Cross-lingual Language Model Pretraining

Hussain Md. Safwan
Roll: SH-042

Raihan Dewon Eman
Roll: SH-048

February 23, 2021

Contents

1 Introduction

Cross-lingual language model pretraining paper was published by facebook researchers in 2019. Cross-lingual language model(XLM) is a machine learning model representing relations between words in different languages.

1.1 Motivation for XLM

- **One model to learn them all**
  This means that there's a model for all of the languages. The advantages of this model is that there's no need for a new model for each language pair that we want to translate between. A con of this could potentially be that we need a larger model overall than a separate language pair model. This could lead to either higher inference time or more computational requirement for training or inferencing.

- **Zero-shot or few-shot translation**
  We will discuss later a cross-lingual model allows us to have high quality translations for low or no resource languages. Cross-lingual language model will allow us to not only translate between languages with very little data especially for translation, but also model the language better itself.

1.2 Why is pretraining important?

Pretraining is important because it allows us to train a model with a lot of data and to transfer that learning either through transfer learning or fine tuning to other downstream tasks. So, if we pretrain a language model, we can use that language model for named entity recognition, part of speech tagging or text classification. So, pre-raining models are very useful for a variety of purposes.

1.3 Tokenization

In order to get our computer to understand any text, we need to break that word down in a way that our machine can understand. That's where the concept of tokenization in Natural Language Processing (NLP) comes in. Tokenization is not just about breaking down the text. It plays a significant

role in dealing with text data. It is often a very important part of the pretraining.
Tokenization is a way of separating a piece of text into smaller units called tokens. Tokenization is performed on the corpus to obtain tokens. The tokens are then used to prepare a vocabulary. Vocabulary refers to the set of unique tokens in the corpus. Here, tokens can be either words, characters, or subwords. Hence, **tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization**.

- **Word Tokenization:**
  Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed. Pretrained Word Embeddings such as **Word2Vec** and **GloVe** comes under word tokenization.
  Drawbacks of Word Tokenization
  One of the major issues with word tokens is dealing with **Out Of Vocabulary (OOV) words**. OOV words refer to the new words which are encountered at testing. These new words do not exist in the vocabulary.

- **Character Tokenization:**
  Character Tokenization splits a piece of text into a set of characters. It overcomes the drawbacks we saw above about Word Tokenization.

  – Character Tokenizers handles OOV words coherently by preserving the information of the word. It breaks down the OOV word into characters and represents the word in terms of these characters

  – It also limits the size of the vocabulary. Want to talk a guess on the size of the vocabulary? 26 since the vocabulary contains a unique set of characters

  Drawbacks of Character Tokenization Character tokens solve the OOV problem but the length of the input and output sentences increases rapidly as we are representing a sentence as a sequence of char-

acters. As a result, it becomes challenging to learn the relationship between the characters to form meaningful words.

- **Subword Tokenization:**
  Subword Tokenization splits the piece of text into subwords (or n-gram characters). For example, words like lower can be segmented as low-er, smartest as smart-est, and so on.
  Transformer based models rely on Subword Tokenization algorithms for preparing vocabulary. One of the most popular Subword Tokenization algorithm is known as Byte Pair Encoding (BPE).

1.4 Sub-Word Tokenization

- Advantages
  - Out of vocabulary words
  - Accounts for internal structure of words
  - Single vocabulary for all languages
- Popular implementations
  - BPE
  - WordPiece
  - SentencePiece
- Applications
  - BPE - XLM, GPT, RoBERTa
  - WordPIece - BERT
  - SPM – mBART, XLM-R

1.5 Byte Pair Encoding(BPE)

Byte Pair Encoding (BPE) is a widely used tokenization method among transformer-based models. BPE addresses the issues of Word and Character Tokenizers:

- BPE tackles Out Of Vocabulary(OOV) effectively. It segments OOV as subwords and represents the word in terms of these subwords

- The length of input and output sentences after BPE are shorter compared to character tokenization

BPE is a word segmentation algorithm that merges the most frequently occurring character or character sequences iteratively. Here is a step by step guide to learn BPE.

