Assume a sequence of L words (and a batch size fof $N = 1$ for simplified understanding).

1 Question 1: Role of attention and positional encoding

Attention square mask

The square mask has 2 purposes:

- **Causality of the prediction**: The mask serves to mask the future tokens. So the current predicted word cannot see what's coming next. defined as a square matrix.
- **Performances** Save some computation time during training
  - The attention scores are not computed for one word at a time. We directly let the whole sequence vector attend to the whole sequence vector, resulting in a (L, L) attention score matrix.
  - *Note: otherwise you'd have to loop L times for all the words, assume you're treating word l, you have to compute an attention vector of size l to the previous word. But tensor operations are not friendly with varying shapes. So better do a bit of extra attention computation and be able to process a whole sequence at once (actually $\frac{L*(L-1)}{2}$) useless attention scalars are computed.*
  - In the end, you compute a (L, L) attention score and pytorch adds the mask.
  - Adding $-\infty$ to the score will result in a weight of 0 after the softmax operations. Basically, we do not take the future words into account when multiplying with the values.
  - Adding 0 to the score will leave the original weights untouched.

```
Attention mask for a sequence of 5 tokens.
10 useless computations for a sequence of 5 tokens
   [[0., -inf, -inf, -inf, -inf],
    [0.,   0., -inf, -inf, -inf],
    [0.,   0.,   0., -inf, -inf],
    [0.,   0.,   0.,   0., -inf],
    [0.,   0.,   0.,   0.,   0.]]
```

Positional encoding

The role of positional encoding is to provide the order of the words in the sentence to the attention mechanism.

- Observation 1: *Order in a sequence matters: "Le chat chasse la souris" has a totally different meaning that "La souris chasse le chat".*
- Observation 2: *Unlike recurrent neural networks (RNNs) like the GRU-based architecture we used in lab 1 for translation, the Transformer [2] architecture [2] doesn't process source sentences in a step-by-step manner. Instead, it processes all tokens in the sequence simultaneously. This parallel processing capability gives Transformer [2]s their efficiency but also means that they lack an inherent sense of position or order in a sequence.*
- To solve the issue mentioned above, positional encoding vector is added to the embeddings of tokens before they're fed into the Transformer [2]. The positional encoding is a set of values that's designed to give the model information about the position of each token within the sequence. The goal is to ensure that the resulting combined embeddings (token embedding + positional encoding) will be unique for each position, even if the token is the same.
- This also lets the Transformer [2] architecture (in its internal computations) behave differently depending on the "distance" between words.

2 Question 2: Classification head

Language modeling

During pre-training, the Transformer [2] model transforms each token into a representative feature vector supposed to model the next word.
The classification head used for this purpose maps that vector into a logit/probability of getting the next word. This allows to train on a corpus of un-annotated sentences. If you want to generate new text or complete a sequence, you can use this classifier.
But **if you want to do any other downstream task** (e.g sentiment classification or spam classification) **the original classifier becomes useless.**

Downstream task(like classification)

Instead, you can replace the original classification head dedicated to language modeling by a new classification head for a specific downstream task.
To fine-tune, we can basilly freeze the Transformer [2] pretrained weigths and leverage the fact that the representation outputs of the Transformer [2] will model the tokens in a very sensitive feature space. A powerful feature space which models many inherent things in the sentence. Therefore, with a few annotated data samples for your downstream classification task (e.g Google maps comments on restaurants annotated with their star ratings, spam/not spam examples), you can simply retrain a new classifier head. *Freezing the Transformer [2] weights is not mandatory, you could also fine tune the Transformer [2] layers... but it seems a bit risky in my opinion if there's not much data to perform the downstream task training.*

3 Question 3: Trainable parameters

Notations

- V is the vocabulary size
- D is the hidden feature dimension which is also equal to the embedding dimension: *hdim* $=$ *embeddingDim* for simplifications.
- M is the number of attention layers.
- A is the number of attention heads (denoted h in [2]).
- C is the number of classes.

