Deep Learning in Life Sciences

Transformers and RNN for sequence analysis

D. Malchiodi, 08/04/2024

CHARLES UNIVERSITY
SORBONNE UNIVERSITÉ
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
UNIVERSITY OF WARSAW
UNIVERSITÀ DEGLI STUDI DI MILANO
4eu+ EUROPEAN UNIVERSITY ALLIANCE

Who am I?

dario.malchiodi@unimi.it
https://malchiodi.di.unimi.it

TEACHING

Associate professor @unimi (statistics & data analysis, algorithms for massive datasets)

RESEARCH

Data-driven induction of non-classical sets, compression of ML models, negative example selection, application of ML to medicine, veterinary, forensics & cultural heritage. Visiting scientist @uca @inria

POPULARIZATION OF COMPUTING

Italian National Science and Technology museum, RadioPopolare, ALaDDIn

Sequence analysis

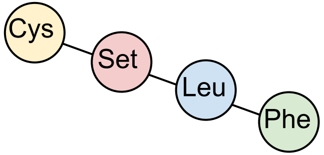
It might mean a lot of things:
- analysis of generated sequences
- sequence classification
- token classification
- sequence translation
- sequence summarization
- question answering

It can be done with several tools, among which:
- RNNs
- Transformers

Sequence generation

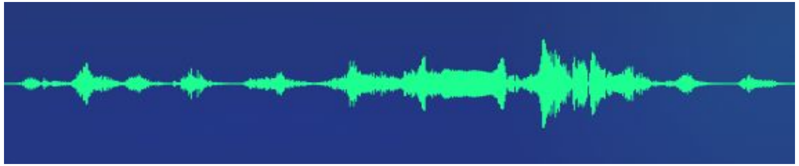
– Natural language
– Proteins
– Code
– Audio, music notation, …

Certainly! Here is a list of…



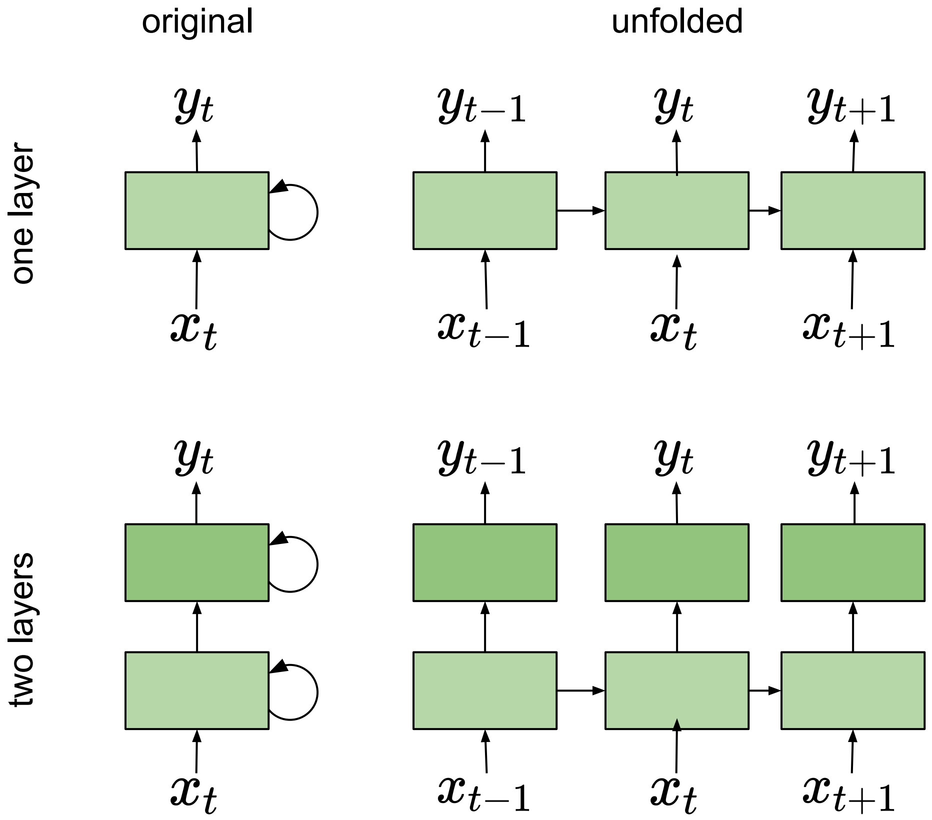

```c
#include <linux/buffer_head.h>
#include <linux/module.h>
#include <linux/fs.h>
#include "efs.h"
#include <linux/efs_fs_sb.h>

static int efs_read_folio(struct file *file, struct folio *folio)
{
        return block_read_full_folio(folio, efs_get_block);
}

static sector_t _efs_bmap(struct address_space *mapping, sector_t block)
{
        return generic_block_bmap(mapping,block,efs_get_block);
}
```

Image source: Library of Congress, public domain



RNNs: a quick recap

– Now with intra-layer self-loops.
– Add «state» to neurons.
– Exist in several flavours.
– We will use the «Long Short-Term Memory» architecture.



Teacher forcing

«Correct» possible errors of the model while feeding back the generated sequence:

for $i = 1, \ldots n$

feed the correct symbols $x_1 \cdots x_{i-1}$

generate symbol $\widehat{x}_i$

Seq. generation: hallucinated Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

3-layer RNN, each with 512 hidden nodes

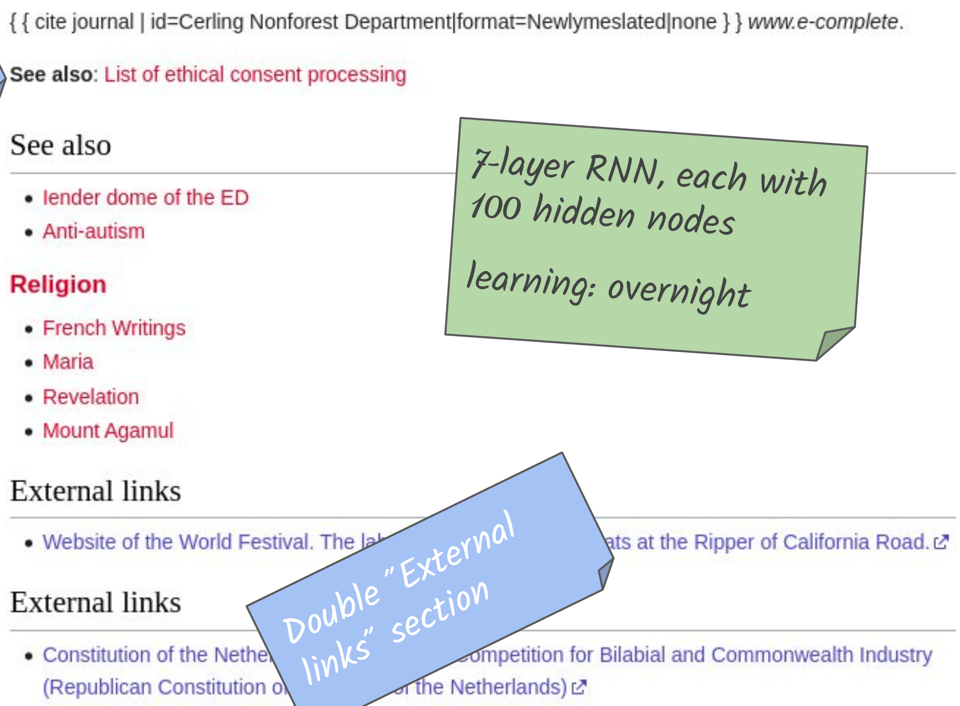
learning: a few hours

Examples from: A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks [1]