1.5.1 Steps to learn BPE

- Split the words in the corpus into characters after appending ¡/w¿
- Initialize the vocabulary with unique characters in the corpus
- Compute the frequency of a pair of characters or character sequences in corpus
- Merge the most frequent pair in corpus
- Save the best pair to the vocabulary
- Repeat steps 3 to 5 for a certain number of iterations

We will understand the steps with an example. Consider a corpus like below:

Corpus		
low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

1a) Append the end of the word (say <\w>) symbol to every word in the corpus:

Corpus		
low<\w>	lower<\w>	newest<\w>
low<\w>	lower<\w>	newest<\w>
low<\w>	widest<\w>	newest<\w>
low<\w>	widest<\w>	newest<\w>
low<\w>	widest<\w>	newest<\w>

1b) Tokenize words in a corpus into characters:

Corpus		
l o w<\w>	l o w e r<\w>	n e w e s t<\w>
l o w<\w>	l o w e r<\w>	n e w e s t<\w>
l o w<\w>	w i d e s t<\w>	n e w e s t<\w>
l o w<\w>	w i d e s t<\w>	n e w e s t<\w>
l o w<\w>	w i d e s t<\w>	n e w e s t<\w>

2. Initialize the vocabulary:

Vocabulary								
d	e	i	l	n	o	s	t	w

Iteration 1:

3. Compute frequency:

Frequency		
d-e (3)	l-o (7)	t-<\w> (8)
e-r (2)	n-e (5)	w-<\w> (5)
e-s (8)	o-w (7)	w-e (7)
e-w (5)	r-<\w> (2)	w-i (3)
i-d (3)	s-t (8)	

4. Merge the most frequent pair:

Corpus		
l o w<\w>	l o w e r<\w>	n e w es t<\w>
l o w<\w>	l o w e r<\w>	n e w es t<\w>
l o w<\w>	w i d es t<\w>	n e w es t<\w>
l o w<\w>	w i d es t<\w>	n e w es t<\w>
l o w<\w>	w i d es t<\w>	n e w es t<\w>

5. Save the best pair:

Vocabulary								
d	e	i	l	n	o	s	t	w
es								

Repeat steps 3-5 for every iteration from now. Let me illustrate for one more iteration.

Iteration 2:

3. Compute frequency:

Frequency		
d-es (3)	l-o (7)	t-<\w> (8)
e-r (2)	n-e (5)	w-<\w> (5)
es-t (8)	o-w (7)	w-es (5)
e-w (5)	r-<\w> (2)	w-i (3)
i-d (3)	w-e (2)	

4. Merge the most frequent pair:

Corpus		
l o w<\w>	l o w e r<\w>	n e w est<\w>
l o w<\w>	l o w e r<\w>	n e w est<\w>
l o w<\w>	w i d est<\w>	n e w est<\w>
l o w<\w>	w i d est<\w>	n e w est<\w>
l o w<\w>	w i d est<\w>	n e w est<\w>

5. Save the best pair:

Vocabulary								
d	e	i	l	n	o	s	t	w
es	est							

After 10 iterations, BPE merge operations looks like:

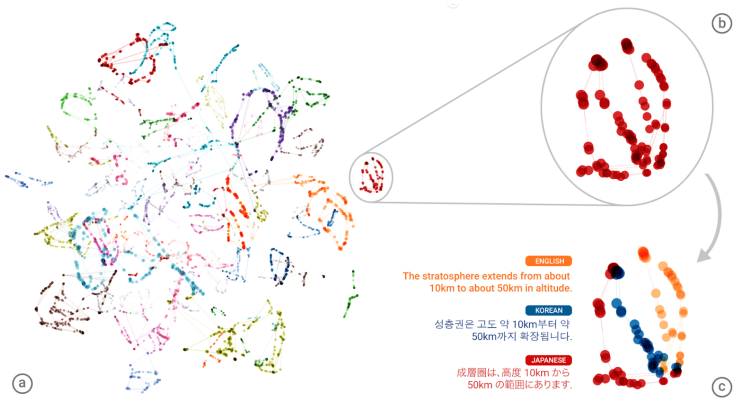
Vocabulary								
d	e	i	l	n	o	s	t	w
es	est	est<\w>	lo	low	low<\w>	ne	new	newest<\w>