Parameters computation

$$
\begin{aligned}
\text{Total number of parameters} = & \\
& \underbrace{V.D}_{\text{embedding look up table}} \\
& + M * \Big[\underbrace{\overbrace{3}^{\text{Key,Query,Value}} * (A * (\overbrace{D * \frac{D}{A}}^{\text{weights}} + \overbrace{\frac{D}{A}}^{\text{bias}}))}_{\text{Linear embeddings projection}} + \underbrace{(\overbrace{D^2}^{\text{weights}} + \overbrace{D}^{\text{bias}})}_{\text{Weighted values linear projection after concatenation}} \\
& + \underbrace{2 * (\overbrace{D^2}^{\text{weights}} + \overbrace{D}^{\text{bias}})}_{\text{2 chained feedforward netwoks}} + \underbrace{2 * (\overbrace{D}^{\text{slope}} + \overbrace{D}^{\text{offset}})}_{\text{2 layer normalizations}} \Big] \\
& + \underbrace{\overbrace{D * C}^{\text{weights}} + \overbrace{C}^{\text{bias}}}_{\text{classifier head}} \\
& = V.D + M * \left[6D^2 + 10D \right] + C(D+1)
\end{aligned}
\tag{1}
$$

We're able to cross check:

- the number of parameters returned by the pytorch *numel* function
- the analytical computation using *sympy* for symbolic computation in python.

Let's check a simple example with a single layer ($M = 1$).

Modules	Parameters	Analytic validation	Analytic formula
base.encoder.weight	10000200	10000200	D*V
base.transformer_encoder.layers.0.self_attn.in_proj_weight	120000	120000	3*D²
base.transformer_encoder.layers.0.self_attn.in_proj_bias	600	600	3*D
base.transformer_encoder.layers.0.self_attn.out_proj.weight	40000	40000	D²
base.transformer_encoder.layers.0.self_attn.out_proj.bias	200	200	D
base.transformer_encoder.layers.0.linear1.weight	40000	40000	D²
base.transformer_encoder.layers.0.linear1.bias	200	200	D
base.transformer_encoder.layers.0.linear2.weight	40000	40000	D²
base.transformer_encoder.layers.0.linear2.bias	200	200	D
base.transformer_encoder.layers.0.norm1.weight	200	200	D
base.transformer_encoder.layers.0.norm1.bias	200	200	D
base.transformer_encoder.layers.0.norm2.weight	200	200	D
base.transformer_encoder.layers.0.norm2.bias	200	200	D
classifier.decoder.weight	10000200	10000200	C*D
classifier.decoder.bias	50001	50001	C

Language modeling vs classification

Let's take an example

- $V = 50001$
- $D = 200$ hidden feature dimension or embeddings dimension
- $M = 4$ is the number of attention layers.
- $A = 2$ is the number of attention heads (but it does not affect parameters compution)

We have 2 configurations:

- Language modeling: C classes $= V$ vocabulary size
- Sentiment classification: C classes $= 2$ (good/bad)
- Here we'll assume that for the classification fine tuning, the transformer backbone (embeddings+ transformer layers) is totally frozen and that we only train the classifier's weights. This is a very lightweight fine tuning.

Number of trainable parameter	Language modeling task $C = V$	Classification task $C = 2$
Word embeddings	Trainable $V.D$	0 trainable parameters = Frozen
M Transformer layers with A heads	Trainable $M * \left[6D^2 + 10D\right]$	0 trainable parameters = Frozen
Classifier head	Trainable $V(D+1)$	Trainable $2(D+1)$
Total (analytics)	$V.D + M * \left[6D^2 + 10D\right] + C(D+1)$	$2(D+1)$
Total parameter	21M parameters (21018401)	402 parameters

Remark on the number of attention heads

Please note that due to the tricky implementation of torch, A does not appear in the final computation (chunk Key K, Query Q and Value V into D chunks of size $\frac{D}{A}$). This allows keeping the same computation budget with various attention heads. This explicitly appears in the pytorch source code of MultiheadAttention

```
self.head_dim = embed_dim // num_heads
```

Task 3

```
<sos> Comment montrer que je suis autonome .
Comment montrer que je suis autonome . <eos>
```

Training logs. Seems like everything is going fine. On a Nvidia T500, there is a memory limitation, we have to limit the sentence length to avoid out of memory issues.