Seq. generation: hallucinated Wikipedia

```
{ { cite journal | id=Cerling Nonforest
Department|format=Newlymeslated|none } }
''www.e-complete''.

'''See also''': [[List of ethical

== See also ==
*[[Iender dome of the ED]]
*[[Anti-autism]]

===[[Religion|Religion]]===
*[[French Writings]]
*[[Maria]]
*[[Revelation]]
*[[Mount Agamul]]

== External links==
* [http://www.biblegateway.nih.gov/entrepre/ Website of the
World Festival. The labour of India-county defeats at the
Ripper of California Road.]

==External links==
* [http://www.romanology.com/ Constitution of the Netherlands
and Hispanic Competition for Bilabial and Commonwealth
Industry (Republican Constitution of the Extent of the
Netherlands)]
```

Wrong citation format



7-layer RNN, each with 100 hidden nodes

learning: overnight

Double "External links" section

Hallucinated LaTeX

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on
$\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to
\mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian
sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let
$\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on
$\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module.
Then
$\mathcal{F}$ is an abelian catenary over
$\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U$$

is an open subset of X. Thus U is affine. Th... map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Almost compiles...

Seq. generation: hallucinated Linux kernel

```
/*
 * Increment the size file of the new incorrect
UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
```

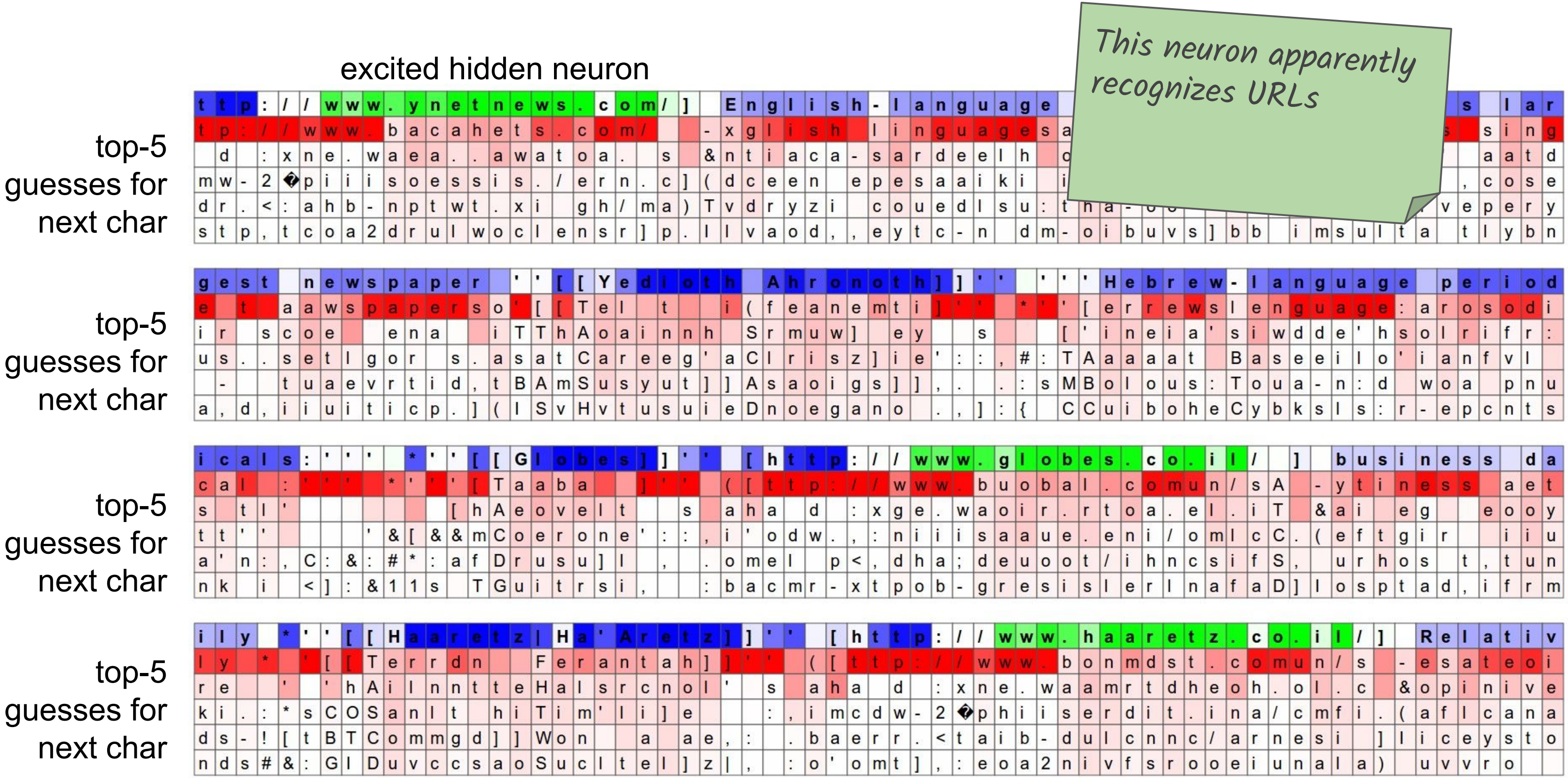
```
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] |
PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Doesn't really compile...

3-layer RNN, each with 512 hidden nodes

learning: a few days

Peeking into RNNs

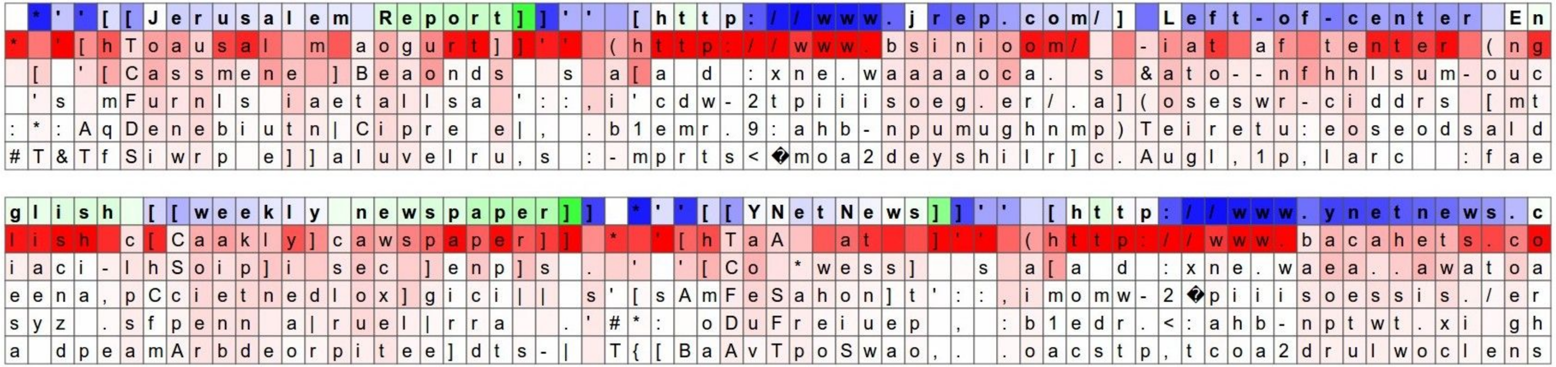


Peeking into RNNs

This neuron apparently recognizes relative positions within URLs

hidden neuron activation fading in

top-5 guesses for next char



top-5 guesses for next char

Peeking into RNNs

This neuron apparently detects comments and strings.

Measuring validity of generated sequences

– Accuracy
– Soft accuracy
– Perplexity

Accuracy

Actual sequence $\mathbf{x}$ $\boxed{x_1 \cdots x_n}$

Predicted sequence $\widehat{\mathbf{x}}$ $\boxed{\widehat{x}_1 \cdots \widehat{x}_n}$

$$\mathrm{acc}(\mathbf{x}, \widehat{\mathbf{x}}) = \sum_{i=1}^{n} \delta(x_i, \widehat{x}_i)$$

$$\delta(x_i, \widehat{x}_i) = \begin{cases} 1 & \text{if } x_i = \widehat{x}_i, \\ 0 & \text{otherwise.} \end{cases}$$

Test set $\mathbf{x}^1, \ldots, \mathbf{x}^m$

$$\overline{\mathrm{acc}} = \frac{1}{m} \sum_{j=1}^{m} \mathrm{acc}(\mathbf{x}^j, \widehat{\mathbf{x}}^j)$$

$$\sigma_{\mathrm{acc}} = \sqrt{\frac{1}{m-1} \sum_{j=1}^{m} \left(\mathrm{acc}(\mathbf{x}^j, \widehat{\mathbf{x}}^j) - \overline{\mathrm{acc}} \right)^2}$$

Soft accuracy

Actual sequence $\mathbf{x}$ $\boxed{x_1 \cdots x_n}$

Predicted sequence $\widehat{\mathbf{x}}$ $\boxed{\widehat{x}_1 \cdots \widehat{x}_n}$

$$\mathrm{softacc}(\mathbf{x}, \widehat{\mathbf{x}}) = \sum_{i=1}^{n} \tilde{\delta}(x_i, \widehat{x}_i)$$

