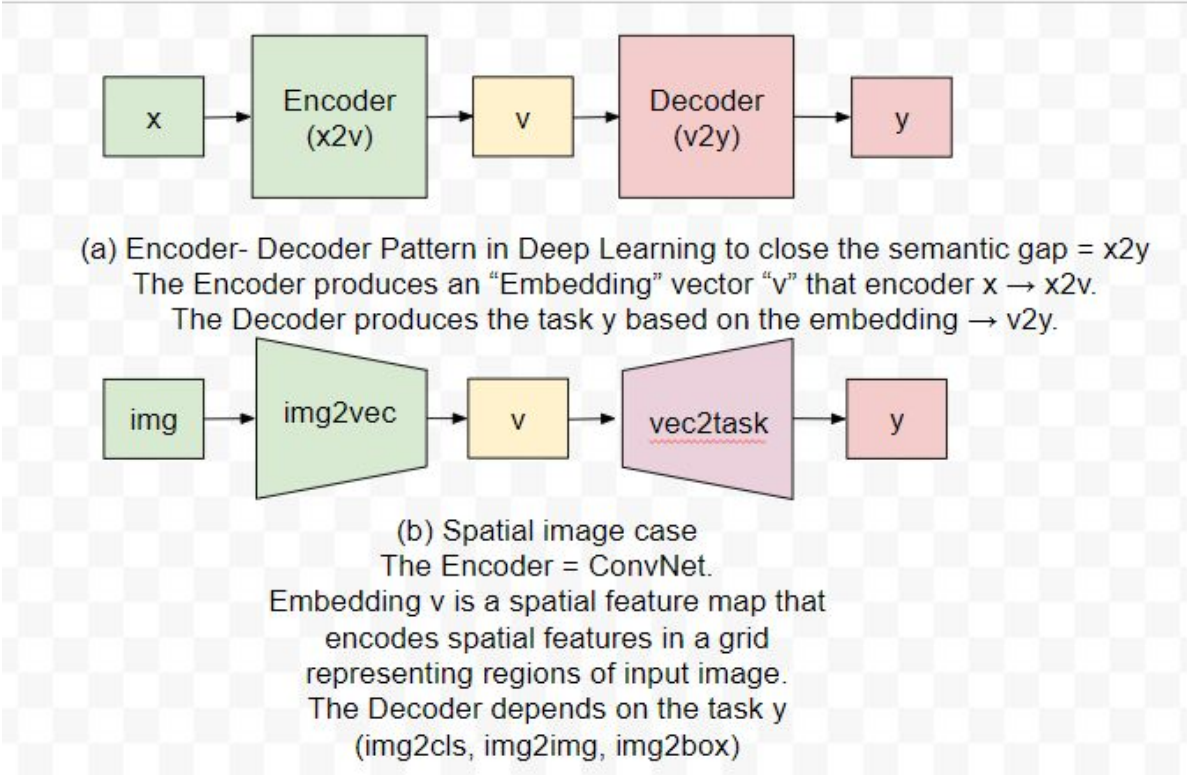
The aim is to close the semantic-gap: x2y

Encoder encodes context of the input: x2vec

The encoder is designed based on the input.

Decoder produces the output: vec2y

The design of the decoder is task dependent.

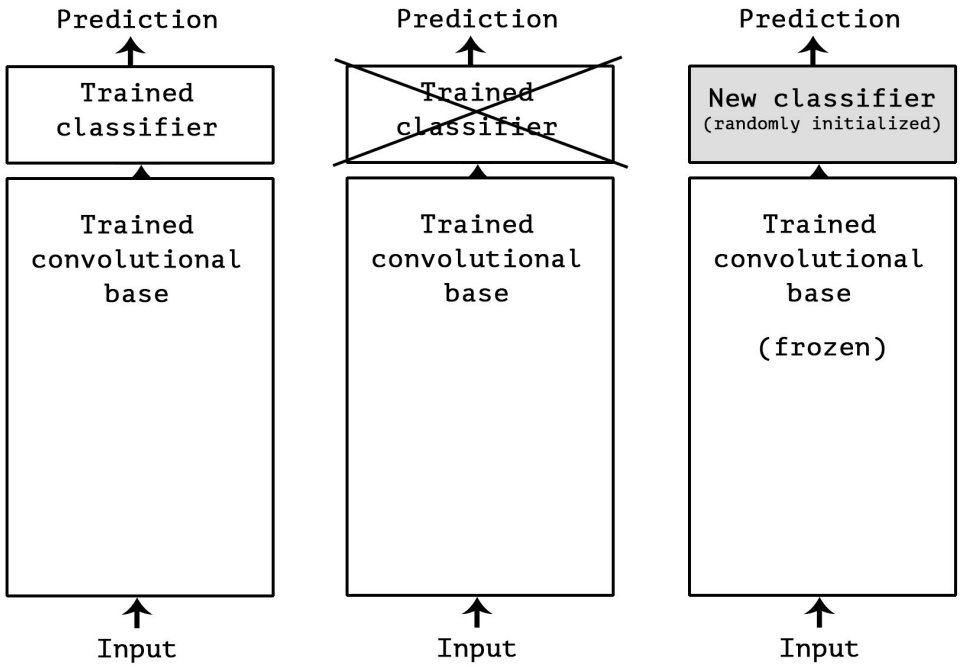


NLP ImageNet time - 2018

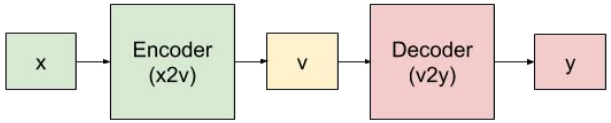
“Jeremy Howard”

Encoder - Decoder Pattern

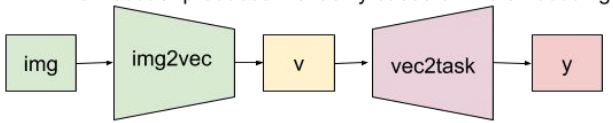
In CV → Pre-train encoder on ImageNet → Classification is the excellent task + ImageNet is abundan



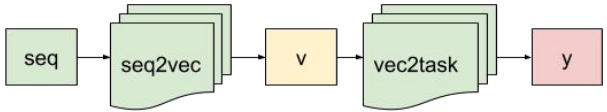
Encoder-Decoder pattern in NLP



(a) Encoder- Decoder Pattern in Deep Learning to close the semantic gap = x2y
The Encoder produces an “Embedding” vector “v” that encoder x → x2v.
The Decoder produces the task y based on the embedding → v2y.

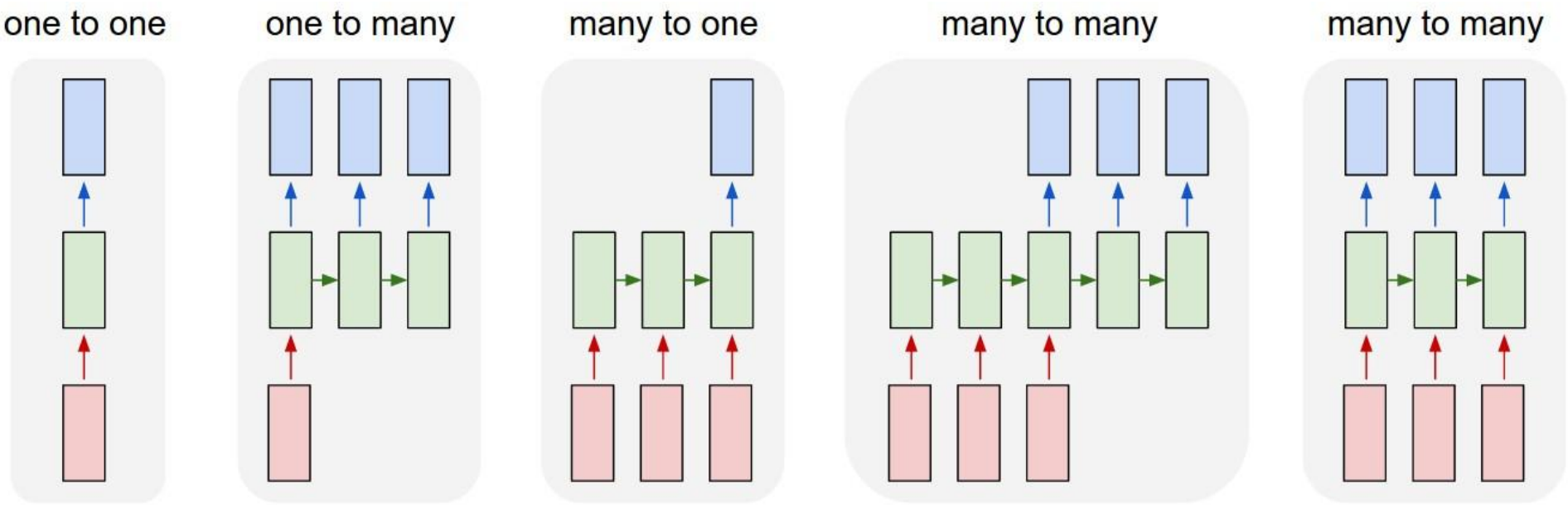


(b) Spatial image case
The Encoder = ConvNet.
Embedding v is a spatial feature map that encodes spatial features in a grid representing regions of input image.
The Decoder depends on the task y (img2cls, img2img, img2box)



(c) Sequential language case
The Encoder = Sequence model (RNN, Transformer,..etc)
Embedding v is a summarization of the sequence.
The Decoder depends on the task y (seq2cls, seq2seq)

Encoder-Decoder pattern in NLP



Examples

- One-One: Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
- One-many: Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
- Many-one: Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
- Many-Many: Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
- Synced sequence input and output: NER

What is the best representation of language?

This question summarizes all NLP efforts!

For what?

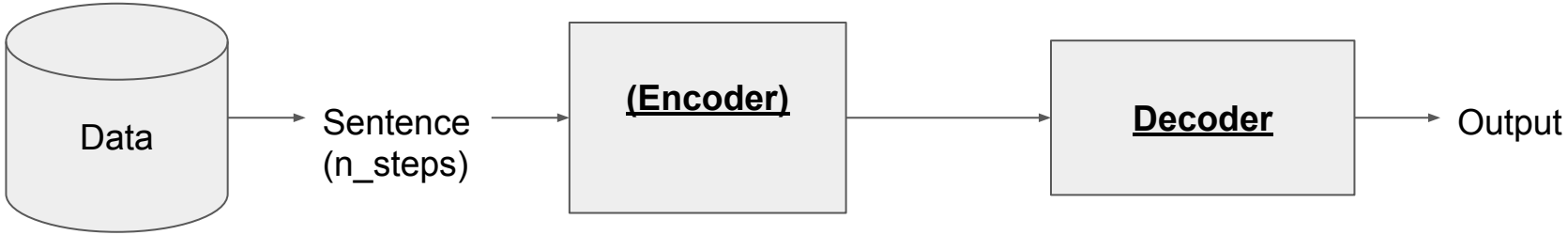
- Classification
  - Many-to-one: Seq2Class
  - Sentiment analysis, Toxicity detection (JIGSAW), Real or not? Disaster tweets
- Dialogue:
  - Many-to-many: Seq2Seq → Unaligned case → More on that later
  - MT, Spelling correction, Speech, OCR,...etc

NLP models meta-architectures

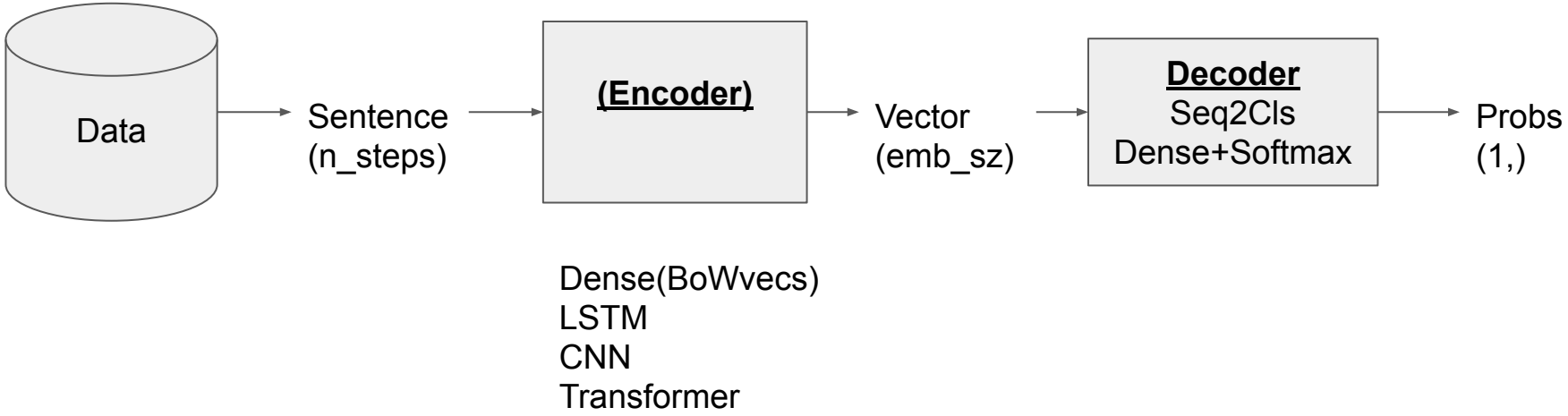
Just like in CV: **Encoder-Decoder**

- **Seq2Class:**
  - Encoder = words vectors aggregation **(How?)**
  - Decoder = None (just classifier=softmax)
  - Analogy to CV: Encoder-Softmax (AlexNet, VGG,...etc)
- **Seq2Seq:**
  - Encoder = words vectors aggregation **(How?)**
  - Decoder = multiple words generation **(How?)**
  - Analogy to CV: Encoder-Decoder in semantic segmentation. But in SS, we have aligned many2many, while in NLP, we have unaligned sequences→ challenge in annotation, model, when to stop, position encoding...etc
- **Word vectors are the input to all the above meta-architectures:**
  - Unlike in CV, where pixels are already digitized
  - Also, in NLP **order matters! = Context**