- $batch_{size} = 16$
- $max_{len} = 64$

```
| epoch   1 |   500/ 3125 steps | loss 7.29929 | ppl 1479.254
| epoch   1 |  1000/ 3125 steps | loss 6.47710 | ppl  650.081
| epoch   1 |  1500/ 3125 steps | loss 6.19202 | ppl  488.834
| epoch   1 |  2000/ 3125 steps | loss 6.05407 | ppl  425.842
| epoch   1 |  2500/ 3125 steps | loss 5.93240 | ppl  377.057
| epoch   1 |  3000/ 3125 steps | loss 5.82976 | ppl  340.279
| epoch   2 |   500/ 3125 steps | loss 5.50992 | ppl  247.131
| epoch   2 |  1000/ 3125 steps | loss 5.46325 | ppl  235.864
| epoch   2 |  1500/ 3125 steps | loss 5.42948 | ppl  228.032
| epoch   2 |  2000/ 3125 steps | loss 5.40475 | ppl  222.461
| epoch   2 |  2500/ 3125 steps | loss 5.37446 | ppl  215.824
| epoch   2 |  3000/ 3125 steps | loss 5.35723 | ppl  212.136
```

Here are the prediction results compared to the provided pretrained model.

```
Epoch 2:
--------
Bonjour les plus tard, il a été mis à la fin de la région de la Commission,
mais aussi de la loi de la région de la Commission

Pretrained model:
-----------------
Bonjour les gens qui ont été très accueillants et sympathiques.
```

4 Question 4: Results interpretation

We train on 1600 sentences for the downstream task of sentiment analyzis, which is a classification task (good/bad review). Some nitty gritty details on the implementation:

- ”Taking the last word feature vector” does not mean take the last element of the tensor output of the model. For each batch element, you need to retrieve the last word before padding.
- I removed the mask to let each word access the whole sentence... there may be a tiny discrepancy with regard to the original Language modeling task as the pretrained transformer may not be used to see some words ”in the future”. I tried both solutions by the way, picked what felt right... it didn’t change the training curves much by the way.

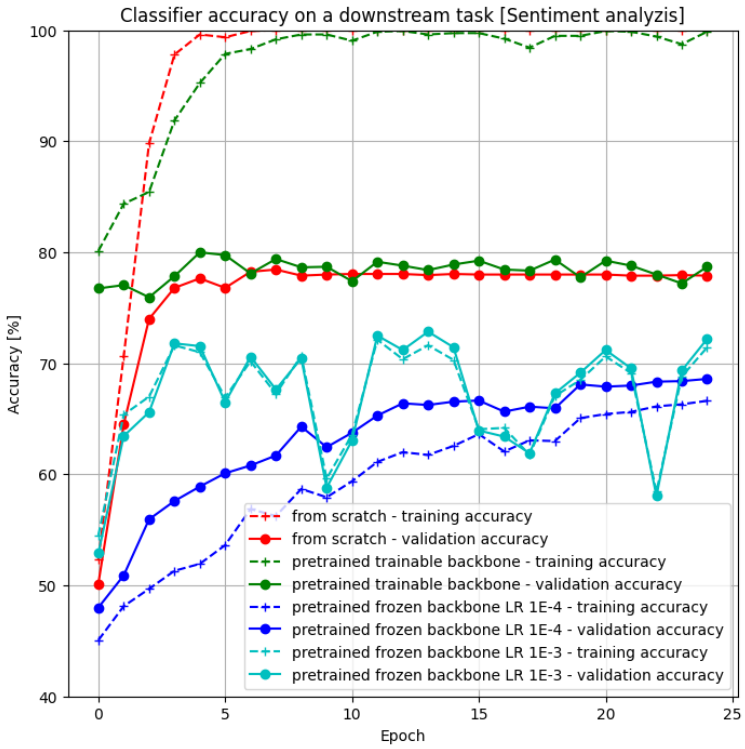

Figure 1: Training and validation accuracies for different setups (start from scratch, pretrained model, frozen pretrained model to only train the lightweight classifier)

From 1, we can see that starting the classification training from a pretrained transformer model on the language modeling task works gets much better accuracy in a single epoch than training a model initalized randomly. $76\% >> 65\%$ after 2 epoch. This is quite incredible and This can be explained by the fact that the feature vectors learnt on the language modeling task are very versatile and definitely contain a lot of semantic information. We see that if we’d taken the best fine tuned pretrained model at epoch=5, we get 80% accuracy when pretraining from scratch goes to 78% (slower but still an remarkable performance). These training curves also show that in a few epochs, we’re able to memorize the whole training dataset (training accuracy goes to 100%, loss goes to zero) as shown by the dashed red and green curves. With 20M parameters, it’s not a surprise. These classifiers may lack a bit of the skill to generalize. They may memorize some easy ”words” or

patterns to recognize whether or not a review is positive or not. I also tried to freeze the transformer weights (blue and cyan curves), this training goes faster as there are only a few parameters to train in the classifier head (see Question 3 with the assumption I made for the fine-tuning stage)... but the accuracy is not as great. This still avoids the memorization of the training set.
Let's take a look at some results using the fine tuned model based on the pretrained Transformer. Finally