Applying BPE to OOV words Here is a step by step procedure for representing OOV words:

- Split the OOV word into characters after appending <\w>
- Compute pair of character or character sequences in a word
- Select the pairs present in the learned operations
- Merge the most frequent pair
- Repeat steps 2 and 3 until merging is possible

Multiple language can be tokenized, if we use unicode characters.

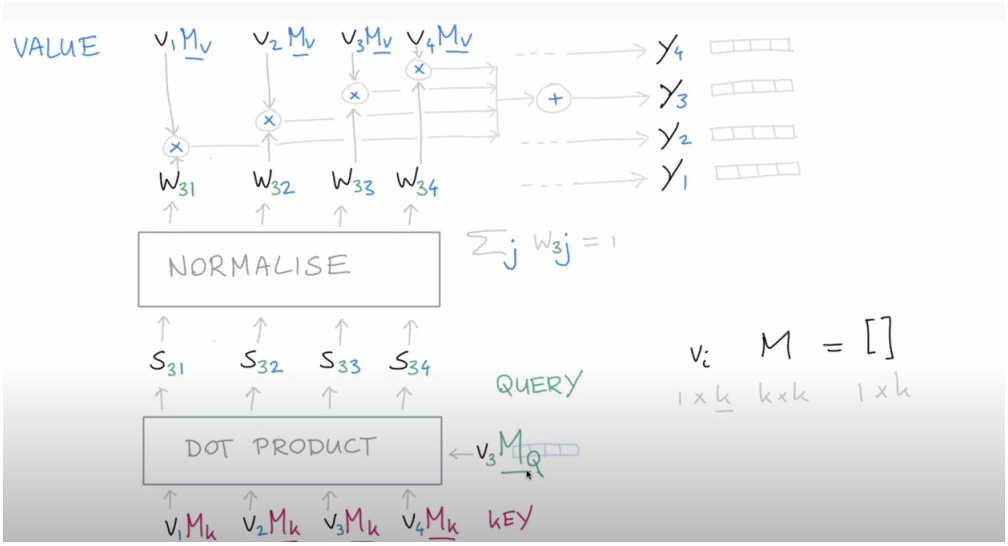
1.6 Zero-Shot Translation



Zero-shot translation refers to the fact that we can translate between two languages even though we don't have training parallel data between two languages to train the system on. The example above is published by google in 2017. Using LSTM sequence to sequence models the machine translation system was trained on English to Korean data and English to Japanese data. It is demonstrated that same system can also translate between Korean and Japanese even though it isn't trained on Korean and Japanese language translation. So, zero shot refers to there was zero data for Korean and Japanese and it can still translate between the two.

2 Attention Based Architectures

Attention is the cognitive process of focusing on a certain portion of data while blurring the others. Say we have a sentence ***Bank of the river***. Its obvious from our intuition that the word **bank** here refers to the one related to water and not the one to money. It is clear that it is desireable for the model to focus on the word **river** while processing the word **bank** to be aware of the context. This is where the idea of attention comes in. We want the model to ***attend*** to the word **river**. A fair question is how is that done?



1. All the words are embedded to the corresponding vector. Let V be space that contains all the embeddings.

2. For each word vector $v_i \epsilon V$, perform an inner product with each of the vector $v_j \epsilon V$. Let the inner product produces a value of $S_{ij} \epsilon S$, where S is a matrix.