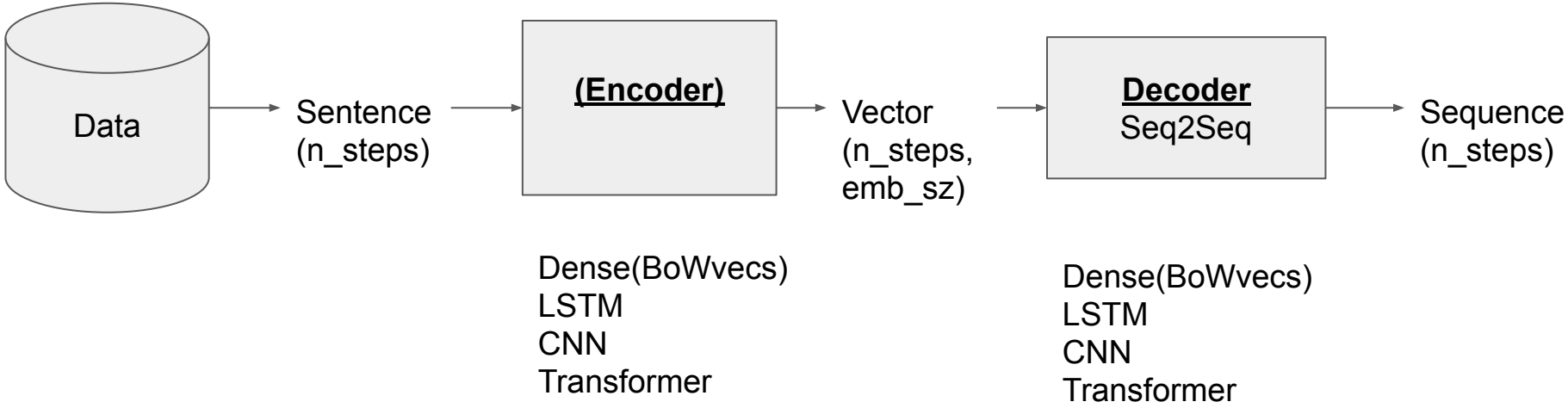
Encoder-Decoder meta-architecture



Encoder-Decoder meta-architecture

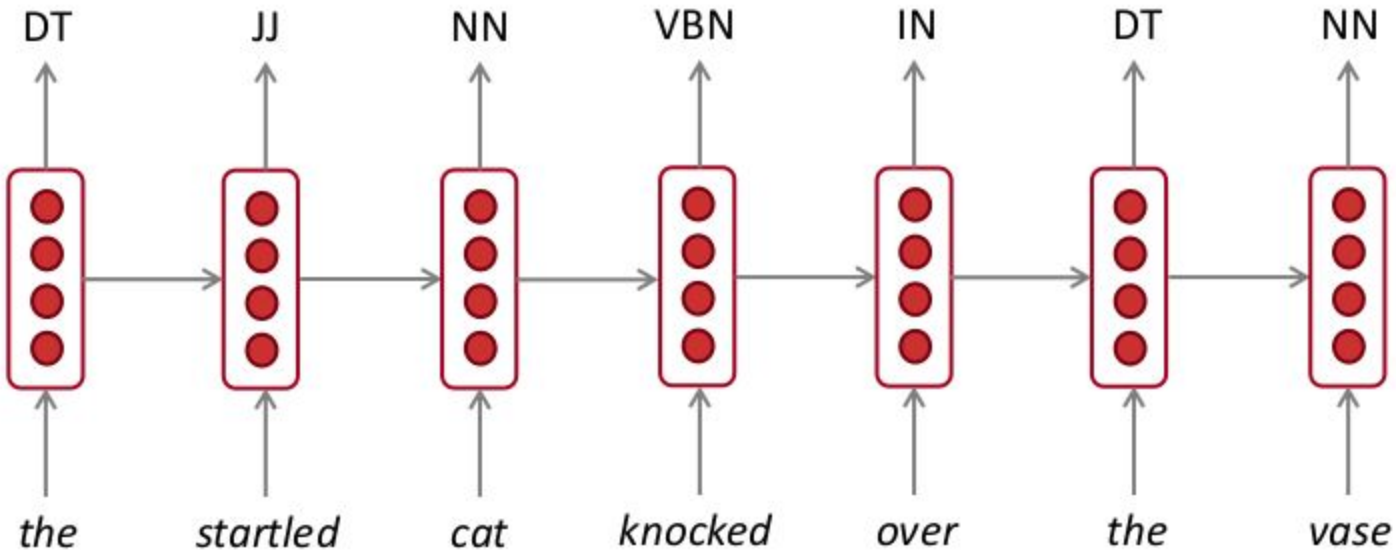


Encoder-Decoder meta-architecture



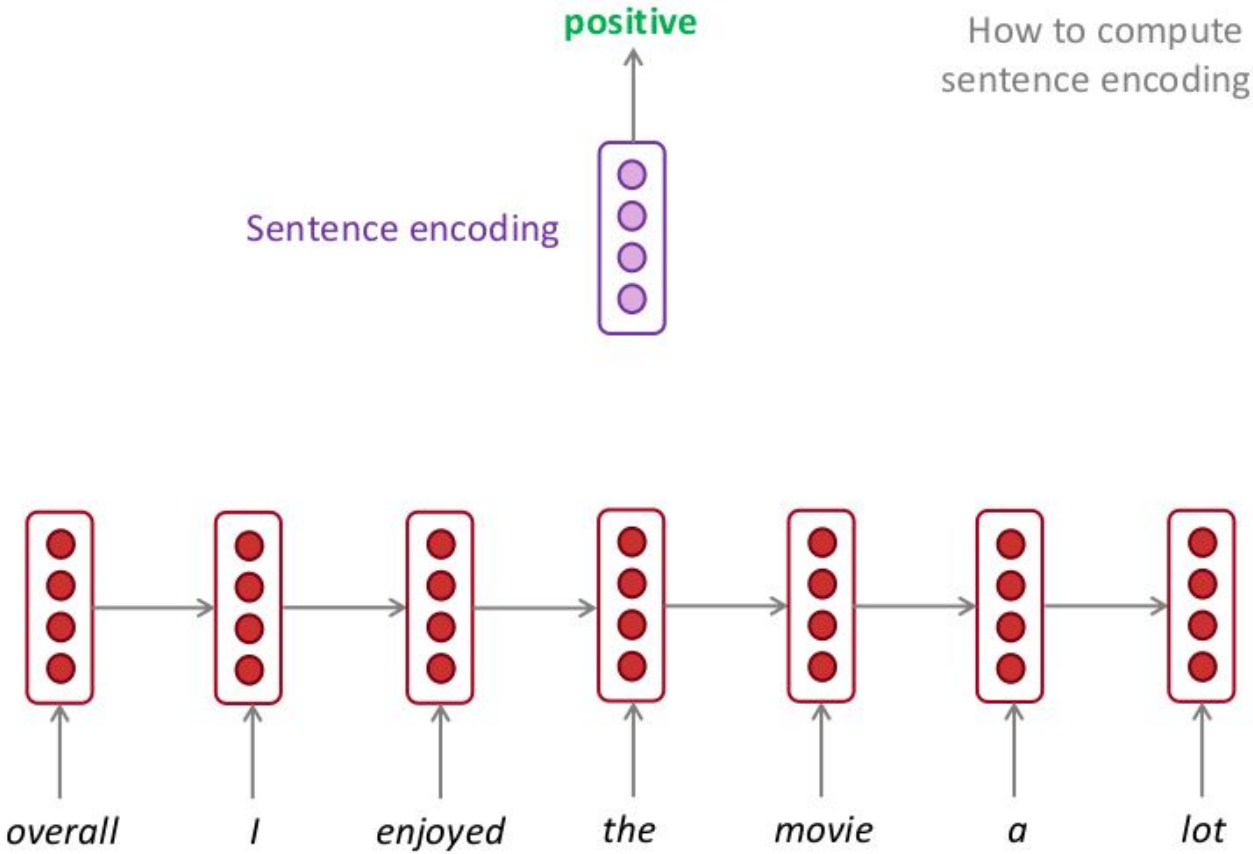
RNNs can be used for tagging

e.g. part-of-speech tagging, named entity recognition



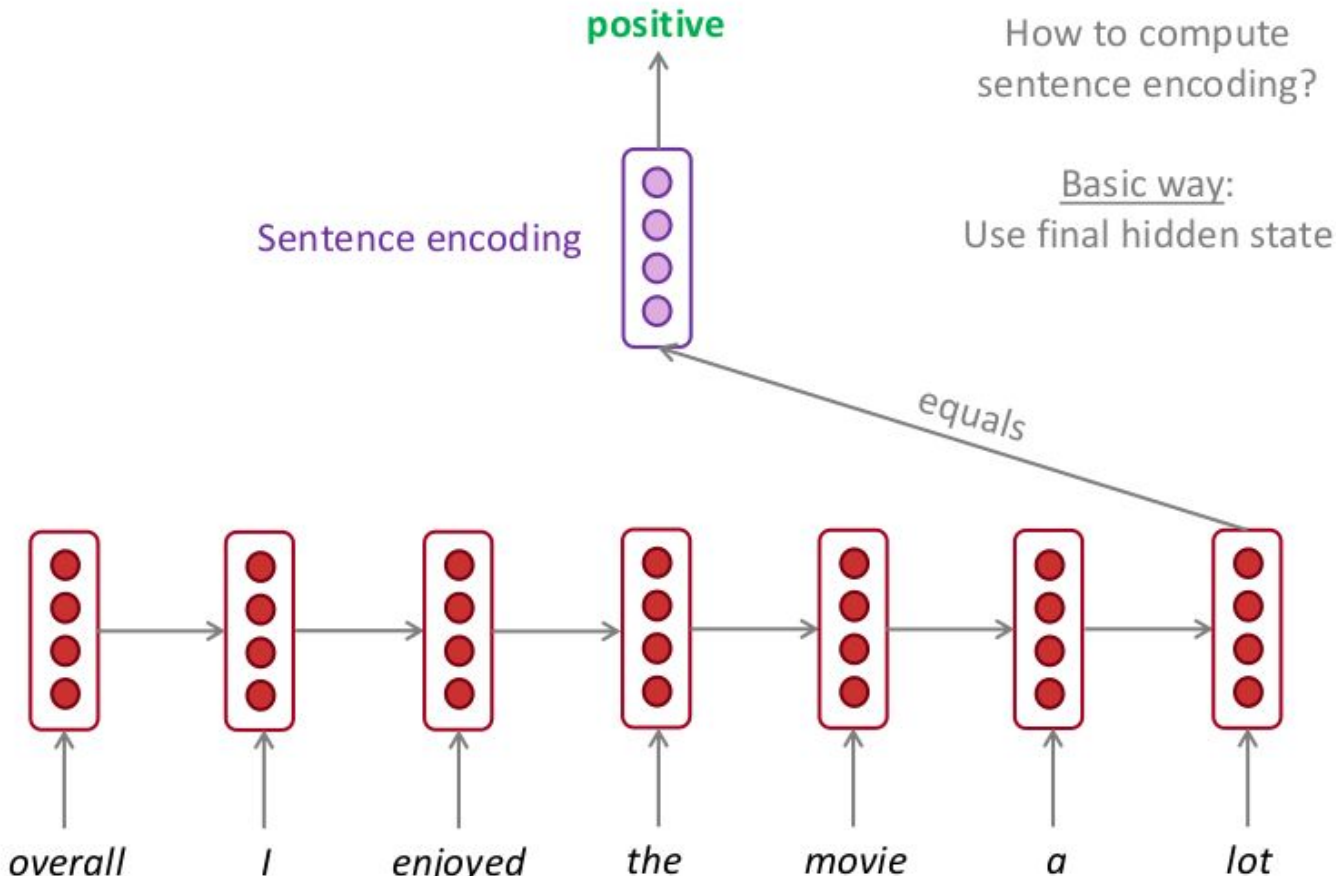
RNNs can be used for sentence classification

e.g. sentiment classification



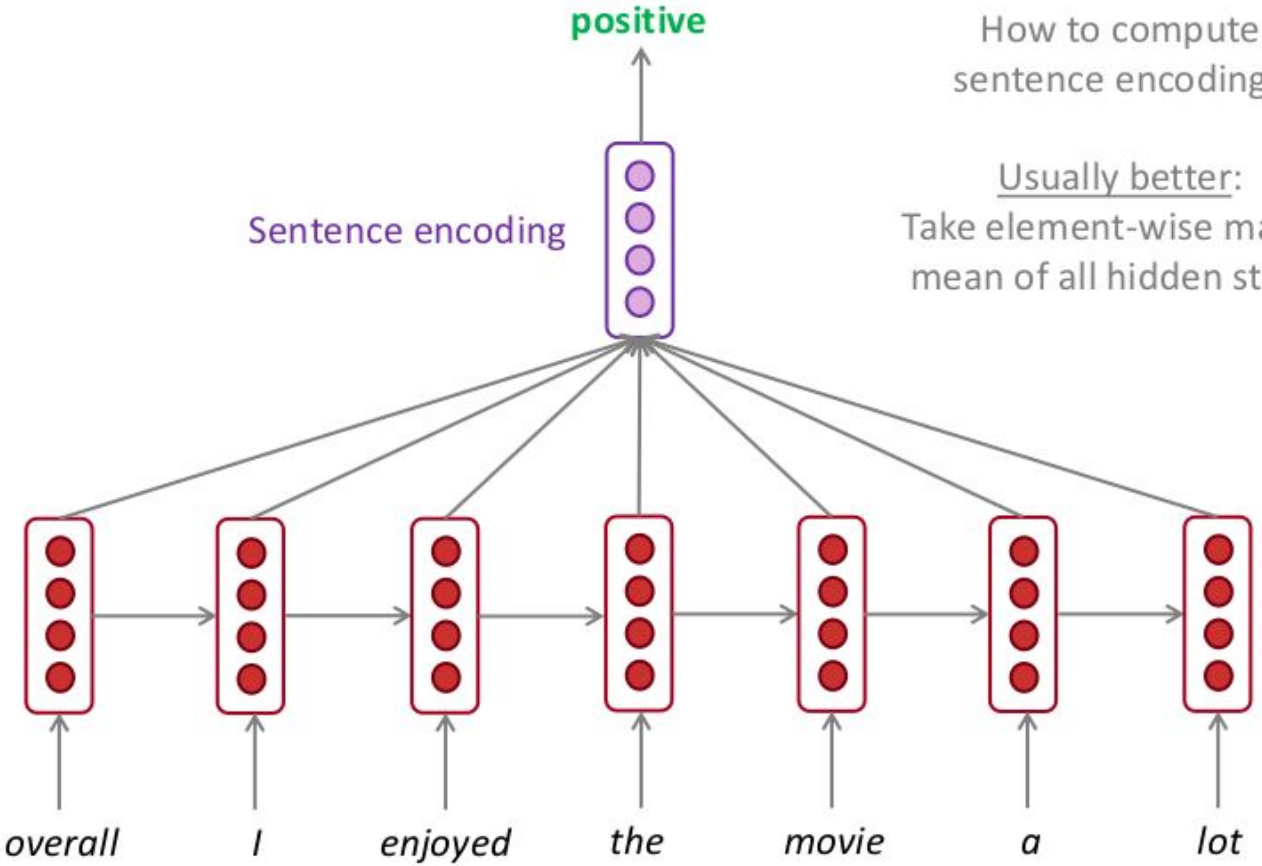
RNNs can be used for sentence classification

e.g. sentiment classification



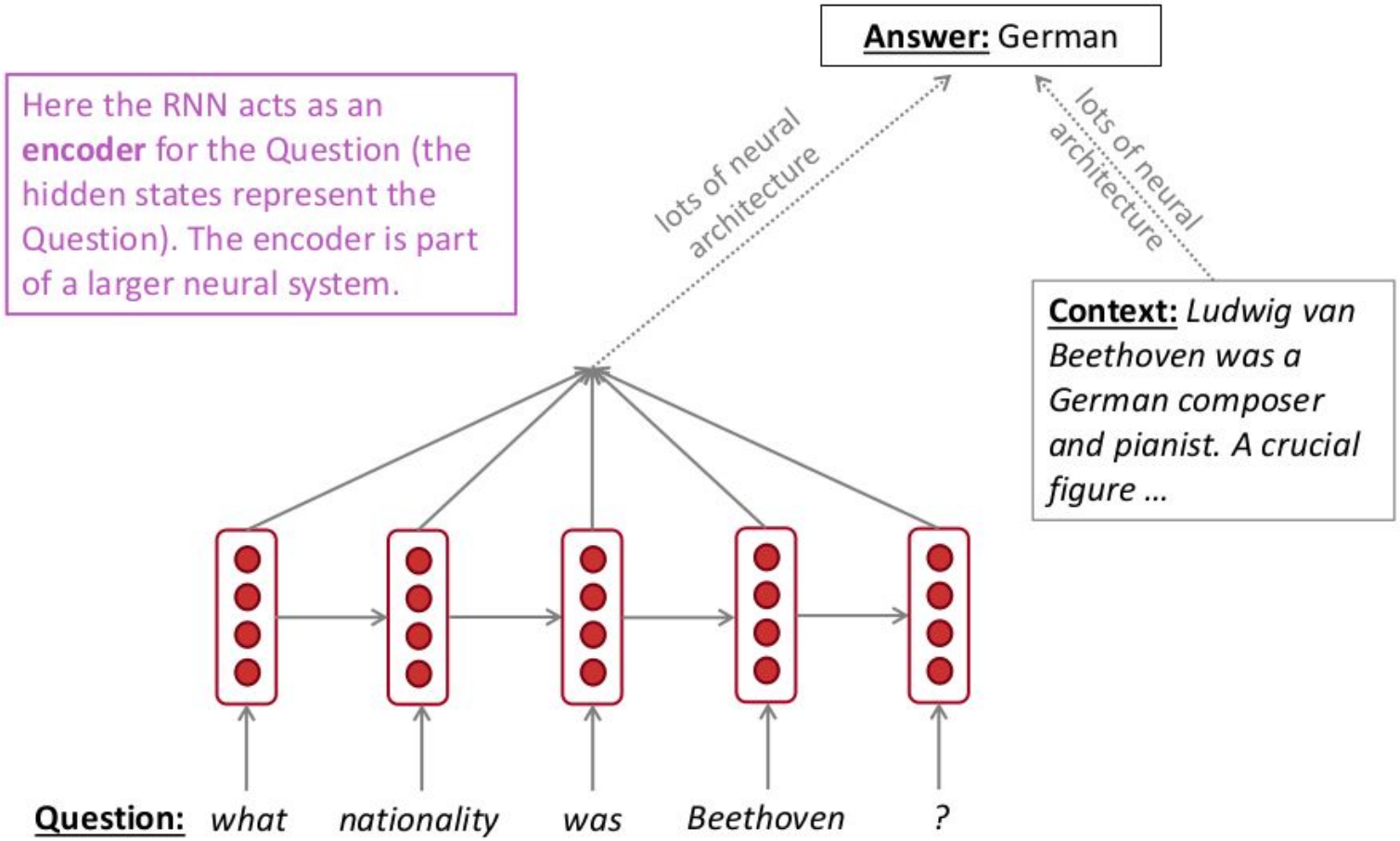
RNNs can be used for sentence classification

e.g. sentiment classification



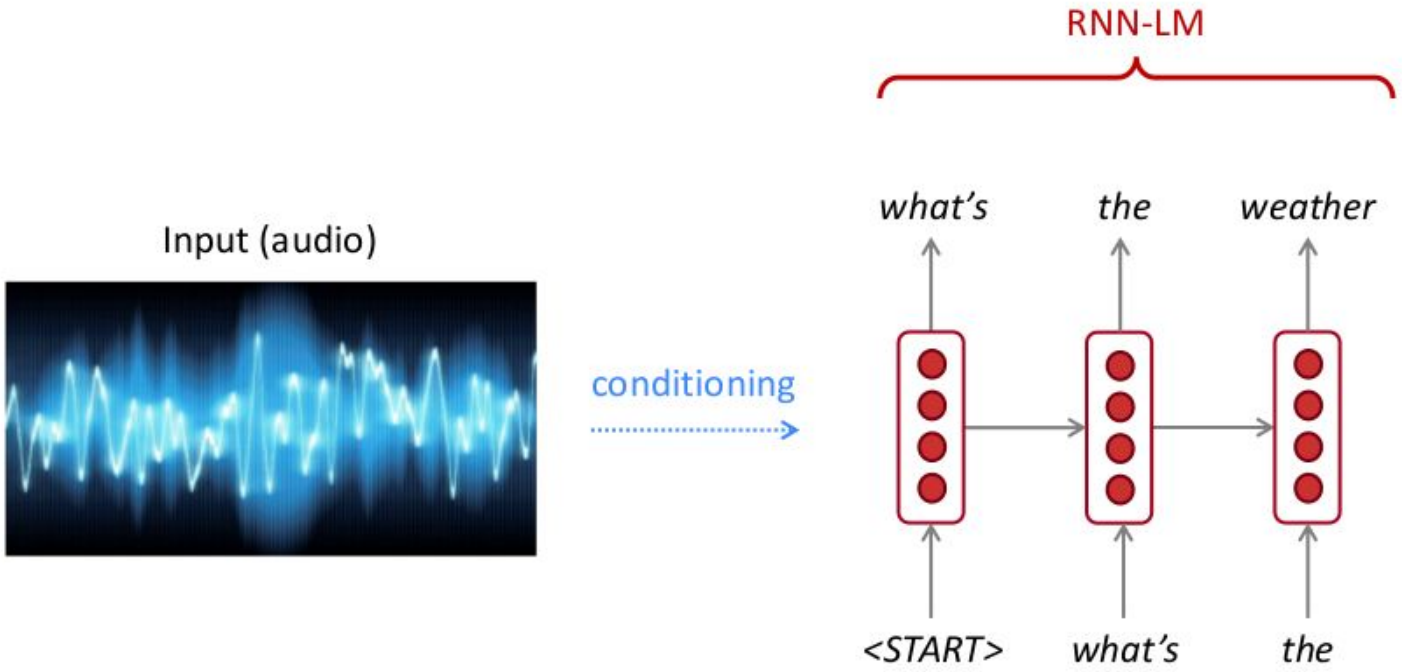
RNNs can be used as an encoder module

e.g. question answering, machine translation, *many other tasks!*



RNN-LMs can be used to generate text

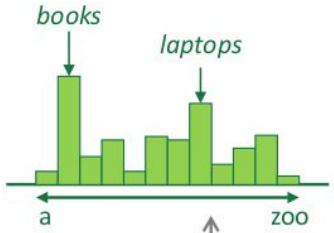
e.g. speech recognition, machine translation, summarization



This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail later.

A RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

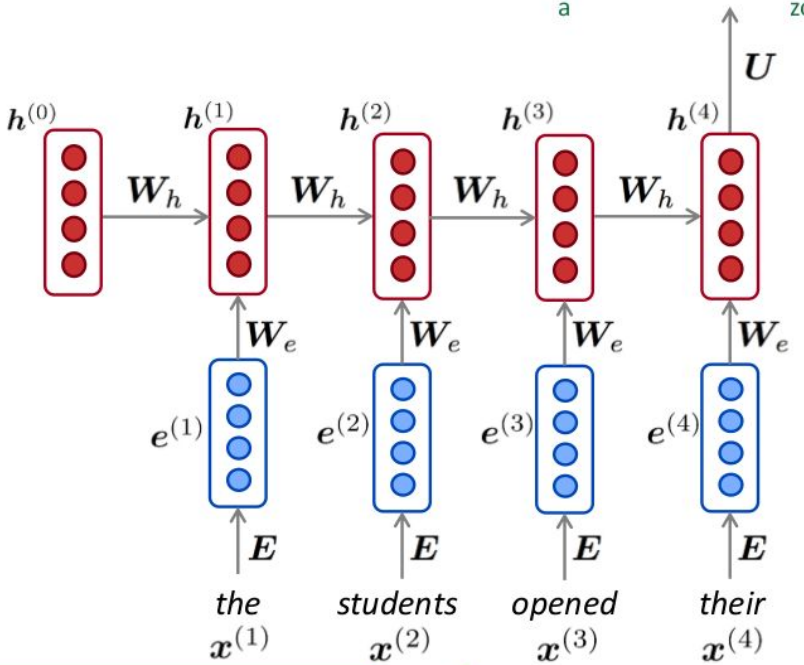
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



***Note**: this input sequence could be much*

ULMFiT

Universal Language Model Fine-tuning for Text Classification

LM is an excellent task for NLP!