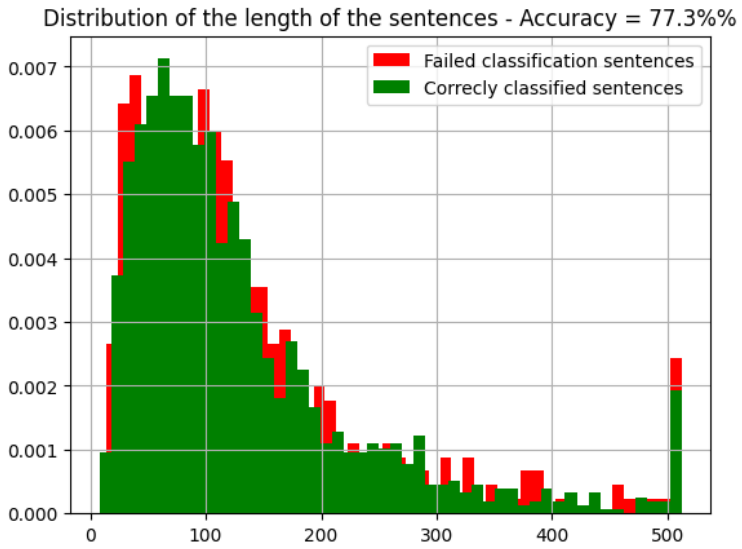

Figure 2: Distribution of the length of the sentences when classifier fails or succeed... there's no obvious gap here.

in 2, we show that there's no easy analyzis to make like "the model is unable to classify long sentences"... as basically the distribution of sentences length looks the same whether or not they're classified correctly or wrongly.
Some samples of fail below to see some concrete examples where the model failed. These are not so easy as you can see.

```
GROUNDTRUTH=NEGATIVE -> PREDICTION= POSITIVE
je ne recommande absoluement pas ce torchon , véritable offense
à la mémoire de Michael Jackson . Il repose enfin en paix ...
```

"repose enfin en paix" may still be difficult to classify as it's not a "bad" sentiment but a respectful sentence. "torchon" and "offense" are strong words where a model shall not fail and consider it as negative feelings.

```
GROUNDTRUTH=POSITIVE -> PREDICTION= NEGATIVE #
ce livres et une vrai drogue ! la derniere fois je n' arrivais meme pas a m' arreter
! A LIRE ABSOLUMENT !!!! !!!! !!!! !!!
```

"Drogue" may have biased the result toward the negative prediction.

5 Question 5: Language modeling limitations

- The proposed approach to language modeling is next word prediction which implies causality. It is also known as autoregressive language modeling. Example: "Salut les copains, comment ???"
- BERT [1] introduces a different training objective which is masked language modeling. Example: "Salut les copains, ??? allez vous?"

In the causal LM objective, each token can only attend to previous tokens in the sequence, it's unidirectional. This is because the model is designed to predict the next token in a sequence. While this setup is great for generative tasks, it doesn't always capture the context from both sides (before and after) of a word or phrase, which can be beneficial for understanding its meaning in certain situations.
A comment might begin negatively but end positively or vice versa. Being able to see and consider the entire sentence at once can help in better grasping its overall sentiment. Example: "L'ensemble du livre est médiocre. Mais les dix premières pages m'ont vendu tellement de rêve que je l'ai lu jusqu'au bout." The classifier based on the autoregressive language model may simply forget the begining of the sentence and classify the review as a "good review". The BERT model will actually have to focus on both the negative and positive sentences to actually understand the "Mais" word.

Note

For the specific downstream task considered in this notebook, sentiment classification, it didn't seem straigthforward at first that we'd need the bidirectional advantage of BERT to simply grasp whether or not the recommendation was good or not. It doesn't seem like a very complete. BERT[1] shines on much more complex reasoning benchmarks examples mentioned in the GLUE [3] benchmark like the Winograd canonical example:
Who does the pronoun **They** refer to in the 2 following sentences?.

- **The city councilmen** refused the demonstrators a permit because **they** feared violence.
- The city councilmen refused **the demonstrators** a permit because **they** advocated violence.

Answering this question definitely requires the context before and after the word **they** and seems much more complex than feeling what's negative or positive in a review.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[3] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.