$$\tilde{\delta}(x_i, \widehat{x}_i) = \begin{cases} 1 & \text{if } x_i \approx \widehat{x}_i, \\ 0 & \text{otherwise.} \end{cases}$$

$x_i \approx \widehat{x}_i$

If the prediction is a «good» match

Soft accuracy

Examples of a «good match»:

– synonyms in text generation

We believe the President was fair/objective/impartial

– similarities via BLOSUM score in protein generation

	C	S	T	A	G	P	D	E	Q	N	H	R	K	M	I	L	V	W	Y	F	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
A	0	1	0	4																	A
G	-3	0	-2	0	6																G
P	-3	-1	-1	-1	-2	7															P
D	-3	0	-1	-2	-1	-1	6														D
E	-4	0	-1	-1	-2	-1	2	5													E
Q	-3	0	-1	-1	-2	-1	0	2	5												Q
N	-3	1	0	-2	0	-2	1	0	0	6											N
H	-3	-1	-2	-2	-2	-2	-1	0	0	1	8										H
R	-3	-1	-1	-1	-2	-2	-2	0	1	0	0	5									R
K	-3	0	-1	-1	-2	-1	-1	1	1	0	-1	2	5								K
M	-1	-1	-1	-1	-3	-2	-3	-2	0	-2	-2	-1	-1	5							M
I	-1	-2	-1	-1	-4	-3	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-1	-4	-3	-4	-3	-2	-3	-3	-2	-2	2	2	4					L
V	-1	-2	0	0	-3	-2	-3	-2	-2	-3	-3	-3	-2	1	3	1	4				V
W	-2	-3	-2	-3	-2	-4	-4	-3	-2	-4	-2	-3	-3	-1	-3	-2	-3	11			W
Y	-2	-2	-2	-2	-3	-3	-3	-2	-1	-2	2	-2	-2	-1	-1	-1	-1	2	7		Y
F	-2	-2	-2	-2	-3	-4	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	1	3	6	F
	C	S	T	A	G	P	D	E	Q	N	H	R	K	M	I	L	V	W	Y	F	

Y is a good match for F
I is a good match for V
E is a good match for Q

Perplexity

- Sequence $x_1 \cdots x_n$
- Learnt probability distribution p_θ

$$\text{perplexity}(x_1 \cdots x_n) = \left(\frac{1}{p_\theta(x_1 \cdots x_n)}\right)^{-n} = \left(\prod_i \frac{1}{p_\theta(x_i | x_{<i})}\right)^{-n}$$

Geometric mean, i-th term low if small uncertainty in producing i-th sequence element.

- Log perplexity = cross-entropy

$$\log \text{perplexity}(x_1 \cdots x_n) = -\frac{1}{n} \sum_i \log p_\theta(x_i \mid x_{<i}) = \widetilde{H}_\theta(x_1 \cdots x_n)$$

$$\text{perplexity}(x_1 \cdots x_n) = 2^{\widetilde{H}_\theta(x_1 \cdots x_n)}$$

- Cross-entropy: average # bits to represent the sequence elements using the learnt model.
- Note that cross-entropy is minimized when the learnt distribution equals the actual one (which we don't know).

Perplexity

– If each element of the sequence is emitted without uncertainty

$$p_\theta(x_i \mid x_{<i}) = 1$$

thus the perplexity is 1 (each log nullifies).

– The classical definition of perplexity requires logarithms in base 2 (cross-entropy measured in bits).

– Pytorch used natural logarithms (cross-entropy measured in nats), thus perplexity becomes

$$\mathrm{perplexity}(x_1 \cdots x_n) = \mathrm{e}^{\widetilde{H}_\theta(x_1 \cdots x_n)}$$

Computing perplexity in LLMs

Split the generated sequence using a sliding window

Hugging Face is a startup based in NY and Paris

$p_\theta(x_1)$

Hugging Face is a startup based in NY and Paris

$p_\theta(x_2 \mid x_{<2})$

Hugging Face is a startup based in NY and Paris

$p_\theta(x_3 \mid x_{<3})$

Hugging Face is a startup based in NY and Paris

$p_\theta(x_3 \mid x_{<3})$

Hugging Face is a startup based in NY and Paris

$p_\theta(x_4 \mid x_{<4})$

Compute log cross-entropy each time, average, and compute exponentiation.

Example from practical part