Central to ALL NLP tasks → if you have a model capable of predicting the next word, certainly it will be a good starting point to other tasks

Self-supervised → LM requires no annotation, labels are in the data→ Can be trained on the whole wikipedia at no annotation cost

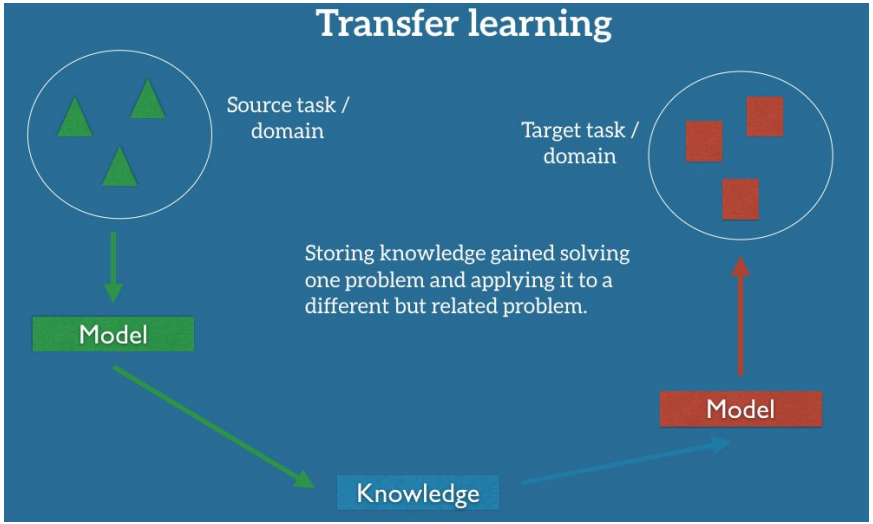
NLP ImageNet time - 2018

“Jeremy Howard”

Encoder - Decoder Pattern

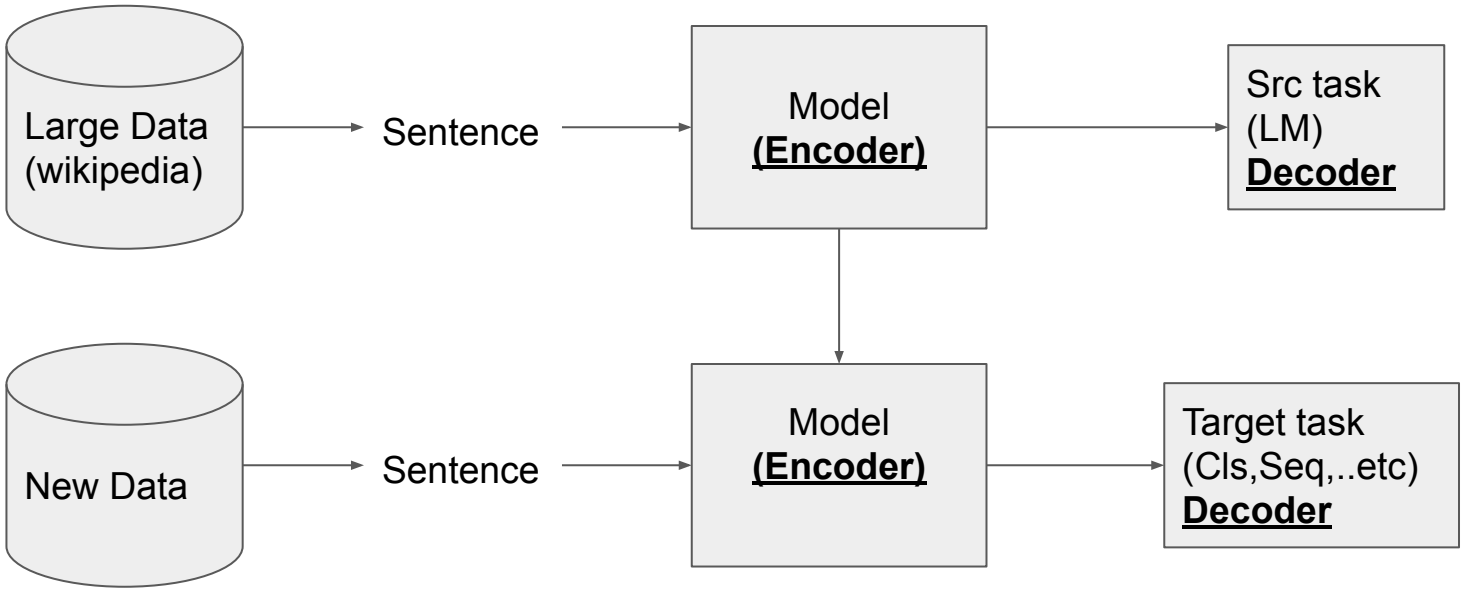
Pre-train the Encoder on src task → TL to target task

Src task usually → LM

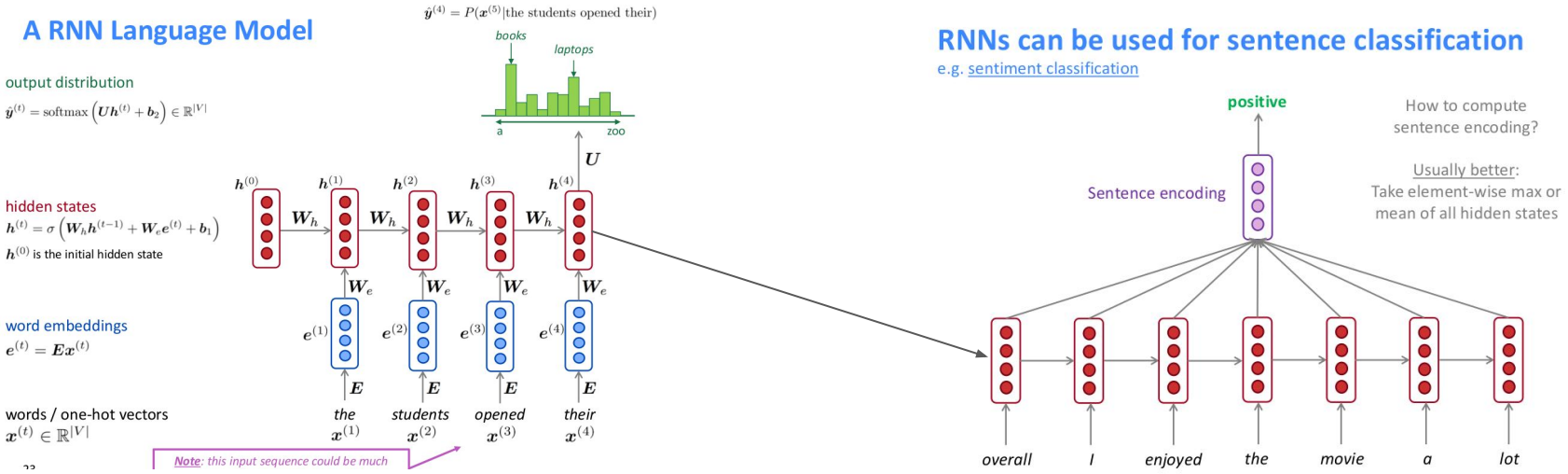


Sentence Embedding

But why we care about individual words vectors, while we can already model the whole sequence/sentence?



TL from LM


A RNN Language Model
output distribution
$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$
hidden states
$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$
$\boldsymbol{h}^{(0)}$ is the initial hidden state
word embeddings
$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$
words / one-hot vectors
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$
$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$
books
laptops
a
zoo
$\boldsymbol{U}$
$\boldsymbol{W}_h$
$\boldsymbol{W}_e$
$\boldsymbol{E}$
the
students
opened
their
Note: this input sequence could be much
RNNs can be used for sentence classification
e.g. sentiment classification
positive
Sentence encoding
How to compute sentence encoding?
Usually better:
Take element-wise max or mean of all hidden states
overall
I
enjoyed
the
movie
a
lot

BERT

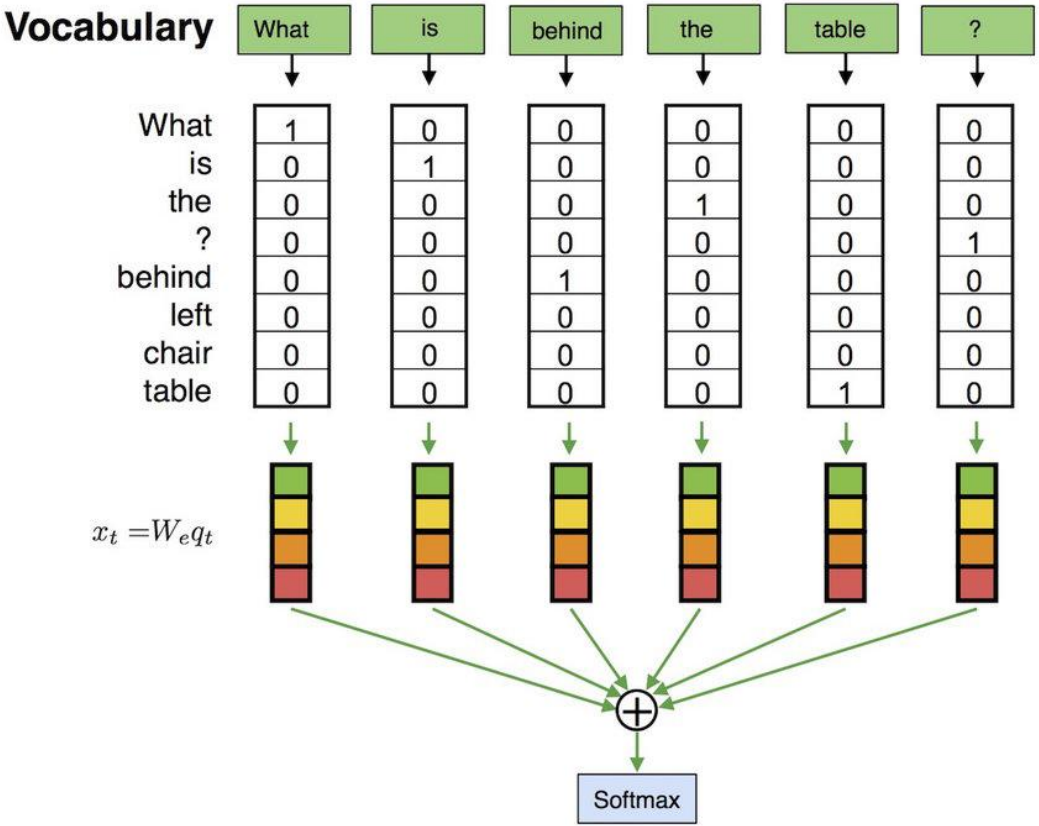
How to represent a sentence?

- BoW
- RNN
- CNN

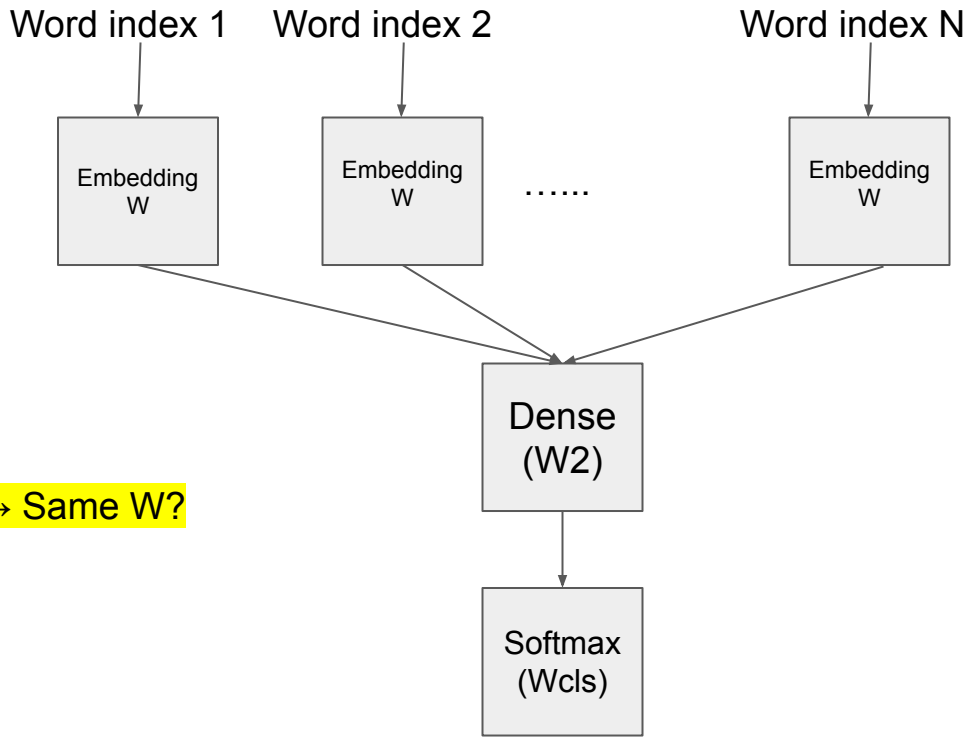
BoW vectors

How to represent words to NN?

Bag-of-Words model

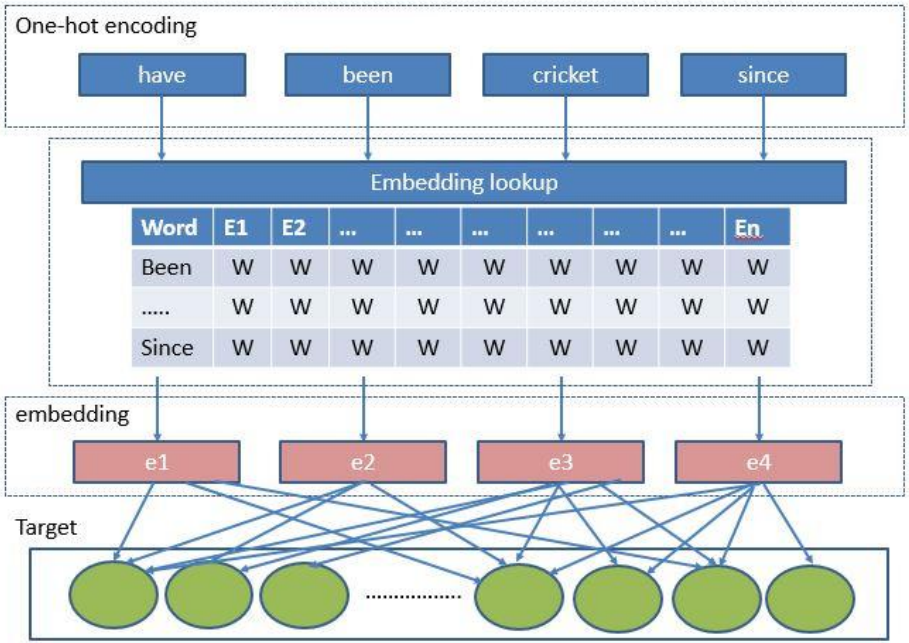


Word Embeddings



ALL Embedding layers → Same W?

BoW vectors model



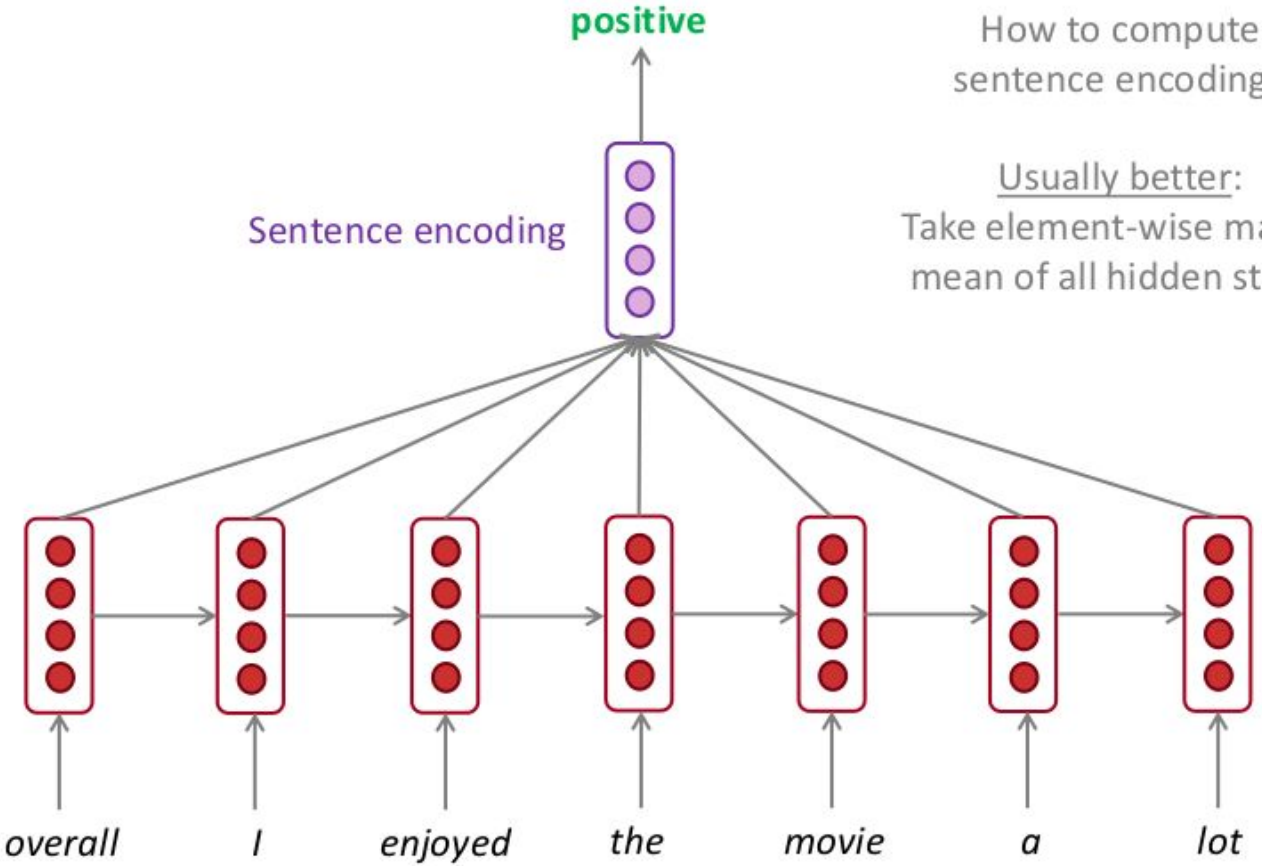
What’s wrong with BoW?

Soup of words vectors → Sequence info is lost!

RNN

RNNs can be used for sentence classification

e.g. sentiment classification



What's wrong with RNN?

It's sequential!

- We cannot make use of parallelism of modern computer architectures or GPU!

What's good about it?