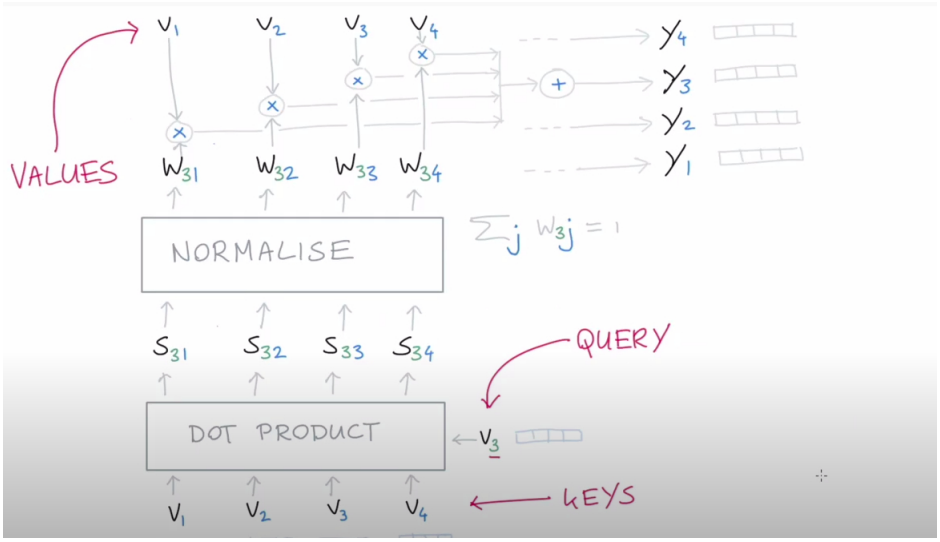
3. Normalize S along each row, such that for row 3, for instance, $\sum_j W_{3j} = 0$. This produces the weight matrix W.

4. Finally to compute y_i, multiply $v_i \epsilon V$ with W_{ij} and sum the resulting

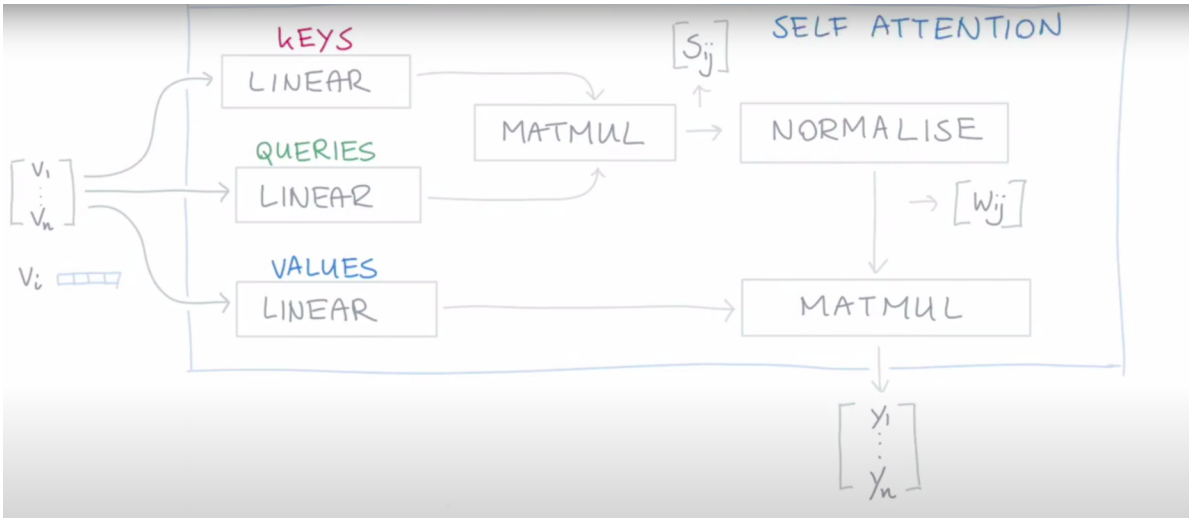
vectors, that is perform the following iteration,

$$y_i = \sum_{j=1}^{i} W_{ij} v_i$$

where y_{ij} for $1 \leq j \leq i$ is the amount of attention token v_i needs to pay to token v_j.