```
max_length = model.config.n_positions
stride = 512
seq_len = encodings.input_ids.size(1)

nlls = []
prev_end_loc = 0
for begin_loc in tqdm(range(0, seq_len, stride)):
   end_loc = min(begin_loc + max_length, seq_len)
   trg_len = end_loc - prev_end_loc  # may be different from stride on last loop
   input_ids = encodings.input_ids[:, begin_loc:end_loc].to(device)
   target_ids = input_ids.clone()
   target_ids[:, :-trg_len] = -100

   with torch.no_grad():
       outputs = model(input_ids, labels=target_ids)

       # loss is calculated using CrossEntropyLoss which averages over valid labels
       # N.B. the model only calculates loss over trg_len - 1 labels, because it internally shifts the labels
       # to the left by 1.
       neg_log_likelihood = outputs.loss

   nlls.append(neg_log_likelihood)
   prev_end_loc = end_loc
   if end_loc == seq_len:
       break

ppl = torch.exp(torch.stack(nlls).mean())
```

Conditional generation (NLP)

In a nutshell: prepend «control codes» to sentences, to add information about
– style (review, horror, …)
– source (e.g., subreddits)

Available in Hugging Face

Most known attempt (so far): CTRL [3]

Technically speaking: now we learn the probability of a sequence *conditioned* on the control codes:

$$p_\theta(x \mid c) = \prod_i p_\theta(x_i \mid x_{<i}, c)$$

Conditional generation (proteins)

Can be done also outsides NLP: ProGen [4] uses a similar approach in protein generation, using as control codes

- keyword tags (cellular component, biological process, and molecular function terms)
- taxonomic tags

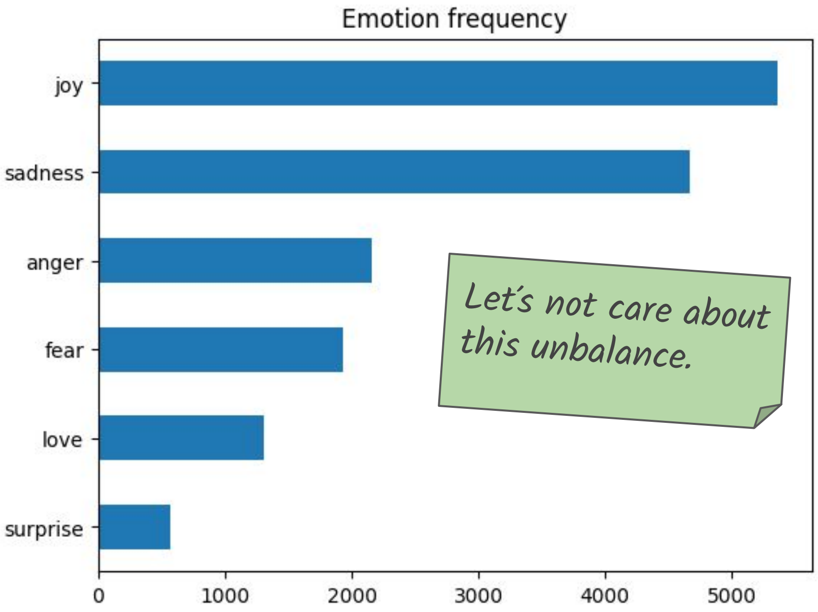
Available in Hugging Face

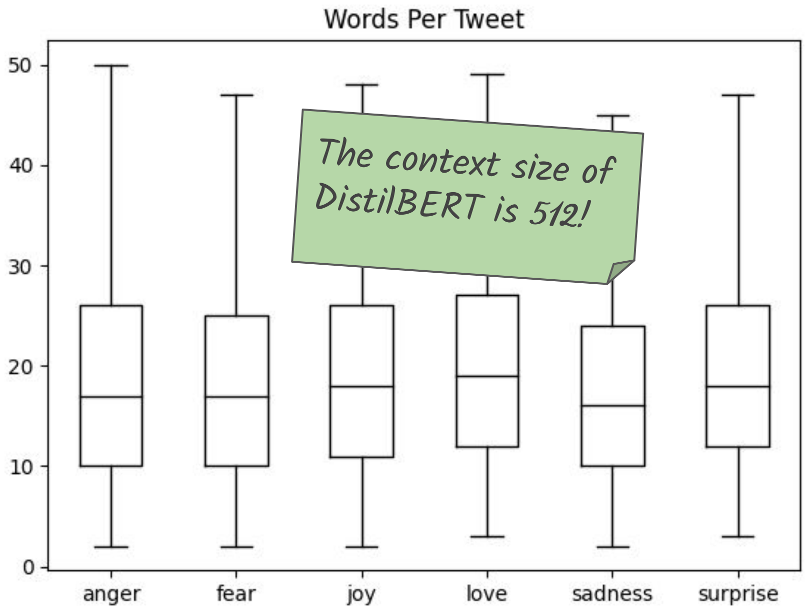
Table 1 from [4]

MODEL	PPL	HARD ACC.
UNIFORM BASELINE	25	4
EMPIRICAL BASELINE	18.14	6
PROGEN	8.56	45
ID-TEST	8.17	45
OOD-TEST	13.34	22
OOD-TEST-20 (RAND. INIT.)	17.78	9
OOD-TEST-20 (FINE-TUNED)	7.45	50

Sequence classification

Running example: sentiment analysis of tweets

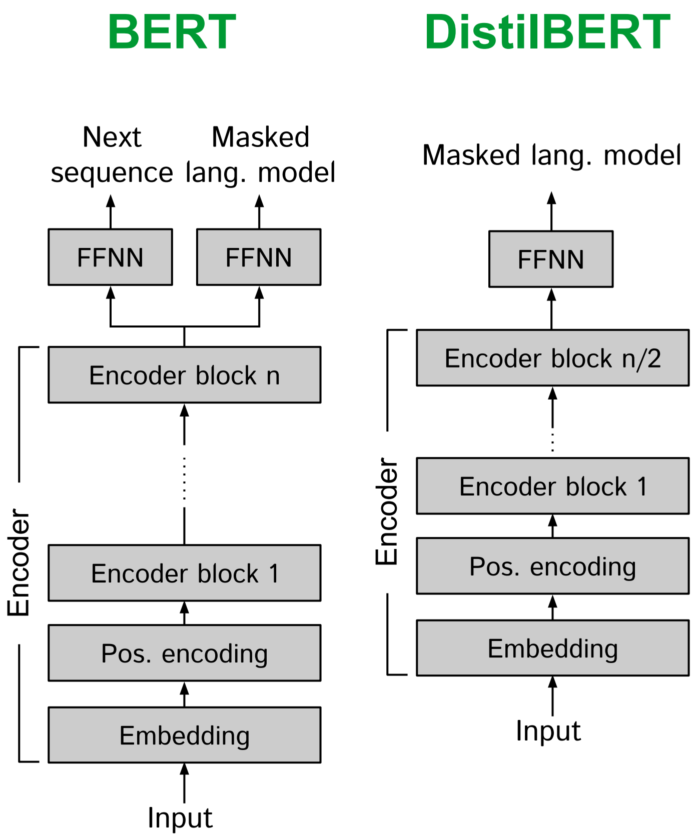




«emotion» dataset in Hugging Face

DistilBERT

- Recall BERT?
- DistilBERT is a thinner version.
- Half of the encoder blocks + various magics.
- 60% of the original size, 60% faster, 97% of original performances.



You know the basics

- Tokenization, positional encoding, and so on.
- Just to make a connection

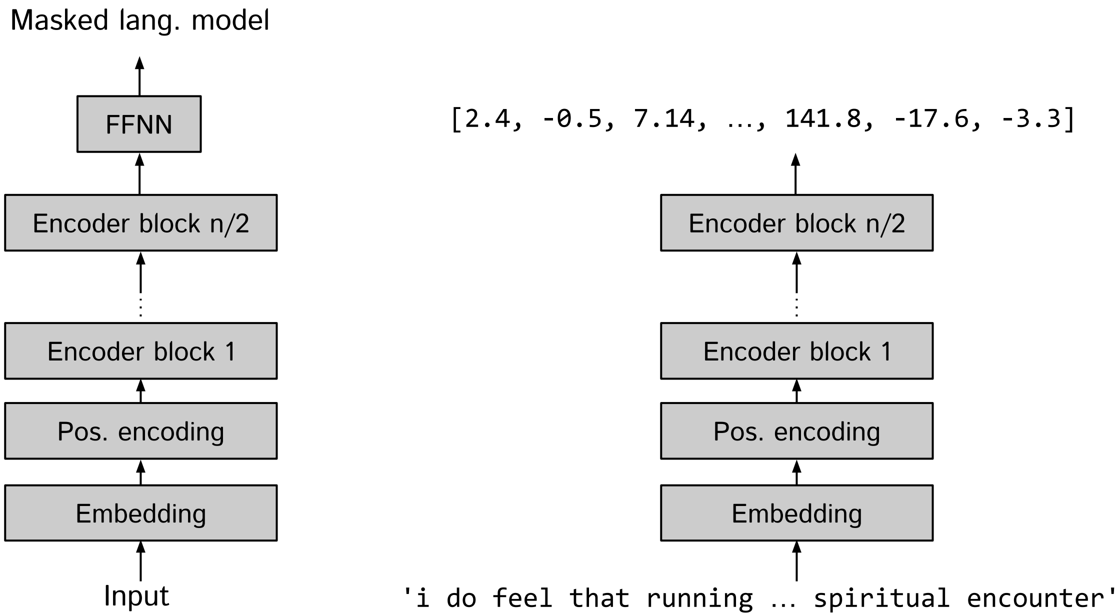
```
'i do feel that running is a divine experience and that i can expect to have some type of spiritual encounter'
```

is tokenized as