- It captures sequence information and summarizes it in a **state**
- Use that state for (condition output on that state):
  - Classification
  - Sequence generation

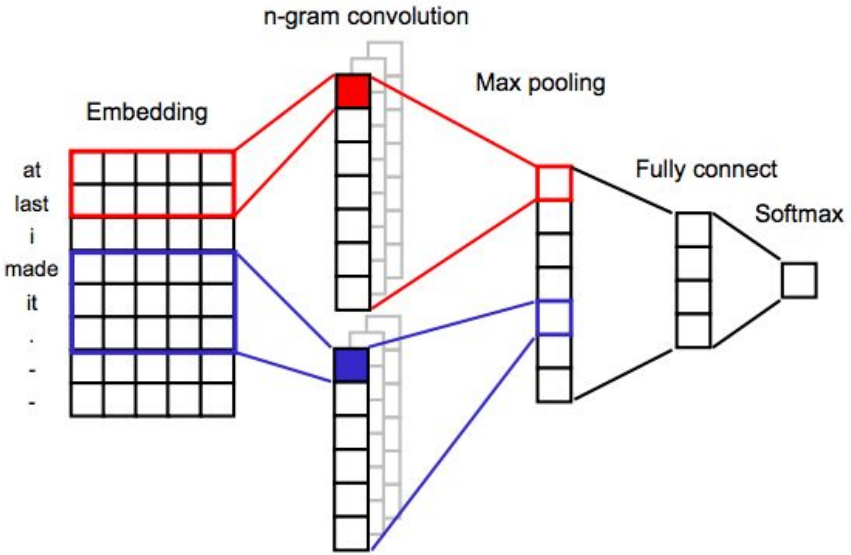
State = Feature of the input

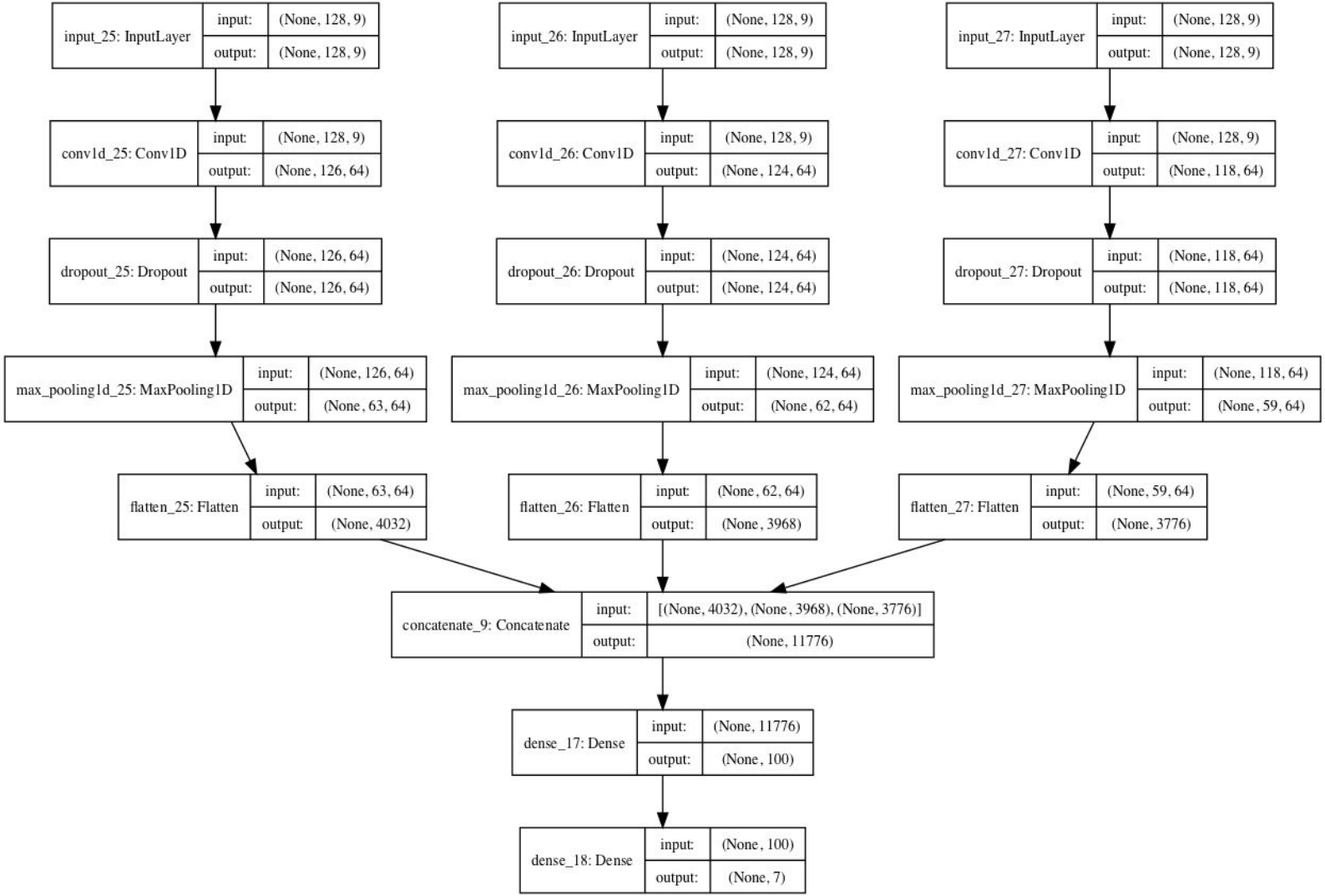
CNN

Multi-headed CNN

Another popular approach with CNNs is to have a multi-headed model, where each head of the model reads the input time steps using a different sized kernel.

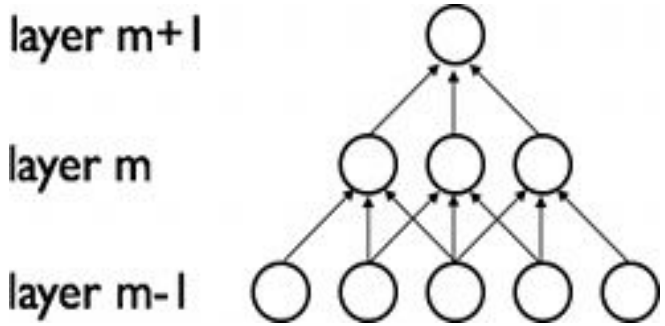
For example, a three-headed model may have three different kernel sizes of 3, 5, 11, allowing the model to read and interpret the sequence data at three different resolutions. The interpretations from all three heads are then concatenated within the model and interpreted by a fully-connected layer before a prediction is made.

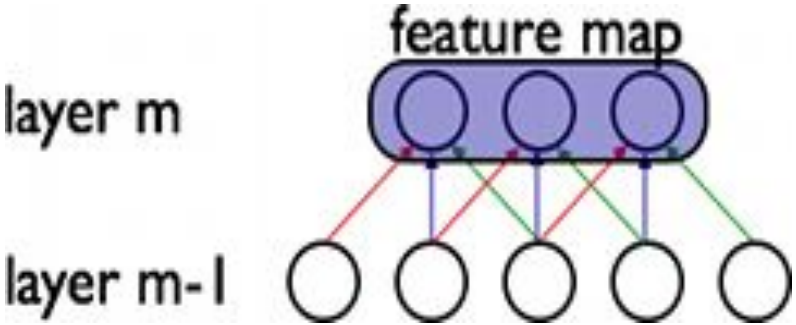



input_25: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
input_26: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
input_27: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
conv1d_25: Conv1D
input: (None, 128, 9)
output: (None, 126, 64)
conv1d_26: Conv1D
input: (None, 128, 9)
output: (None, 124, 64)
conv1d_27: Conv1D
input: (None, 128, 9)
output: (None, 118, 64)
dropout_25: Dropout
input: (None, 126, 64)
output: (None, 126, 64)
dropout_26: Dropout
input: (None, 124, 64)
output: (None, 124, 64)
dropout_27: Dropout
input: (None, 118, 64)
output: (None, 118, 64)
max_pooling1d_25: MaxPooling1D
input: (None, 126, 64)
output: (None, 63, 64)
max_pooling1d_26: MaxPooling1D
input: (None, 124, 64)
output: (None, 62, 64)
max_pooling1d_27: MaxPooling1D
input: (None, 118, 64)
output: (None, 59, 64)
flatten_25: Flatten
input: (None, 63, 64)
output: (None, 4032)
flatten_26: Flatten
input: (None, 62, 64)
output: (None, 3968)
flatten_27: Flatten
input: (None, 59, 64)
output: (None, 3776)
concatenate_9: Concatenate
input: [(None, 4032), (None, 3968), (None, 3776)]
output: (None, 11776)
dense_17: Dense
input: (None, 11776)
output: (None, 100)
dense_18: Dense
input: (None, 100)
output: (None, 7)

Disadv: Depth scales with seq length!

- ConvNet captures sequence information and summarizes it in a **state**
- Use that state for (condition output on that state):
  - Classification
  - Sequence generation
- Network depth scales with max seq (sentence) length!
  - sequence length requires deeper models, it makes it difficult to learn dependencies between distant words
- Inefficient for variable length! Needs padding





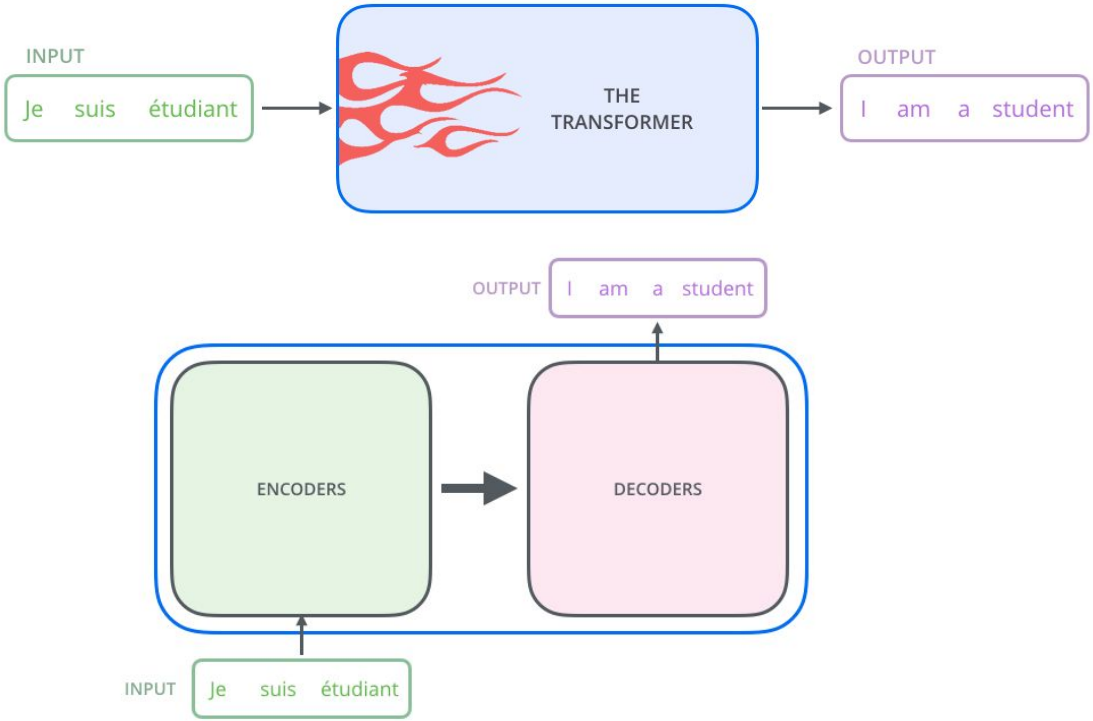
Speed + Performance!

- BoW → Fast → No sequence
- RNN → For sequence! → Slow
- Conv/Dense → For speed! → No sequence + Conv bad scaling with seq len

Can we take the best out of ALL?

Transformers

Transformers



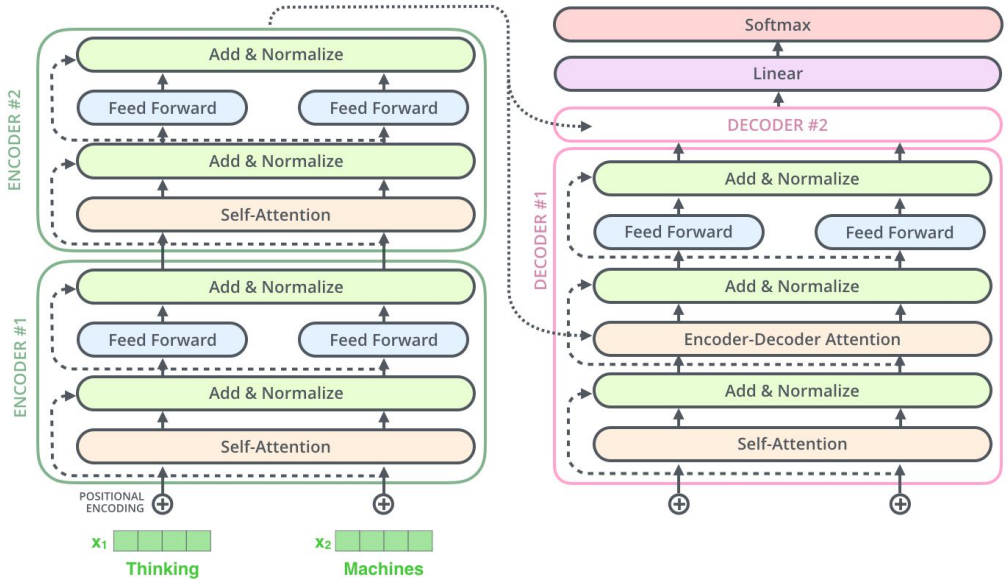
Building blocks step-by-step

Encoder:

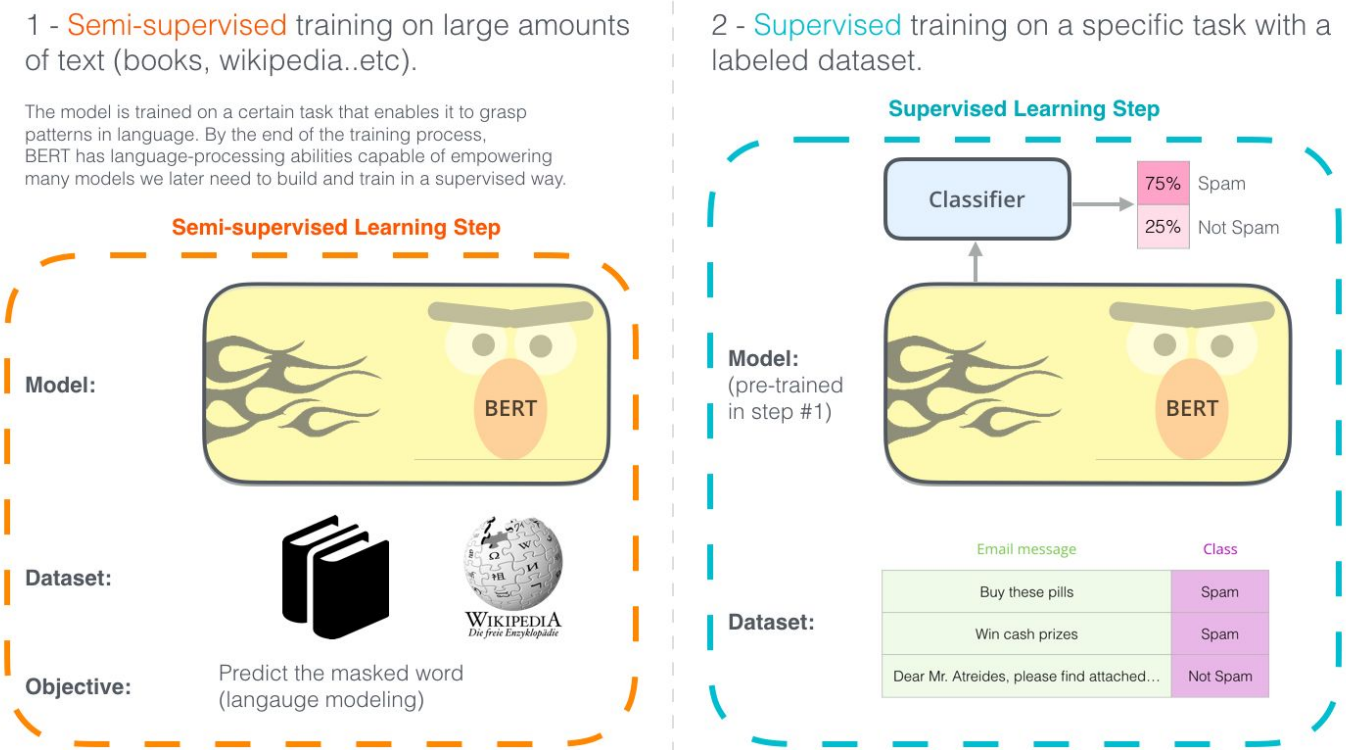
- **Self attention**
- Skip
- Feed fwd
- **Position Encoding**

Decoder:

- Encoder-Decoder attention (masking)
- Skip
- Feed fwd
- Position Encoding
- Decoder feedback (teacher forcing)
- Softmax output

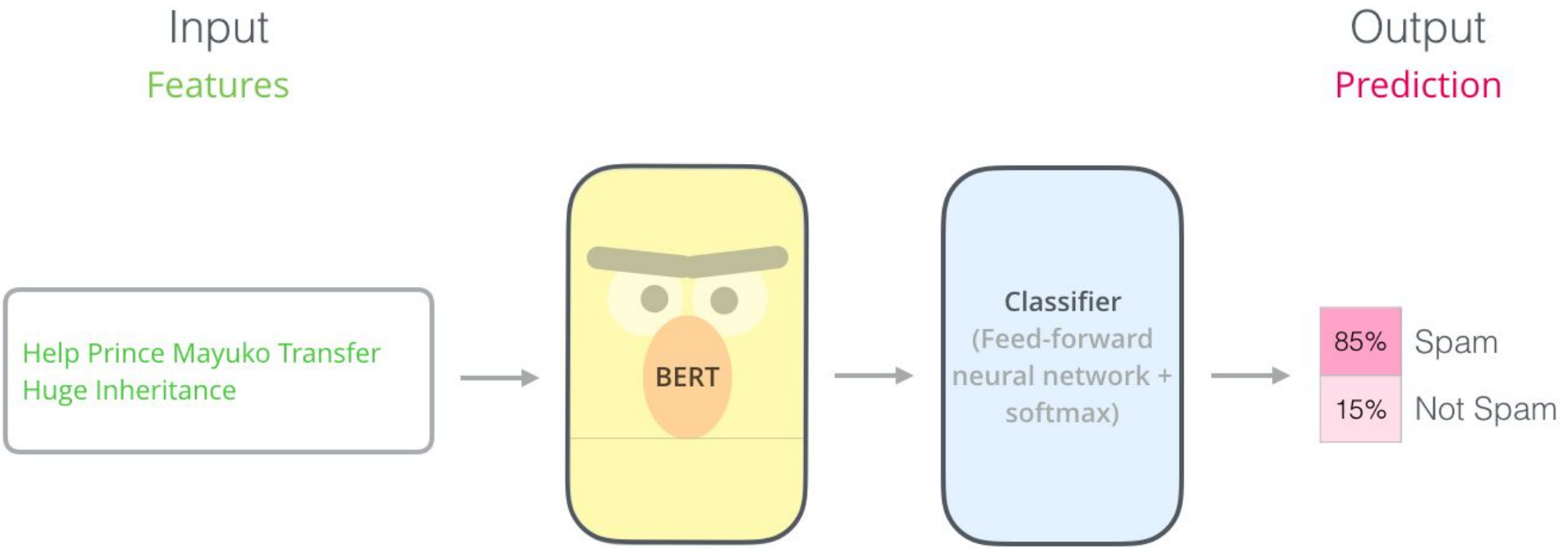


BERT



http://jalammar.github.io/images/bert-transfer-learning.png

BERT fine tuning



OpenAI GPT

GPT1- GPT2 - GPT3

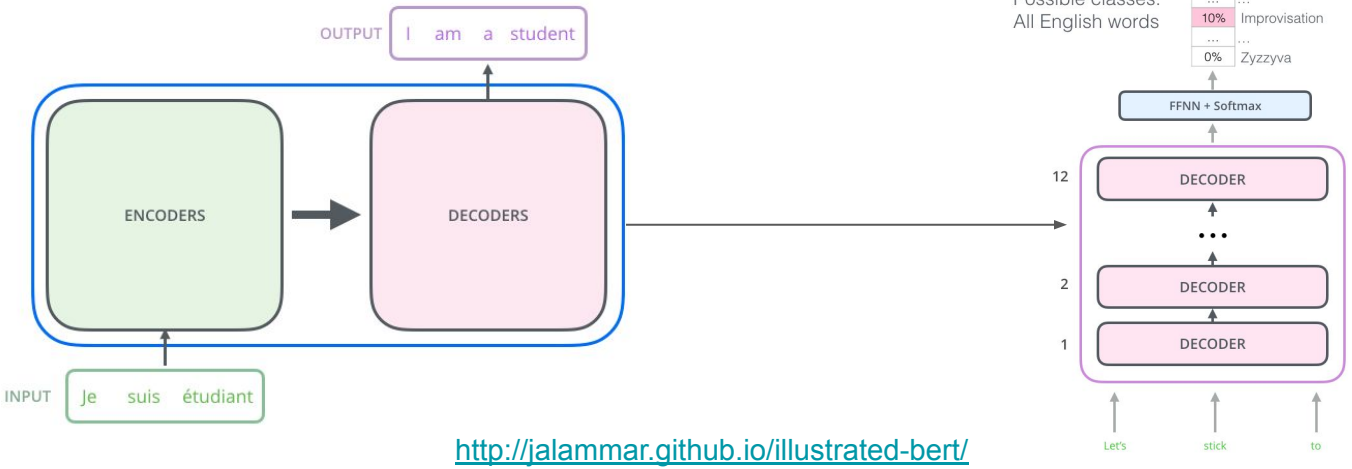
GPT 1 = ULMFiT + Transformer

Generative Pretrained Transformer

Instead of transferring the encoder → transfer the decoder!

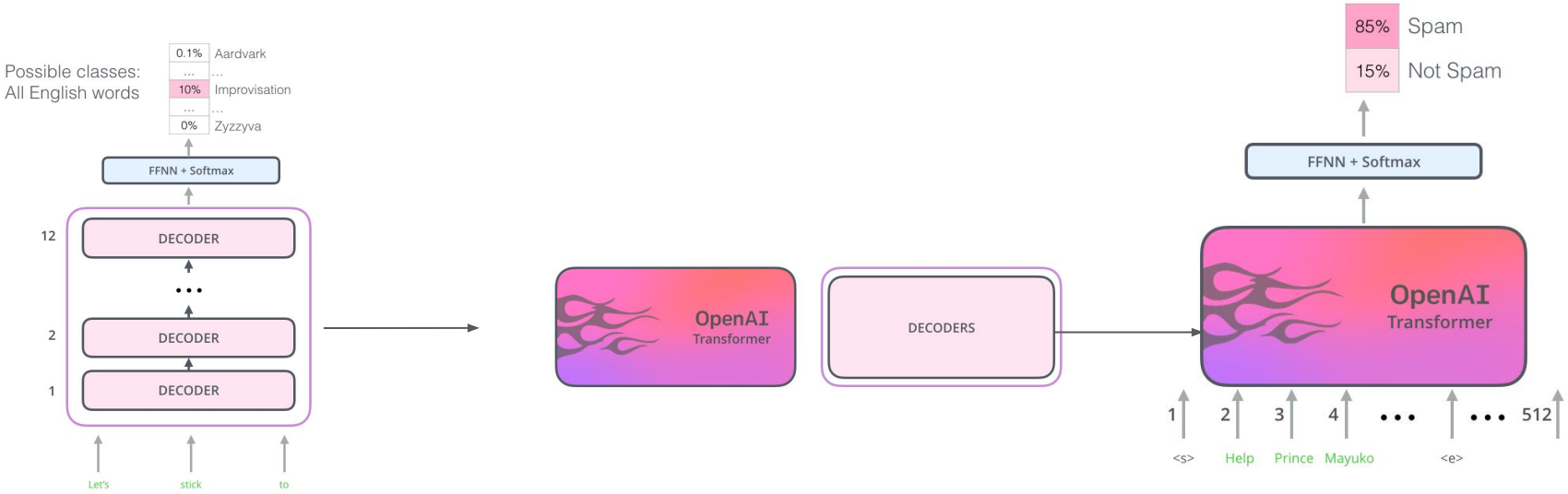
Recall that, in transformer, the encoder had no feedback from it's output → no decoding.

But if we are to do LM, we will need this feedback, so that the next word is conditioned on what we generated so far.



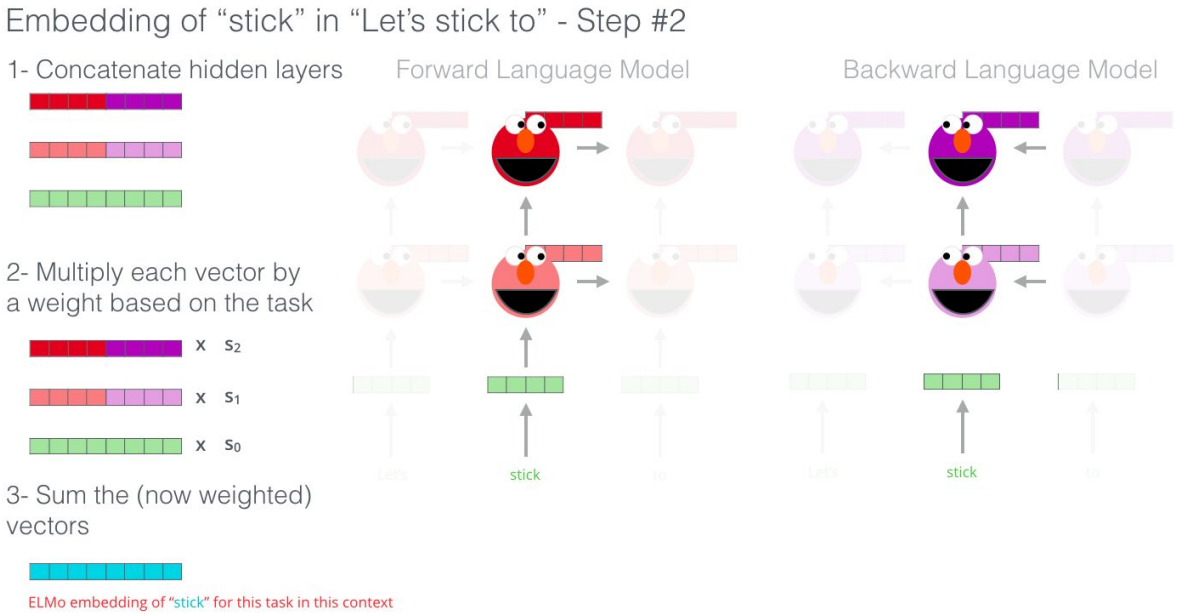
http://jalammar.github.io/illustrated-bert/

GPT 1 TL to downstream tasks



BERT: Back to encoders again

LM are better bi-directional → Like ELMo

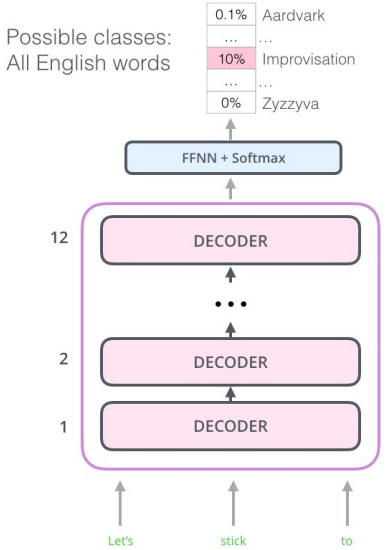


http://jalammar.github.io/images/elmo-embedding.png

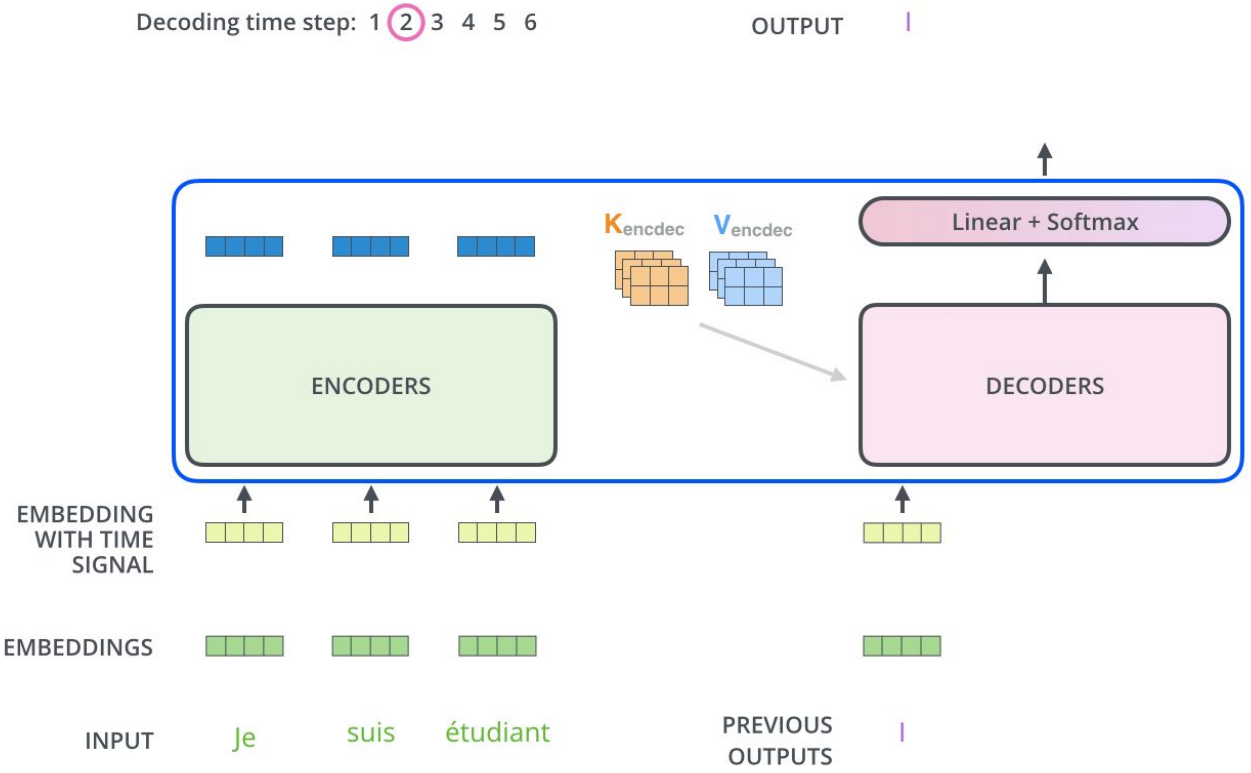
BERT: Back to encoders again

LM are better bi-directional → Like ELMo

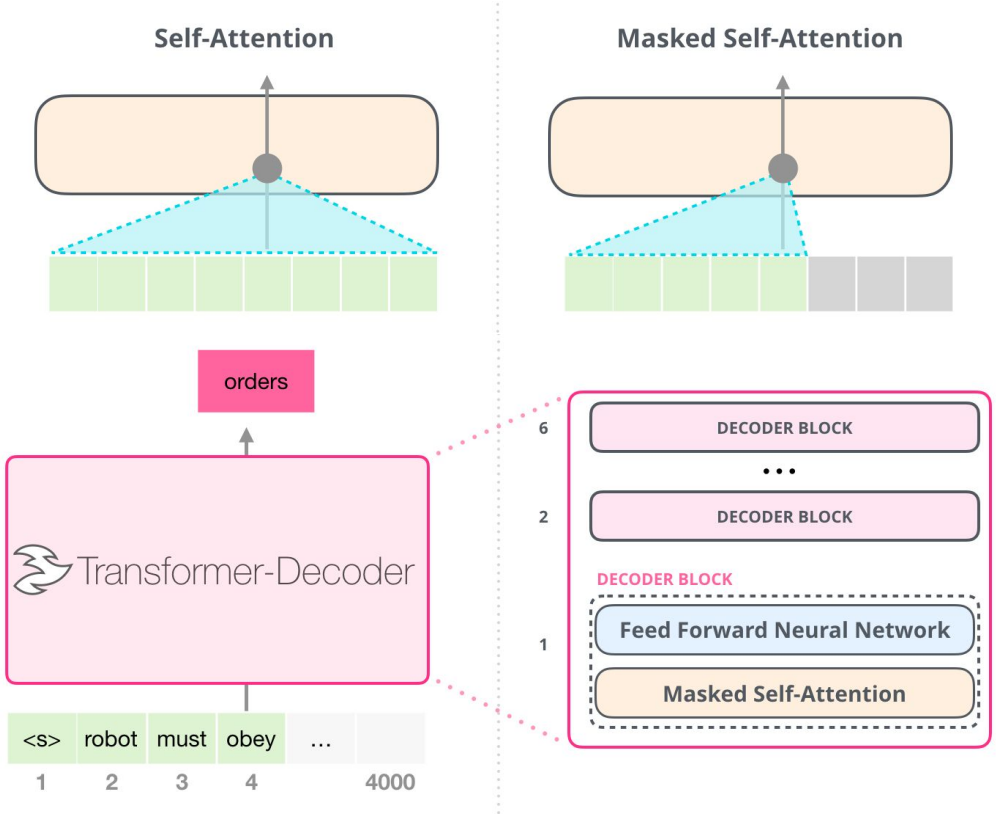
But decoders in GPT are only capable of forward decoding: p(xt+1|xt). → **AutoRegressive LM**



AR LM


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K_{encdec}
V_{encdec}
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Transformer decoders mask the future



Why not left-right + right-left?

What does “bidirectional context” mean?

For some words, their meaning might only become apparent when you look at both the left and right context **simultaneously**.

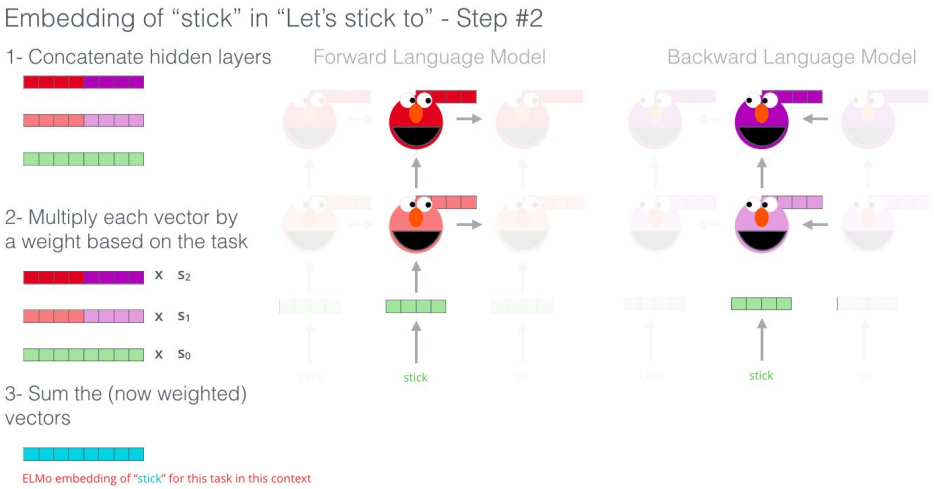
The simultaneous part is important: models like ELMo train two separate models that each take the left and right context into account but do not train a model that uses both at the same time.

Why not left-right + right-left?

What does “bidirectional context” mean?

For some words, their meaning might only become apparent when you look at both the left and right context **simultaneously**.

The simultaneous part is important: models like ELMo train two separate models that each take the left and right context into account but do not train a model that uses both at the same time.



Here “sum” is just a hack to merge both “Separate” contexts. We need some model to consider both contexts simultaneously!

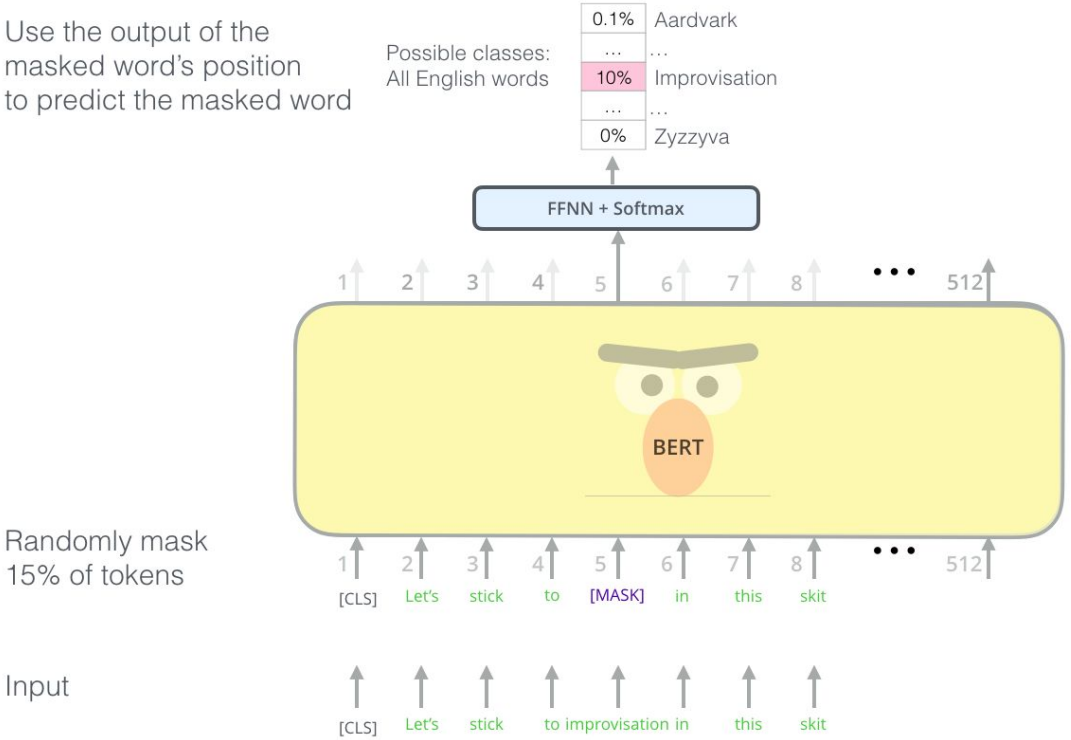
AE LM: Masked LM (MLM)

An idea very similar to CBOW. Even sometimes, the [MASK] is just an invalid word, and ask the model to recover the correct one, somehow similar to SDNS in word2vec.

But in this case, the whole sentence vector is learned

MLM ⇒ AutoEncoding LM

autoencoding based pretraining aims to reconstruct original data from corrupted data



http://jalammar.github.io/illustrated-bert/

Two-sentences task (Next-sentence prediction)

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

In some of the downstream tasks, we need QA setup:

Is this sentence an answer to that question (multiple choice) → sent A <> sent B

We could find some questions in the corpus (like wikipedia) and associate the question and answer texts to each others.

Also, we can make “IsNext” task → similar to SGNS in word2vec

Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`
Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`
Label = `NotNext`

http://jalammar.github.io/images/bert-next-sentence-prediction.png

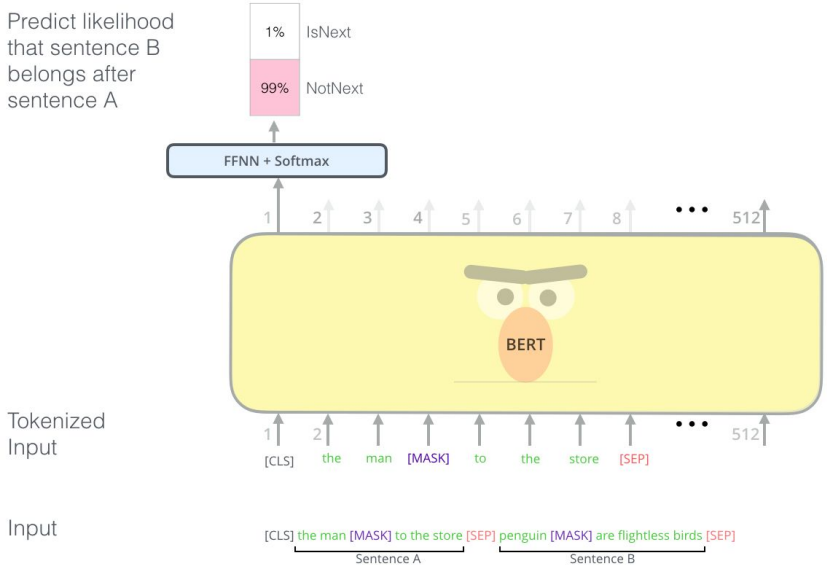
Two-sentences task (Next-sentence prediction)

In some of the downstream tasks, we need QA setup:

Is this sentence an answer to that question (multiple choice) → sent A <> sent B

We could find some questions in the corpus (like wikipedia) and associate the question and answer texts to each others.