```
['[CLS]', 'i', 'do', 'feel', 'that', 'running', 'is', 'a', 'divine', 'experience', 'and', 'that', 'i', 'can', 'expect', 'to', 'have', 'some', 'type', 'of', 'spiritual', 'encounter', '[SEP]']
```

The main idea

Use DistilBERT as a powerful feature extractor

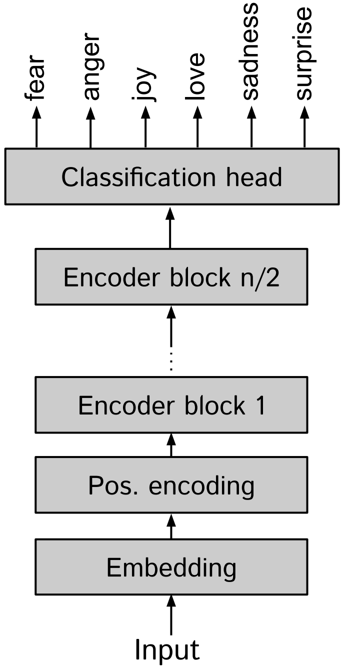


Just remove last FFNN with softmax activation.

You get 768 features.

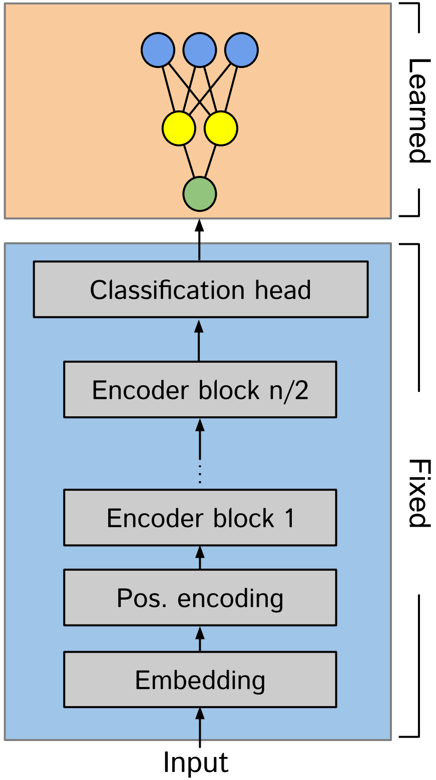
The main idea

- «Attach» a suitable classification head
- Can be based on any model obtained via supervised ML



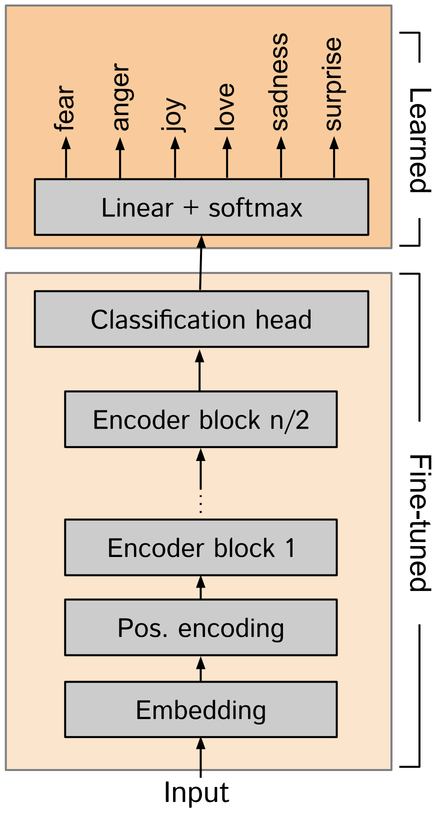
The main idea

- «Attach» a suitable classification head
- Can be based on any model obtained via supervised ML
  - first option: «freeze» the LLM and add a non-differentiable model (e.g., a decision tree)
  - train the model on the extracted features



The main idea

- «Attach» a suitable classification head
- Can be based on any model obtained via supervised ML
  - first option: «freeze» the LLM and add a non-differentiable model (e.g., a decision tree)
  - train this model on the extracted features
  - second option: add a differentiable model (e.g., a dense+softmax layer) and train the whole system starting from
    - random weights for the head
    - existing weights for the rest



First option: «frozen LLM»