Also, we can make “IsNext” task → similar to SGNS in word2vec



Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

http://jalammar.github.io/images/bert-next-sentence-prediction.png

BERT =

ULMFiT → LM TL

+ ELMo → Bi-directional

+ GPT 1 → Transformer

+ Next sentence prediction

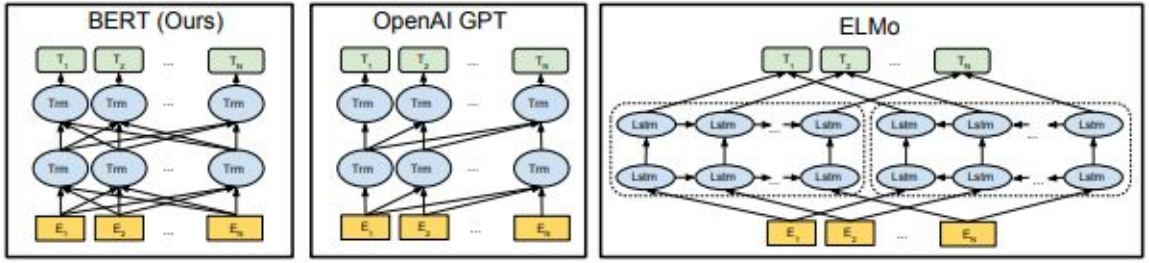
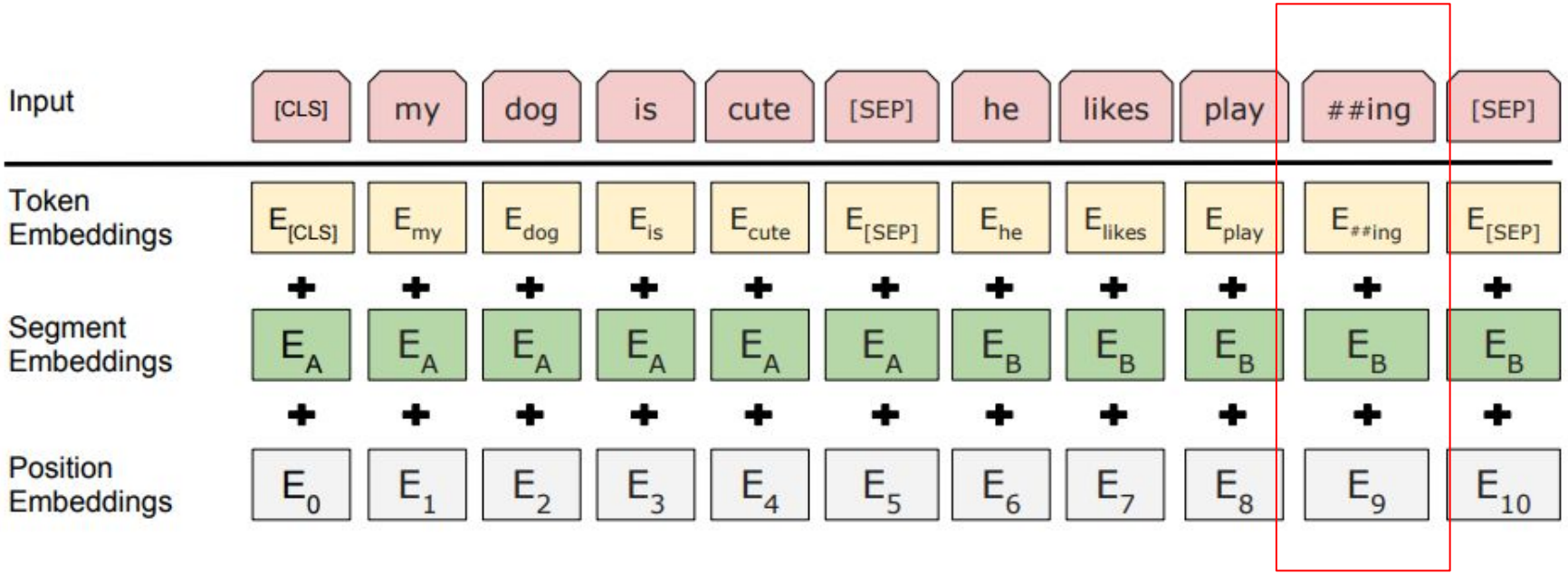


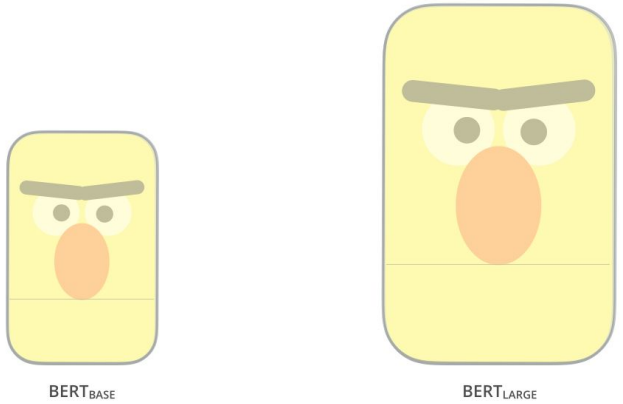
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT 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.

Sub-word embeddings in BERT



BERT versions

- BERT BASE – Comparable in size to the OpenAI Transformer in order to compare performance
- BERT LARGE – A ridiculously huge model which achieved the state of the art results reported in the paper



http://jalammar.github.io/images/bert-base-bert-large.png

Let’s code

https://www.kaggle.com/ahmadelsallab/bert-1

https://www.tensorflow.org/official_models/fine_tuning_bert

TFHub

TensorFlow Hub is a repository of reusable TensorFlow machine learning modules.

Catalogue: https://tfhub.dev/

```
import tensorflow as tf
import pandas as pd
import tensorflow_hub as hub
```

```
bert_path = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"
```

```
self.bert = hub.Module(
    bert_path,
    trainable=self.trainable,
    name="{}_module".format(self.name)
)
```

```
class BertLayer(tf.layers.Layer):
    def __init__(self, n_fine_tune_layers=10, **kwargs):
        self.n_fine_tune_layers = n_fine_tune_layers
        self.trainable = True
        self.output_size = 768
        super(BertLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.bert = hub.Module(
            bert_path,
            trainable=self.trainable,
            name="{}_module".format(self.name)
        )
```

```
# Build model
def build_model(max_seq_length):
    in_id = tf.keras.layers.Input(shape=(max_seq_length,), name="input_ids")
    in_mask = tf.keras.layers.Input(shape=(max_seq_length,), name="input_masks")
    in_segment = tf.keras.layers.Input(shape=(max_seq_length,), name="segment_ids")
    bert_inputs = [in_id, in_mask, in_segment]

    bert_output = BertLayer(n_fine_tune_layers=3)(bert_inputs)
    dense = tf.keras.layers.Dense(256, activation='relu')(bert_output)
    pred = tf.keras.layers.Dense(1, activation='sigmoid')(dense)

    model = tf.keras.models.Model(inputs=bert_inputs, outputs=pred)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()

    return model
```

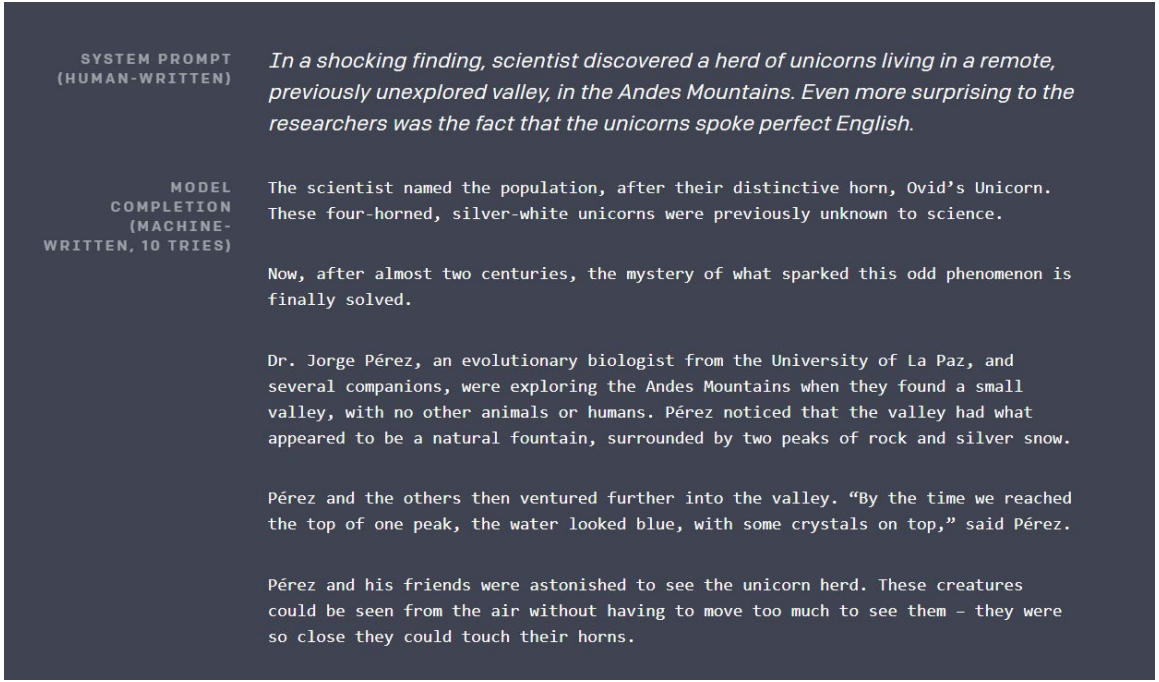
GPT 2 = GPT 1 + Very large!

http://jalammar.github.io/images/gpt2/gpt2-sizes.png

GPT 2: Too good, too dangerous to be released!

Due to our concerns about malicious applications of the technology, we are not releasing the trained model. As an experiment in responsible disclosure, we are instead releasing a much smaller model for researchers to experiment with, as well as a technical paper.

could generate **harmful things** like fake news. At first glance, the contributions are often indistinguishable from real news.



Let’s code: Interview GPT 2

https://towardsdatascience.com/what-does-an-ai-system-think-about-austria-573c2f17ca01

https://colab.research.google.com/drive/1_1iiwkrKZ4TiKXqv50a73DqfU0jRfJZd

```
gpt2.generate(sess,
              model_name="1558M",
              prefix="The most beautiful place in Austria is",
              length=30,
              temperature=0.9,
              top_p=0.9,
              nsamples=3
              )
```

```
The most beautiful place in Austria is in the valley near Vienna. At the top of the valley is the village of Strausbach. It has very good trails, and is a great
====================
The most beautiful place in Austria is near Vienna and this beautiful landmark is about to get more beautiful and gaudy as it is set to be sold to Russian developer STX, and
====================
The most beautiful place in Austria is the Biennale Artistic Villa near Vienna, it's decorated in the old country style. This place is a masterpiece in its own right.

====================
```

GPT-2 writing code

https://www.youtube.com/watch?v=utuz7wBGjKM

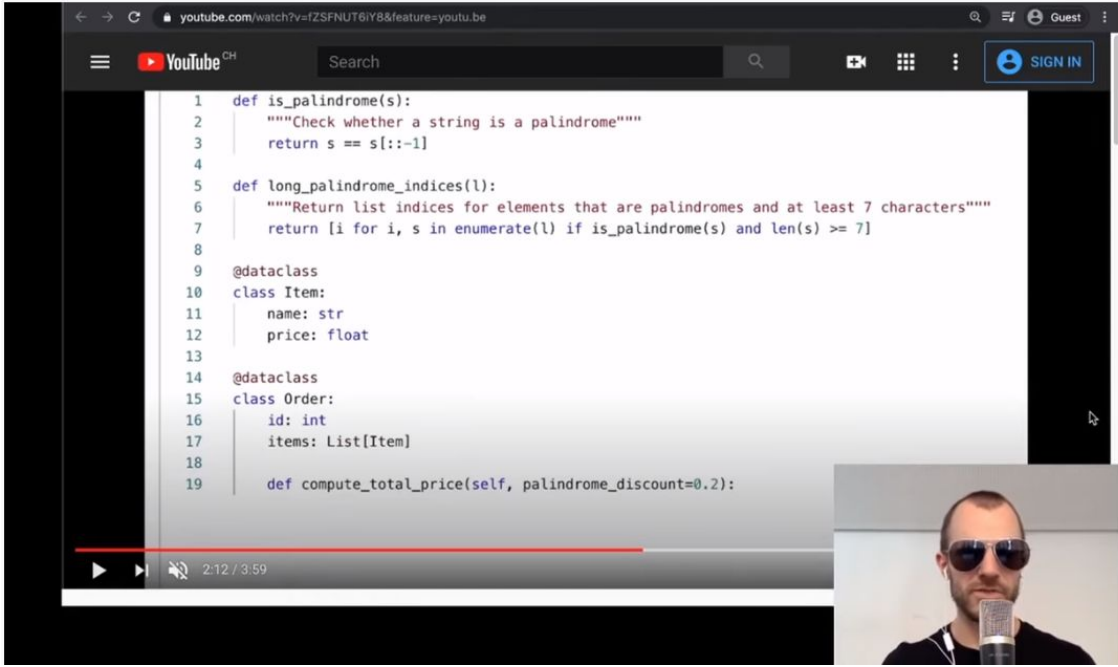
Just write the docstring!

Logically correct code!

Old LM like char RNN → Only syntax correct!

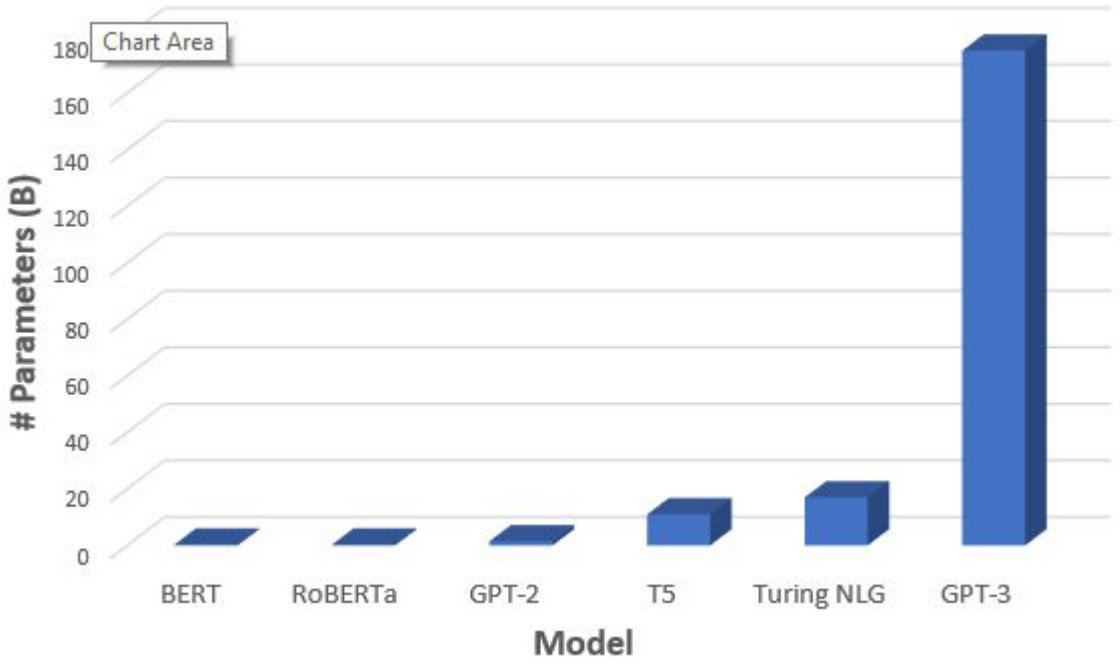
Currently in dev tools!

Kite and Tabnine



GPT 3 = Larger GPT-2 + Few-shot learning

700GB on disk!



https://miro.medium.com/max/582/1*C-KNWQC_wXh-Q2wc6VPK1g.png

GPT-3: Language Models are Few-shot learners

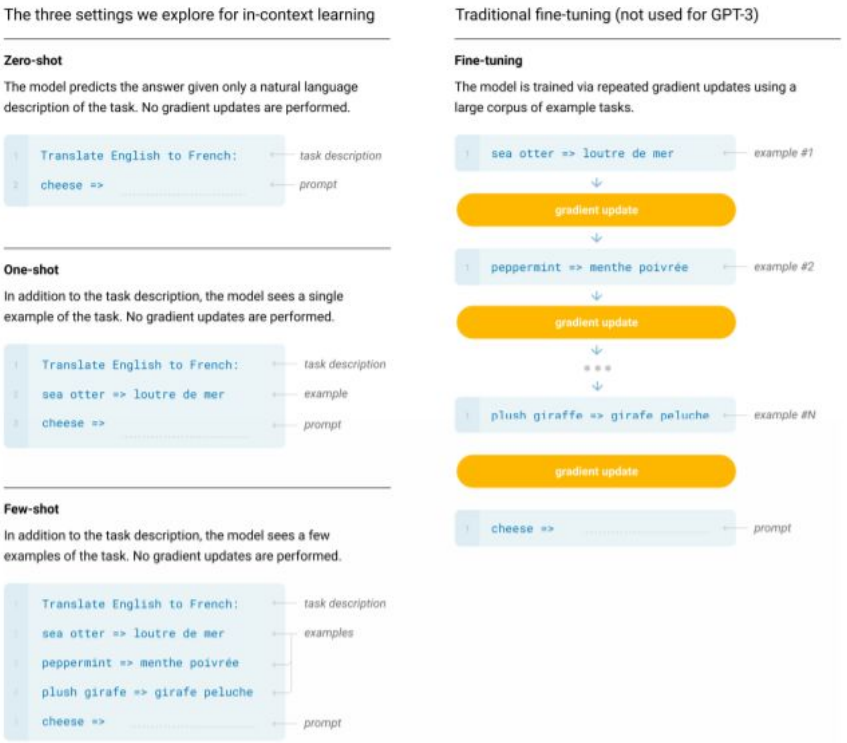
Just train LM

No fine-tuning for downstream tasks!

Unlike BERT

Completely unsupervised

Can be thought of “querying” the big LM



GPT-3: Language Models are Few-shot learners

Just train LM

No fine-tuning for downstream tasks!

Unlike BERT

Completely unsupervised

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

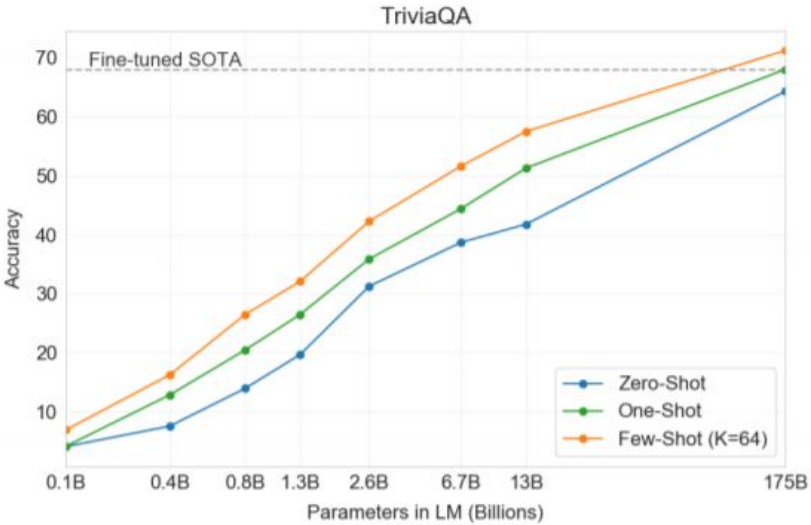
Few-shot vs Fine-tuned

Is it magic? Is it reasoning? Is it AGI?

Can be thought of “querying” the big training data for the most “similar” answers for any question!

This justifies the size of the model

Almost “Encoding” the whole dataset in the “Weights” → sort of overfitting? Under study



RoBERTa

FaceBook

RoBERTa: A Robustly Optimized BERT Pretraining Approach

- To improve the training procedure, RoBERTa r**emoves the Next Sentence Prediction** (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs. Larger batch-training sizes were also found to be more useful in the training procedure.
- Importantly, RoBERTa uses **160 GB** of text for pre-training

ALBERT

Google (Again)

ALBERT: A **Lite** BERT For Self-Supervised Learning of Language Representations

(Bottleneck) Factorized embedding parameterization: Researchers isolated the size of the hidden layers from the size of vocabulary embeddings by **projecting one-hot vectors into a lower dimensional embedding space** and then to the hidden space, which made it easier to increase the hidden layer size without significantly increasing the parameter size of the vocabulary embeddings.