```python
from transformers import AutoModel

model_ckpt = 'distilbert-base-uncased'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = AutoModel.from_pretrained(model_ckpt).to(device)

text = 'this is a test'
inputs = tokenizer(text, return_tensors='pt')

inputs = {k:v.to(device) for k,v in inputs.items()}
with torch.no_grad():
    outputs = model(**inputs)
print(outputs.last_hidden_state)
```

tokenized as
`'[CLS] this is a test [SEP]'`

```
tensor([[[-0.1565, -0.1862,  0.0528,  ..., -0.1188,  0.0662,  0.5470],     → [CLS]
         [-0.3575, -0.6484, -0.0618,  ..., -0.3040,  0.3508,  0.5221],     → this
         [-0.2772, -0.4459,  0.1818,  ..., -0.0948, -0.0076,  0.9958],     → is
         [-0.2841, -0.3917,  0.3753,  ..., -0.2151, -0.1173,  1.0526],     → a
         [ 0.2661, -0.5094, -0.3180,  ..., -0.4203,  0.0144, -0.2149],     → test
         [ 0.9441,  0.0112, -0.4714,  ...,  0.1439, -0.7288, -0.1619]]],   → [SEP]
       device='cuda:0')
```

First option: «frozen LLM»

```python
from transformers import AutoModel

model_ckpt = 'distilbert-base-uncased'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = AutoModel.from_pretrained(model_ckpt).to(device)

text = 'this is a test'
inputs = tokenizer(text, return_tensors='pt')

inputs = {k:v.to(device) for k,v in inputs.items()}
with torch.no_grad():
    outputs = model(**inputs)
print(outputs.last_hidden_state)
```

tokenized as
`'[CLS] this is a test [SEP]'`

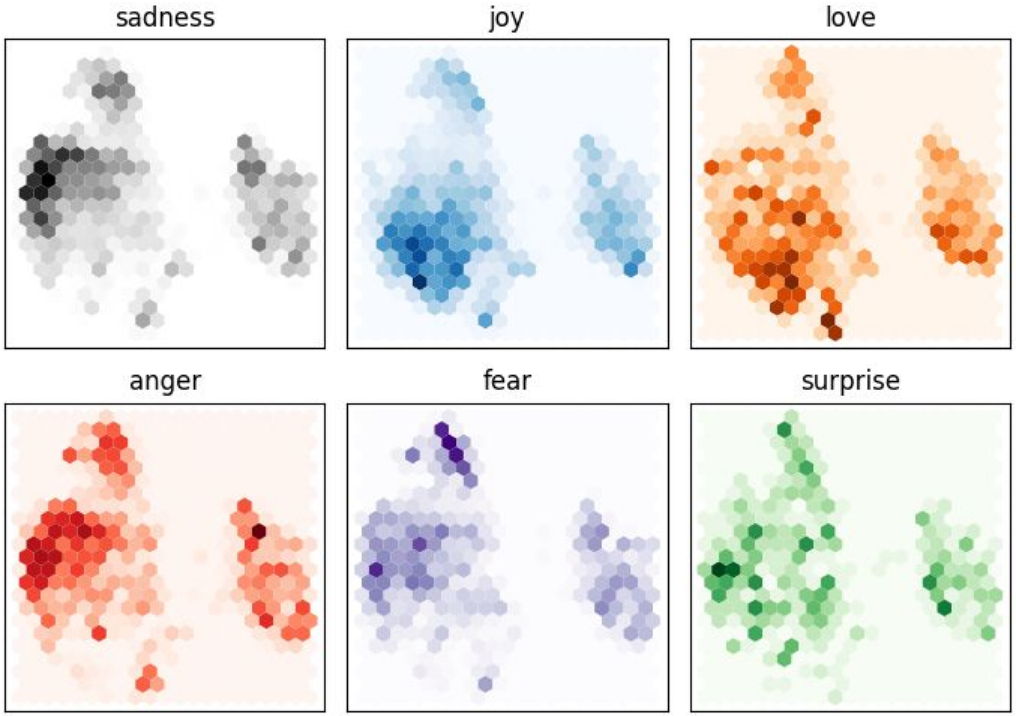
Common choice: retain only hidden state for [CLS] as feature.

```
tensor([[[-0.1565, -0.1862,  0.0528,  ..., -0.1188,  0.0662,  0.5470],     ——> [CLS]
         [-0.3575, -0.6484, -0.0618,  ..., -0.3040,  0.3508,  0.5221],     ——> this
         [-0.2772, -0.4459,  0.1818,  ..., -0.0948, -0.0076,  0.9958],     ——> is
         [-0.2841, -0.3917,  0.3753,  ..., -0.2151, -0.1173,  1.0526],     ——> a
         [ 0.2661, -0.5094, -0.3180,  ..., -0.4203,  0.0144, -0.2149],     ——> test
         [ 0.9441,  0.0112, -0.4714,  ...,  0.1439, -0.7288, -0.1619]]],   ——> [SEP]
       device='cuda:0')
```

From strings to vectors

```
i feel romantic too

i am feeling grouchy

i become overwhelmed and feel defeated
```

```
tensor([...])
tensor([...])
tensor([...])
```

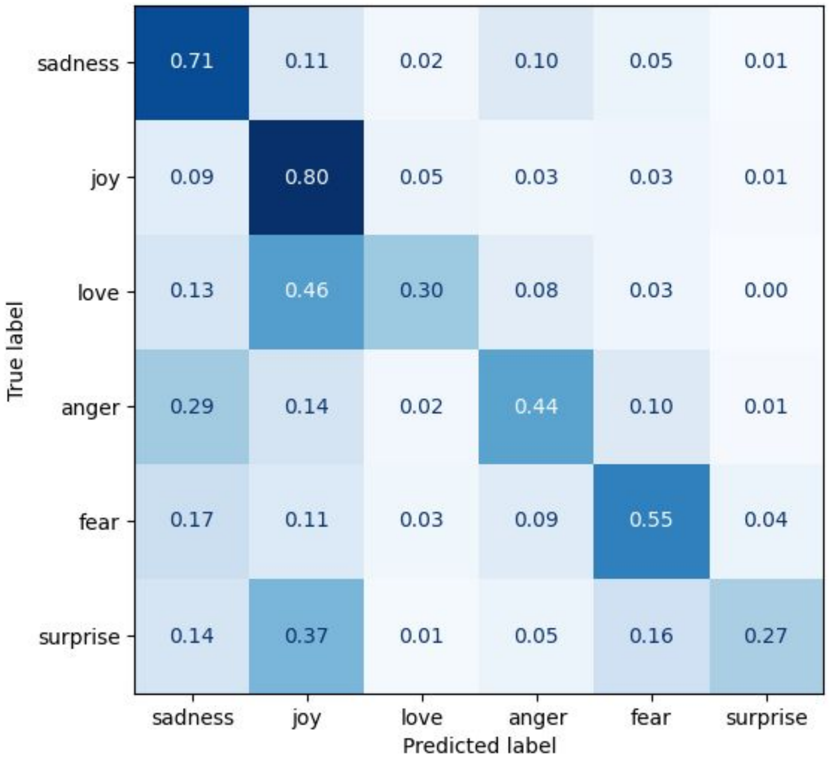


From vectors to models

```
i feel romantic too                         + love
i am feeling grouchy                        + anger
i become overwhelmed and feel defeated      + sadness
```

```
tensor([...])         + love
tensor([...])    X    + anger     y
tensor([...])         + sadness
```

```python
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(X, y)
```



Second option: extend LLM

– No need to manually add a gradient-based classification head
– `AutoModelForSequenceClassification` does that for us

```
num_labels = 6
model = (AutoModelForSequenceClassification
         .from_pretrained(model_ckpt, num_labels=num_labels)
         .to(device))
```

Note: 6 is the number of different labels.

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at
distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight',
'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

The Accelerate library

- Adds an abstraction layer to the training process
- Automatically handles parallel hardware

The Trainer class

– Allows to easily organize all the learning process
– Uses the accelerate library under the hood

```
from transformers import Trainer, TrainingArguments
from sklearn.metrics import accuracy_score, f1_score

batch_size = 64
logging_steps = len(emotions_encoded['train']) // batch_size
model_name = f'{model_ckpt}-finetuned-emotion'
training_args = TrainingArguments(output_dir=model_name, num_train_epochs=2,
                                  learning_rate=2e-5, per_device_train_batch_size=batch_size,
                                  per_device_eval_batch_size=batch_size, weight_decay=0.01,
                                  evaluation_strategy='epoch', disable_tqdm=False,
                                  logging_steps=logging_steps,
                                  log_level='error')
```

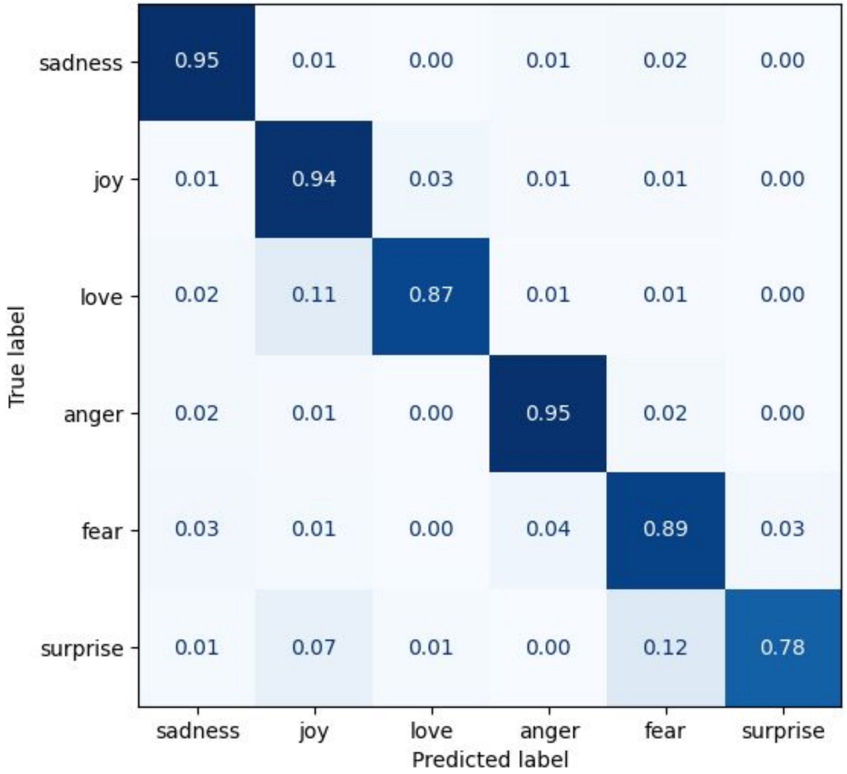
The Trainer class

- Allows to easily organize all the learning process
- Uses the accelerate library under the hood

```
def compute_metrics(pred):
   labels = pred.label_ids
   preds = pred.predictions.argmax(-1)
   f1 = f1_score(labels, preds, average='weighted')
   acc = accuracy_score(labels, preds)
   return {'accuracy': acc, 'f1': f1}