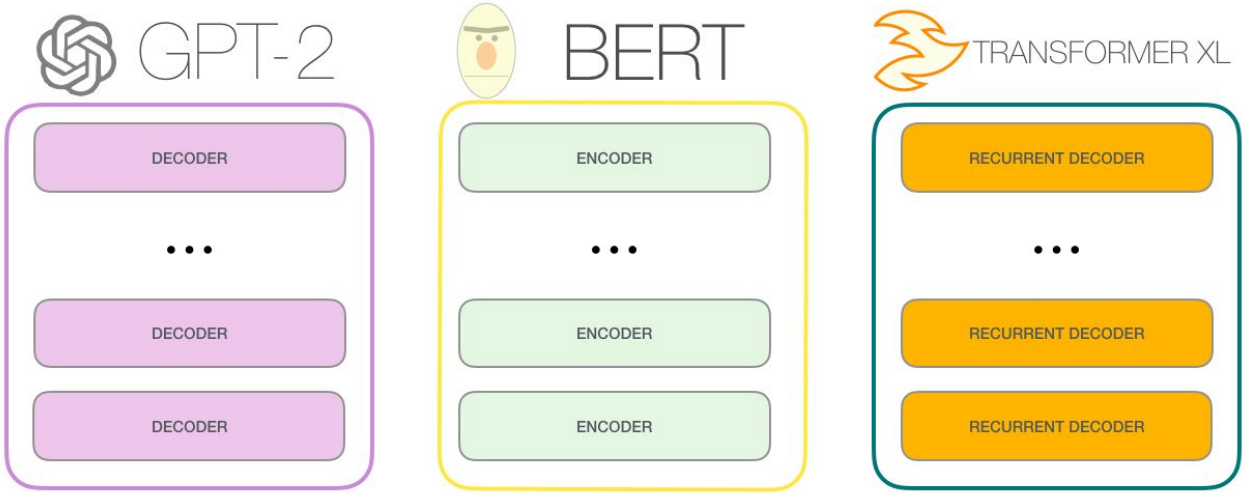
Cross-layer parameter sharing: Researchers chose to **share all parameters across layers to prevent the parameters from growing along with the depth of the network**. As a result, the large ALBERT model has about **18x fewer parameters** compared to BERT-large.

Inter-sentence coherence loss: In the BERT paper, Google proposed a next-sentence prediction technique to improve the model's performance in downstream tasks, but subsequent studies found this to be unreliable. Researchers used a **sentence-order prediction (SOP)** loss to model inter-sentence coherence in ALBERT, which enabled the new model to perform more robustly in multi-sentence encoding tasks.

XLTransformers and XLNet

Xtra-Long Transformer (XLTransformer or Transformer-XL)

Recurrent Transfromer!

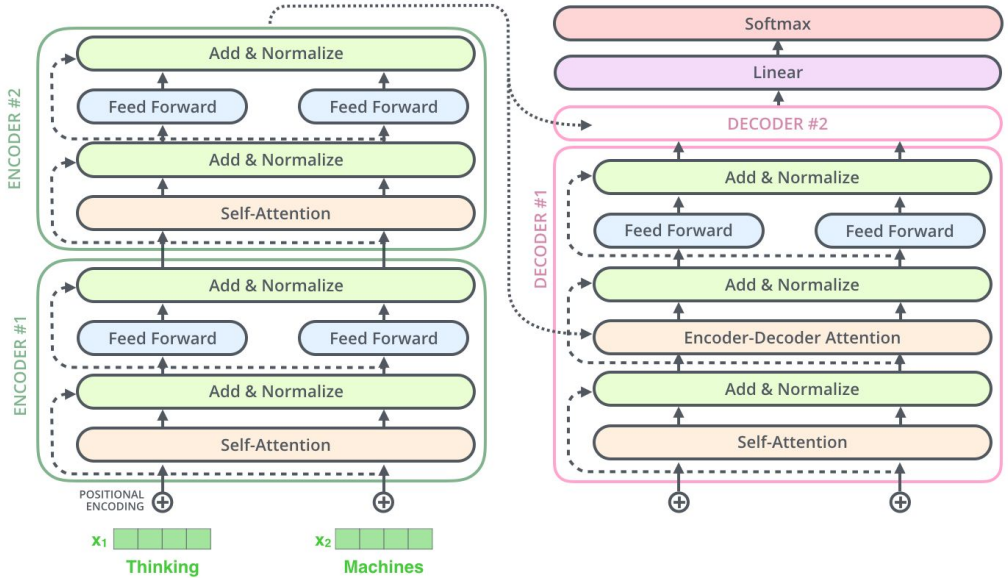


http://jalammar.github.io/images/gpt2/gpt-2-transformer-xl-bert-3.png

What's wrong with Transformers?

A simple attention-based model cannot handle positional information.

The solution to this problem used in the Transformer is simple: add **positional embeddings** to each word that express information regarding the position of each word in a sequence.




```
embeddings = word_embs + pos_embs

h = [embeddings] + [None for _ in attention_layers]


for i, attention in enumerate(attention_layers):

    h[i+1] = attention(queries=h[i], keys=h[i], values=h[i])
```

What's wrong with Transformers?

A simple attention-based model cannot handle positional information.

The solution to this problem used in the Transformer is simple: add **positional embeddings** to each word that express information regarding the position of each word in a sequence.

The normal Transformer is based on tokenizing the input sequence into **segments**.

Such segmentation might **lack long context modeling**, especially for tasks like AR language models.

Position encoding is within a segment only.
It resets over segments.

Not an issue for RNN!

Not an issue for RNN Suppose you had a 50000-word long piece of text that you wanted to feed to a model.

Feeding this into any model all at once would be infeasible given memory constraints.

For an RNN you could work around this by simply chunking the text, then feeding the RNN one chunk at a time ***without resetting the hidden state between chunks.***

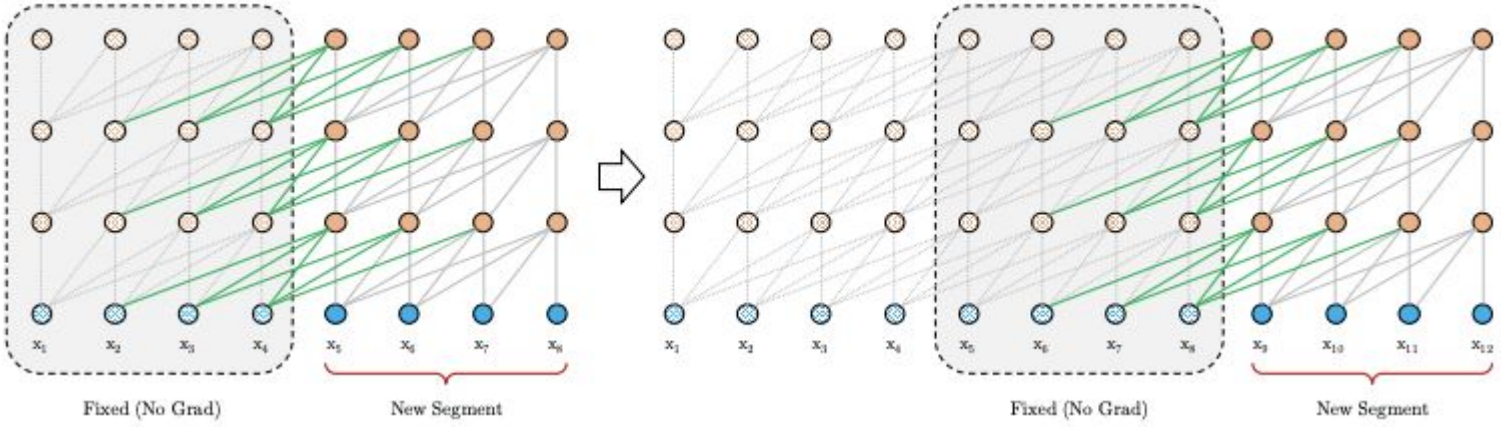
This works because the RNN is recurrent and as long as you keep the hidden state, the RNN can "remember" previous chunks, giving it a theoretically infinite memory.

Xtra-Long Transformer (XLTransformer or Transformer-XL)

The Xtra Long Transformer is an extension over the normal transformer.

The XL-Trans adopts a recurrent decoding technique, accounting for a summary state vector for previous context.

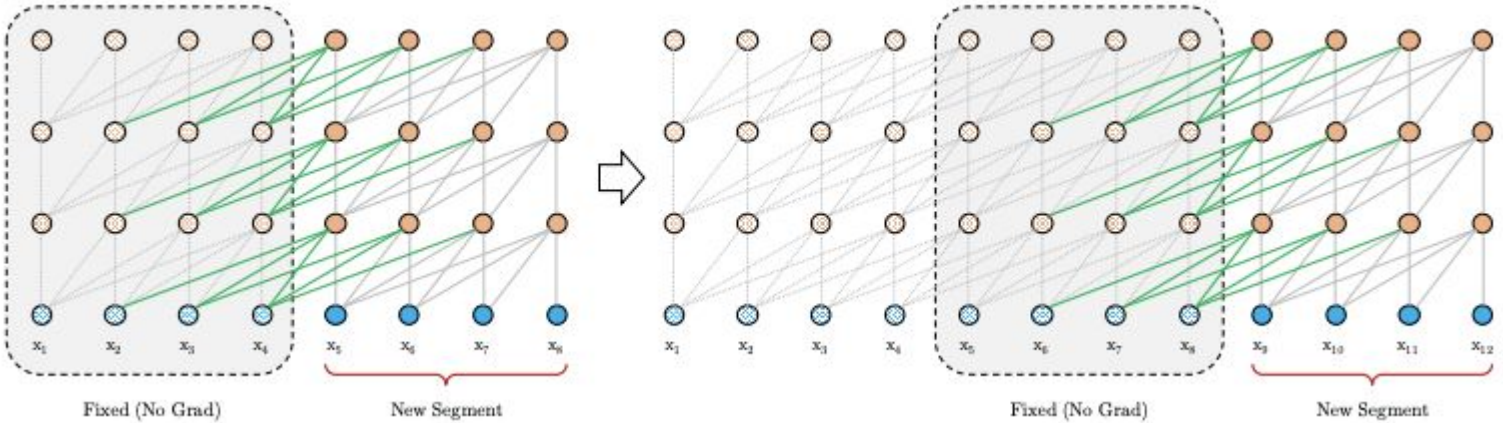
Decoding of next tokens is conditioned on the current segment tokens, in addition to previous context state, under the full attention mechanism of the transformer.



https://miro.medium.com/max/1564/1*633HlVQwSSlvnRwO1cyjmQ.png

Xtra-Long Transformer (XLTransformer or Transformer-XL)

Add RNN to the decoder over the transformer vectors!



https://miro.medium.com/max/1564/1*633HlVQwSSlvnRwO1cyjmQ.png

Trans-XL example

“I went to the store. I bought some cookies.”

we can feed “I went to the store.” first, cache the outputs of the intermediate layers, then feed the sentence “I bought some cookies.” and the cached outputs into the model.

```
memory = init_memory()
h = [embeddings] + [None for _ in attention_layers]

for i, attention in enumerate(attention_layers):
    ext_h_i = concat([h[i], memory[i])
    h[i+1] = Attention(queries=h[i], keys=ext_h_i, values=ext_h_i)

# save memory
memory = h
```

Relative Position Embeddings

In the Transformer, we handled position using **positional embeddings**.

The first word in a sentence would have the “first position” embedding added to it, the second word would have the “second position” embedding added, and so on. But with recurrence, what happens to the positional embedding of the first word in the previous segment? If we’re caching the Transformer outputs, what happens to the positional embedding of the first word in the current segment?

You can easily have sequences like [1,2,3,4], [1,2,3,4]–> [1,2,3,4,1,2,3,4]

To address these issues, the Transformer XL introduces the notion of r**elative positional embeddings**.

Instead of having an embedding represent the absolute position of a word, the Transformer XL uses an embedding to encode the **relative distance between words**.

This embedding is used while computing the attention score between any two words: in other words, the relative positional embedding enables the model to learn how to compute the attention score for words that are n words before and after the current word.

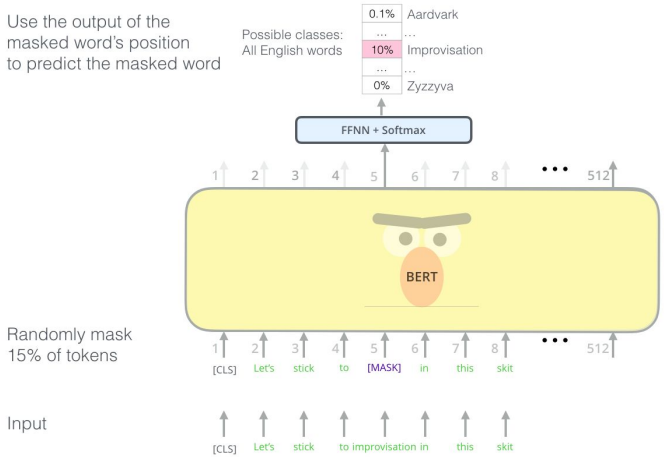
What's wrong with BERT?

AE LM:
- Parallel decoding

"New York is a city"

[MASK] [MASK] is a city → New, York

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city})$$



What's wrong with BERT?

AE LM:
- Parallel decoding

"New York is a city"

[MASK] [MASK] is a city → New, York

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city})$$

No relation between New and York

So if a model predicts "New", it can also predicts "Angeles" and still the overall loss is low, since "Angeles" is a second name of a city same as "York.
I.e. there's no clue that "New" is already predicted, and so "York" is a higher match.

What's wrong with BERT?

AE LM:
- Parallel decoding

"New York is a city"

[MASK] [MASK] is a city → New, York

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city})$$

No relation between New and York

For an AR LM →

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city})$$

Sequential → But slow
+ One directions → York always looks back

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid \mathbf{x}_{<t})$$

What’s wrong with BERT?

“New York is a city”

[MASK] [MASK] is a city → New, York

MASK is not a valid token! It won’t be encountered in reality

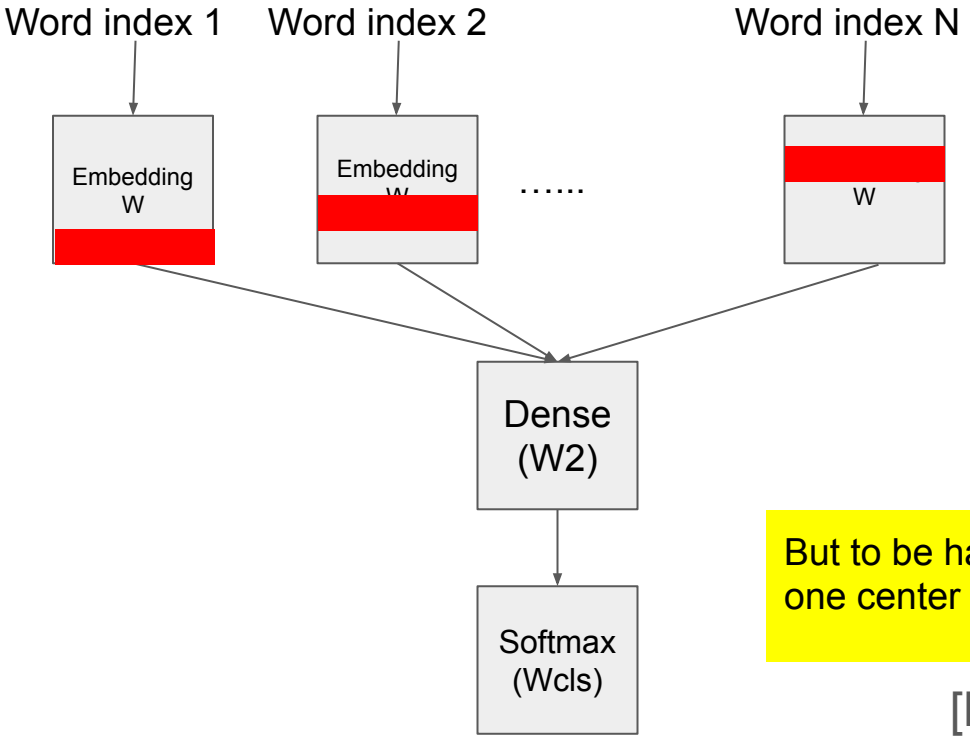
That’s why, 10-15% of the time, the word is not [MASK] but a normal inflected word (like SGNS).

But this even introduces high noise in the training data

Possible fixes

- Switch back to AR LM
- But needs to account for Bi-directions + be fast

CBOW



Jay was hit by a _____ bus in...

by	a	red	bus	in

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

http://jalammar.github.io/illustrated-word2vec/

Context = by a _ bus in → Bi-directional
Center = red → out → No [MASK]

But to be have rich context, we might need to predict MORE than one center given different contexts

[MASK] [MASK] is a city → New, York

Permutations Language Model (PLM)

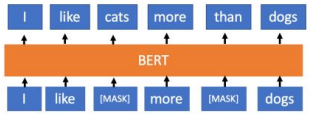
AR LM: A traditional language model would predict the tokens in the order

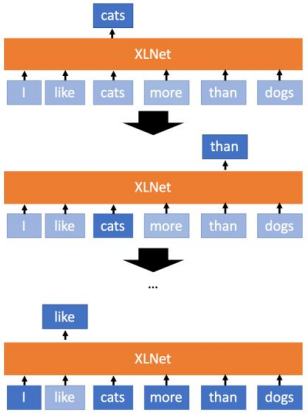
"I", "like", "cats", "more", "than", "dogs"

In permutation language modeling, the order of prediction is not necessarily left to right and is sampled randomly instead. For instance, it could be

"cats", "than", "I", "more", "dogs", "like"

where "*than*" would be conditioned on seeing "*cats*", "*I*" would be conditioned on seeing "*cats*, *than*" and so on.

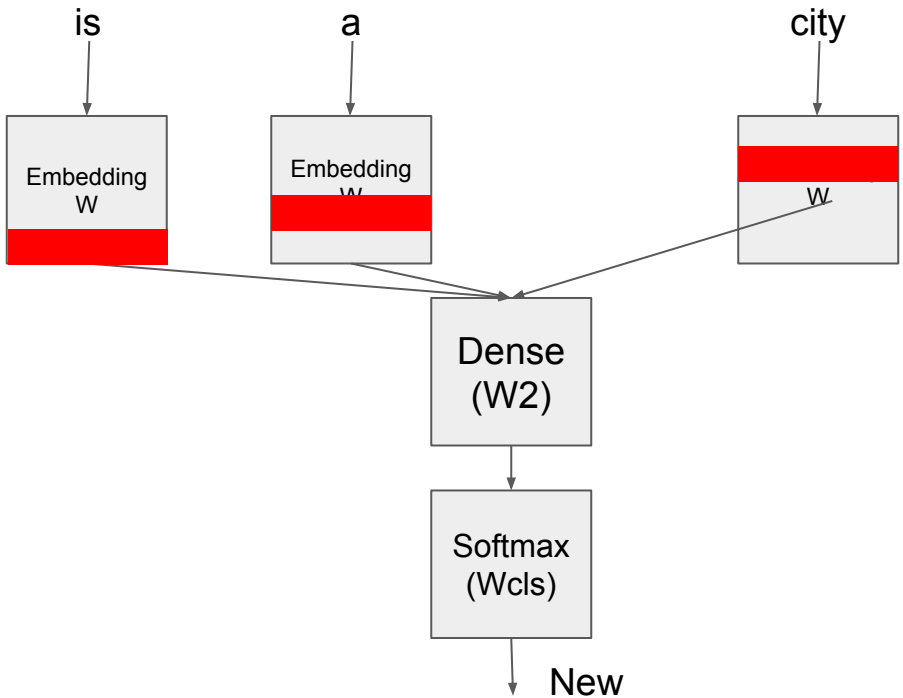




Remember we use Recurrent Trans-XL, so state propagates between feeding each token

Permutations Language Model (PLM)

- Randomly generate a permutation for a context “factorization”: p(New | is, a, city)



Permutations Language Model (PLM)

- Repeat for different permutations = different contexts

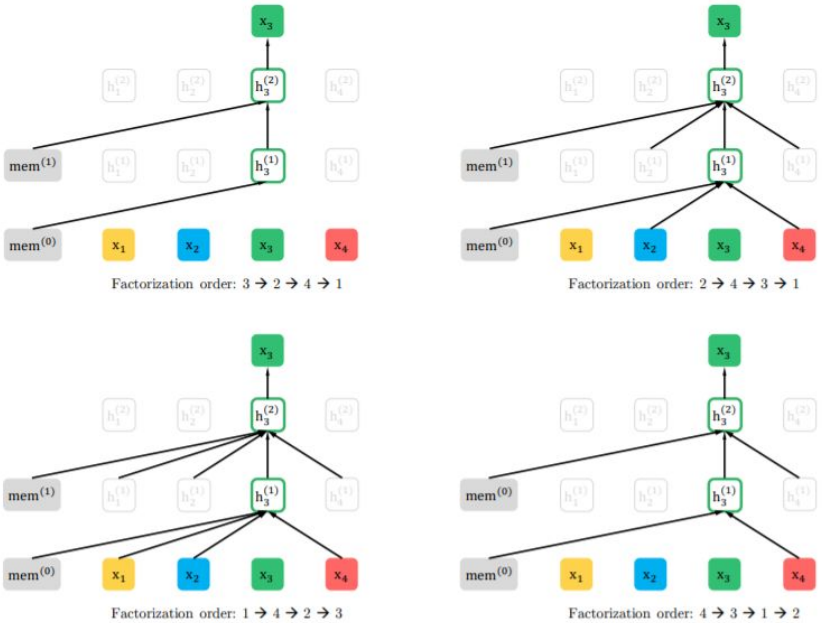


Figure 4: Illustration of the permutation language modeling objective for predicting x_3 given the same input sequence x but with different factorization orders.