trainer = Trainer(model=model, args=training_args,
                  compute_metrics=compute_metrics,
                  train_dataset=emotions_encoded['train'],
                  eval_dataset=emotions_encoded['validation'],
                  tokenizer=tokenizer)
trainer.train()
```

Second option: extend LLM

Fine-tuning results



AutoModel classes

There are several helper classes allowing the reuse of LLMs:

– `AutoModelForSequenceClassification`
– `AutoModelForSummarization`
– `AutoModelForQuestionAnswering`
– `AutoModelForSeq2SeqLM`
– `AutoModelForMaskedLM`
– and others

LLM for text summarization

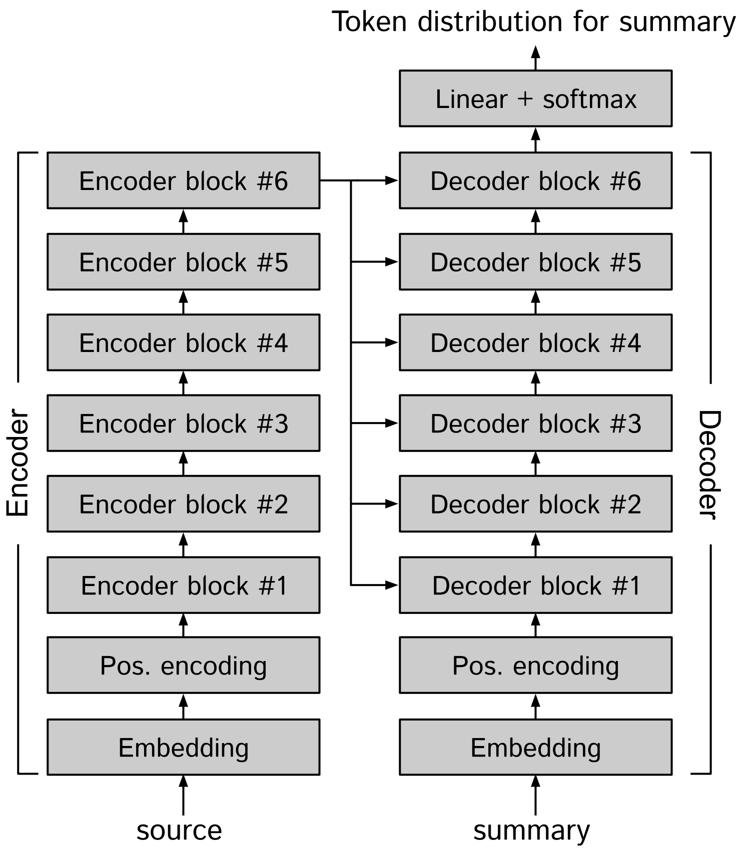
Typically done via encoder-decoder transformers

source

It is reported that on April, 23rd a lunar eclipse will be visible between 10PM and 11PM in the surroundings of Athens.

summary

A lunar eclipse will be visible around Athens during the night of April, 23rd.



Evaluating summaries: BLEU score

Precision-based: fraction of terms in the summary which also appear in the source.

BLEU stands for BiLingual Evaluation Understudy

source	`It is reported that on April, 23rd a lunar eclipse will be visible between 10PM and 11PM in the surroundings of Athens.`
summary	`A lunar eclipse will be visible around Athens during the night of April, 23rd.`

Evaluating summaries: BLEU score

Precision-based: fraction of terms in the summary which also appear in the source.

source	It is reported that on April, 23rd a lunar eclipse will be visible between 10PM and 11PM in the surroundings of Athens.
summary	A lunar eclipse will be visible around Athens during the night of April, 23rd.
precision	9/14

Several issues: synonyms not accounted, false positives, ...

Evaluating summaries: BLEU score

Precision-based: fraction of terms in the summary which also appear in the source.

source	`It is reported that on April, 23rd a lunar eclipse will be visible between 10PM and 11PM in the surroundings of Athens.`
summary	`Athens Athens Athens Athens Athens Athens.`
precision	1

Evaluating summaries: BLEU score

Precision-based: fraction of terms in the summary which also appear in the source.

source	`It is reported that on April, 23rd a lunar eclipse will be visible between 10PM and 11PM in the surroundings of Athens.`
summary	`Athens Athens Athens Athens Athens Athens.`
precision	$1 \rightarrow 1/6$

Only account for a term as many times as it appears in the source.

Evaluating summaries: BLEU score

In general, BLEU score accounts for adjusted precision of all n-grams:

$$p_n = \frac{\sum_{t \in \text{source}} \mathtt{count}(t)}{\sum_{t \in \text{summary}} \mathtt{count}(t)}$$

A penalty term is used to compensate the implicit gain of short summaries:

$$\mathrm{BR} = \min\left(1, \mathrm{e}^{1-\frac{\mathtt{len}(\text{source})}{\mathtt{len}(\text{summary})}}\right)$$

Evaluating summaries: BLEU score

Summing up:

$$\mathrm{BLEU}_N = \mathrm{BR}\left(\prod_{n=1}^{N} p_n\right)^{\frac{1}{N}}$$

```python
from datasets import load_metric

source = 'It is reported that on April, 23rd a lunar eclipse will be visible between'
         ' 10 PM and 11 PM in the surroundings of Athens.'
summary = 'A lunar eclipse will be visible around Athens during the night of April, 23rd.'
bleu = load_metric('sacrebleu', trust_remote_code=True)
bleu.add(prediction=summary, reference=[source])
bleu.compute(smooth_method='floor', smooth_value=0)
```

```
{'score': 18.138480908818533,
 'counts': [12, 6, 4, 2],
 'totals': [16, 15, 14, 13],
 'precisions': [75.0, 40.0, 28.571428571428573, 15.384615384615385],
 'bp': 0.5352614285189903,
 'sys_len': 16,
 'ref_len': 26}
```

BLEU_4

$[p_1, \ldots, p_4]$

BR

Evaluating summaries: ROUGE score

A more complex score also accounting for recall.

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation

```
rouge = load_metric('rouge', trust_remote_code=True)

rouge.add(prediction=summary, reference=[source])
result = rouge.compute()

{k: result[k].mid.fmeasure for k in result}
```

```
{'rouge1': 0.5789473684210527,
 'rouge2': 0.33333333333333337,
 'rougeL': 0.4210526315789474,
 'rougeLsum': 0.4210526315789474}
```

- 'rouge1' → unigram-based
- 'rouge2' → bigram-based
- 'rougeL' → longest common subsequence
- 'rougeLsum' → longest common subsequence (source split on newlines)

Materials

[1] A. Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. 2015. https://karpathy.github.io/2015/05/21/rnn-effectiveness/
[2] The Gradient. Evaluation Metrics for Language Modeling. 2019. https://thegradient.pub/understanding-evaluation-metrics-for-language-models/
[3] N. Keskar et al. CTRL: A Conditional Transformer Language Model for Controllable Generation. 2019. https://arxiv.org/abs/1909.05858
[4] A. Madani et al. ProGen: Language Modeling for Protein Generation. 2020. https://arxiv.org/abs/2004.03497

The lab

- Generate text using RNNs.
- Compute the perplexity of sequences generated via LLMs.
- Wrangle data from a NLP dataset for emotion detection.
- Solve the emotion detection problem using
  - features extracted via a LLM,
  - fine-tuning a LLM with an attached classification head.
- Compute the BLUE and ROUGE scores of summaries.

Thanks!

dario.malchiodi@unimi.it https://malchiodi.di.unimi.it

Assets:

– Google fonts and Material icons, https://fonts.google.com
– Font awesome, https://fontawesome.com