Every time x3 is predicted | its **<u>preceeding context</u>** = AR
1- nothing before x3 in the factorization, so no arrows to predict x3
2- 2-->4-->3 ⇒ arrows from x4 and x2

Note that: the order is not changed.
Same as in CBOW
2→ 4→ 3 does not mean the order of the words is x2 → x4 → x3
But it only means predict x3 given the context of x2 and x4 (see CBOW)
And that's what makes it Bi-directional, because in this context x2 is before x3 but x4 is after.

PLM - Order is preserved → Just change the context in Bi-directions!

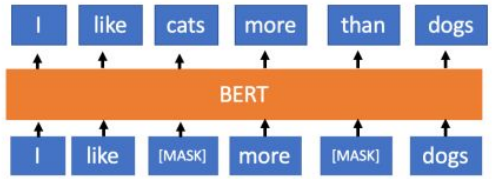
As a word of caution, in permutation language modeling, we are not changing the *actual* order of words in the input sentence.

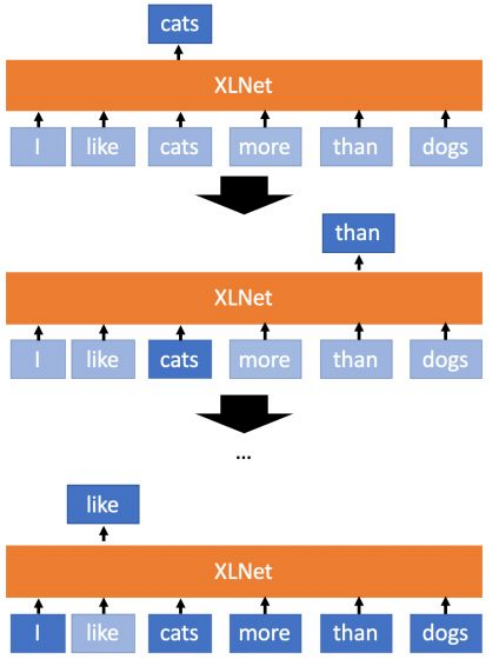
We are just changing the order in which we *predict* them.

Just remember that Transformers use masking to choose which inputs to feed into the model and use positional embeddings to provide positional information.

This means that we can feed input tokens in an arbitrary order simply by adjusting the mask to cover the tokens we want to hide from the model. As long as we keep the positional embeddings consistent, the model will see the tokens "in the right order".

With enough permutations, ALL the context is “seen”





How is sequential AR happening?

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city})$$

Two-stream self-attention

We keep 2 vectors: “New” as an output a separate vector from “New” as an input/context.

So “New” is conditioned on “is a city”

While “York” is conditioned on “New”, “is a city”

With **Teacher forcing** → “New” = predicted (10%) → AR: sequential, or GT (90%) → AE: Parallel

How is sequential AR happening?

$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city})$$

2.3 Architecture: Two-Stream Self-Attention for Target-Aware Representations

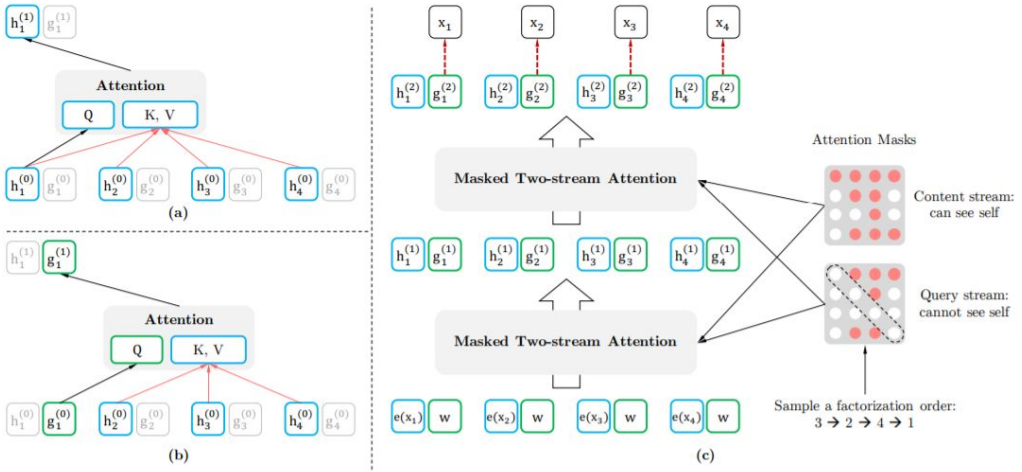


Figure 1: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content x_{z_t}. (c): Overview of the permutation language modeling training with two-stream attention.

XLNet=

- Trans-XL
- + PLM

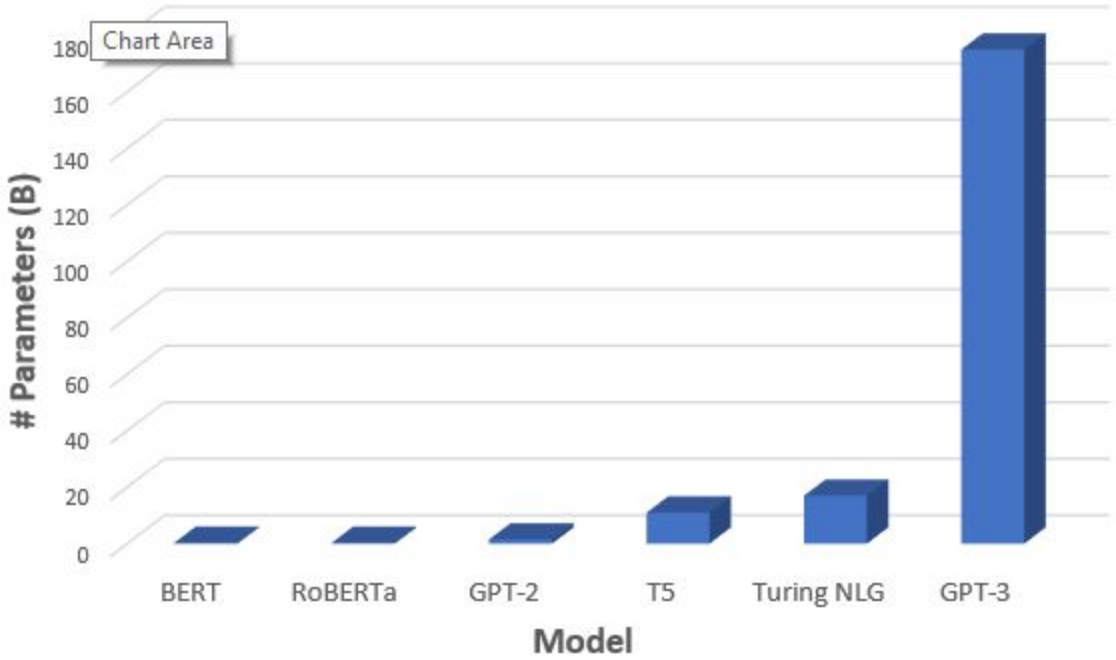
How practical are the Gignatic models?

GPT 2 = GPT 1 + Very large!

http://jalammar.github.io/images/gpt2/gpt2-sizes.png

GPT 3

700GB on disk!



https://miro.medium.com/max/582/1*C-KNWQC_wXh-Q2wc6VPK1g.png

Huge data and Huge Compute

https://arxiv.org/pdf/2005.14165.pdf

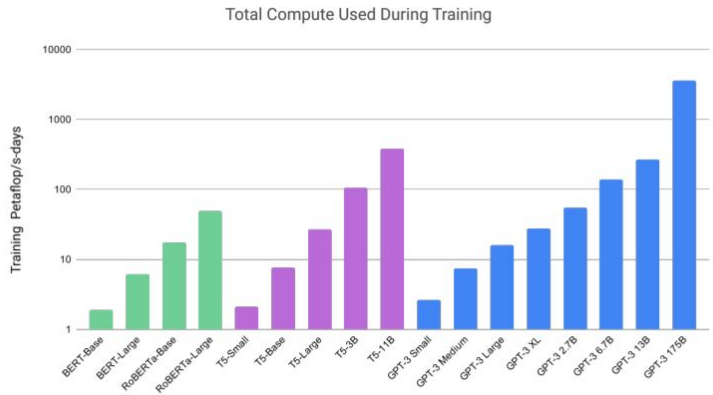


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

Batch size ~= Megas!

https://arxiv.org/pdf/2005.14165.pdf

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

One error = Millions $!

https://arxiv.org/pdf/2005.14165.pdf

Some results were too good → Leak → Contamination of training with similar data⇒ Cannot repeat the training, it costs a lot of power and money! → Just admit the bug

Setting	Winograd	Winogrande (XL)
Fine-tuned SOTA	**90.1**[a]	**84.6**[b]
GPT-3 Zero-Shot	88.3*	70.2
GPT-3 One-Shot	89.7*	73.2
GPT-3 Few-Shot	88.6*	77.7

Table 3.5: Results on the WSC273 version of Winograd schemas and the adversarial Winogrande dataset. See Section 4 for details on potential contamination of the Winograd test set. [a][SBBC19] [b][LYN+20]

Cost of XLNet training from scratch

https://www.youtube.com/watch?v=SY5PvZrJhLE

What to do?

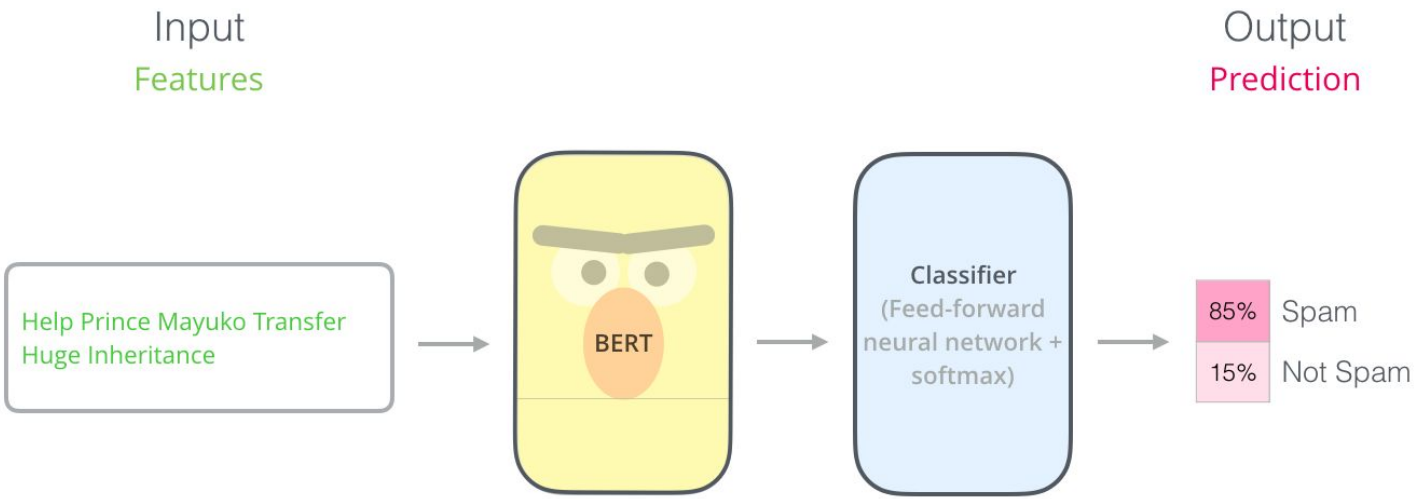
It’s beyond individual or small/medium companies!

1- Fine tune

2- Compress

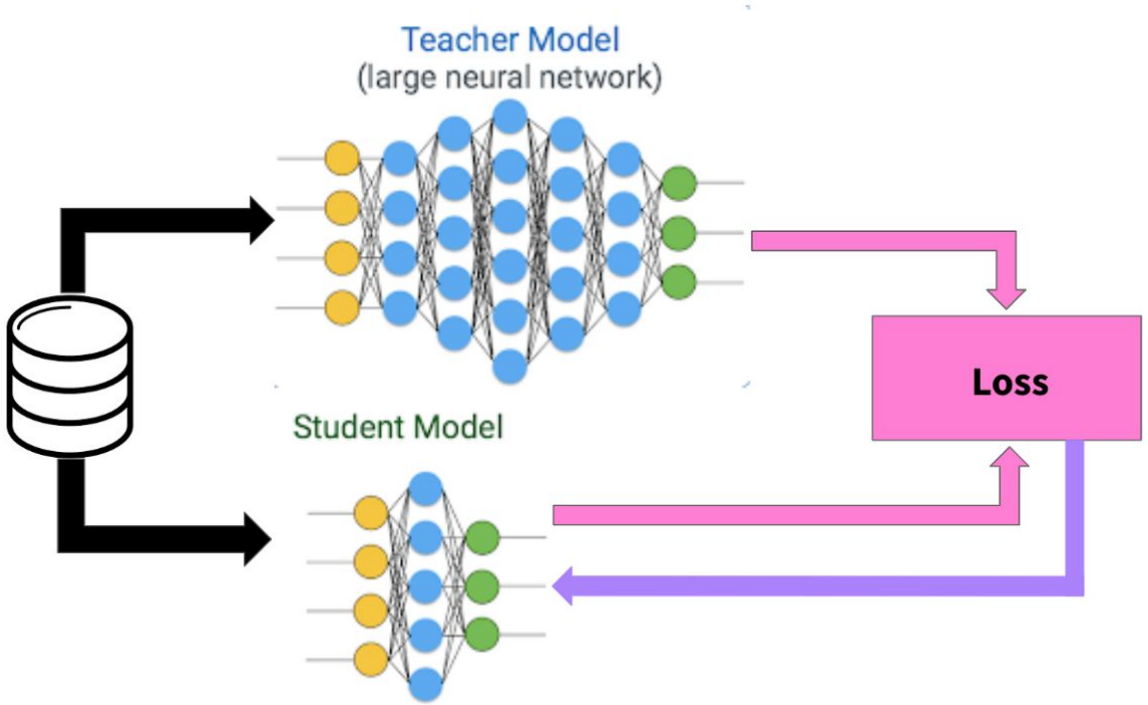
BERT fine tuning

What if new language? Arabic? Chinese?



Knowledge Distillation

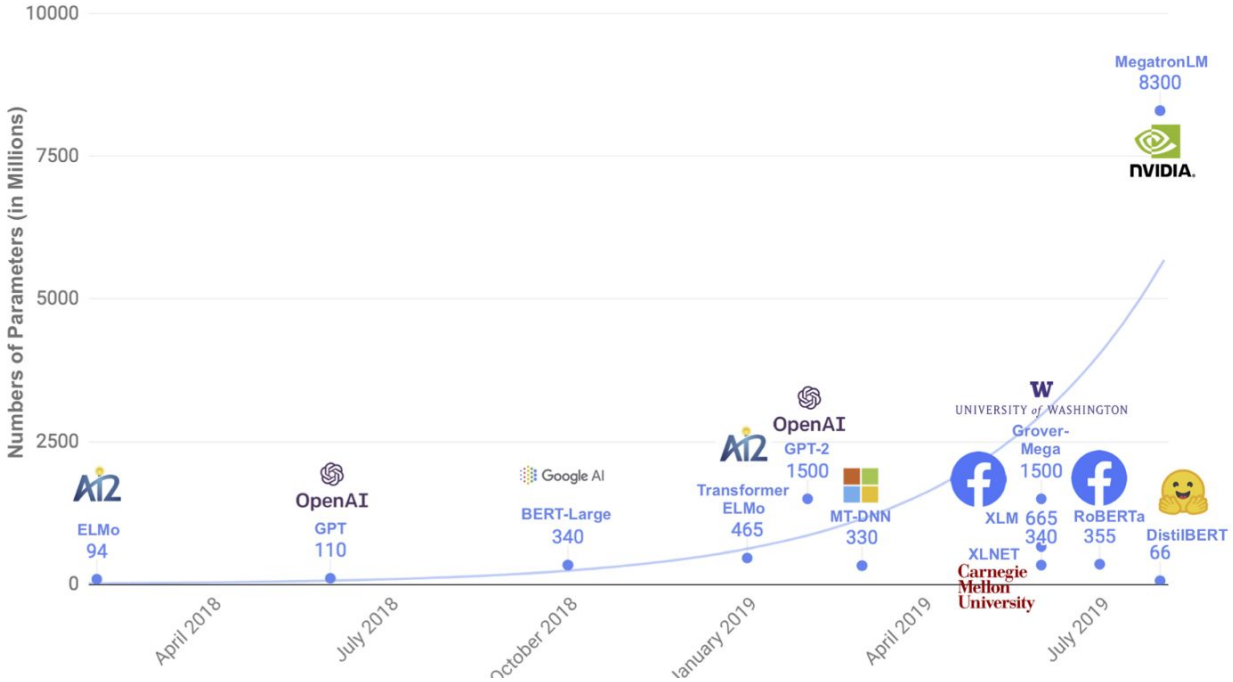
Hinton et al. (2015)



https://miro.medium.com/max/1200/1*DdClMPqhErordaun8Dw14Q.png

DistilBERT

[DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter]



https://medium.com/dair-ai/tl-dr-distillbert-8fb0f9e3c03d

Conclusion

- NLP ImageNet moment
- Word level TL → word2vec
- TL → LM as src task: common + self-supervised
- Sentence level TL → ULMFiT
- GPT → ULMFiT + Transformer
- BERT = GPT + Bi-directional + MLM
  - ALBERT
  - RoBERTa
  - … Engineering hacks
- XLNet = BERT + PLM + XLTrans

References

http://web.stanford.edu/class/cs224n/

http://jalammar.github.io/illustrated-word2vec/

http://jalammar.github.io/illustrated-transformer/

http://jalammar.github.io/illustrated-bert/

http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and

- Deep Learning with Python, Fchollet (Keras)
  - Notebooks
- Hands-On Machine Learning with Scikit-Learn and TensorFlow
  - Notebooks
- Ian Goodfellow DeepLearning Book

References

- http://jalammar.github.io/illustrated-gpt2/
- https://arxiv.org/abs/1906.08237
- https://arxiv.org/abs/1801.06146
- https://arxiv.org/abs/1706.03762
- https://arxiv.org/abs/1906.08237
- https://arxiv.org/abs/1901.02860
- https://arxiv.org/abs/1910.07370
- https://openai.com/blog/better-language-models/
- https://github.com/openai/gpt-3