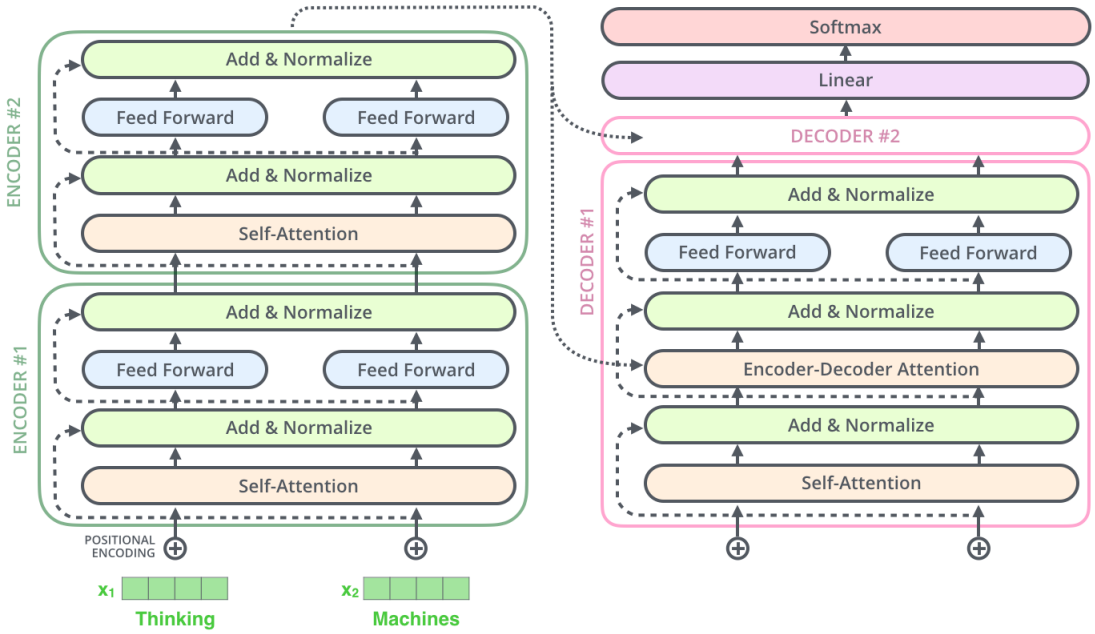
That's the raw concept of how attention is computed. This is known as ***Self Attention***. Note that we're using same values thrice for each iteration. Drawing a database analogy here, the above event can be phrased as the model querying for the attention scores for token v_i, the query is mapped to the set of keys $v_i \epsilon V$ that finally produces a value of weighted sum of $v_i \epsilon V$. Hence the terms ***Query, Key, Value*** matrices. The reason such matrices are needed is that during back propagation they are updated which results in learning. That is illustrated in the following figure,



The kind of attention we've been talking about so far is Self Attention. There is another kind of attention called ***Encoder-Decoder Attention***. Here the query is produced by the decoder while processing tokens of the target language as we'll see later.

2.1 Transformer

Transformer is a neural network architecture that is based on the concept of attention and aimed to replace the sequential models such as RNNs and LSTMs. The idea of Transformer quickly grabbed the attention thanks to the parallel nature of its token encoding. In a transformer the signal need not travel through all the encoders then through the decoders to reach the desired decoder in the series. All the token are concurrently fed into the network via a parallel pipeline. The working strategy of the transformer is explained as follows,



2.1.1 Encoder

The encoder block has the following components

1. **Positional Encoding:** Since the BERT model processes each token in parallel, it can't keep track of position of the tokens within the source sentence. Positional encoding pushes the model to care more about the position. Its like adding a tag to each token that tells the model about its position. One of the popular choice for such encoding is using trigonometric functions, pair of sine and cosine functions.

$$p_t^{(i)} = \begin{cases} sin(\omega_k.t); \ if \ i = 2k \\ cos(\omega_k.t); \ if \ i = 2k+1 \end{cases} \qquad w_k = \frac{1}{10000^{\frac{2k}{d}}}$$

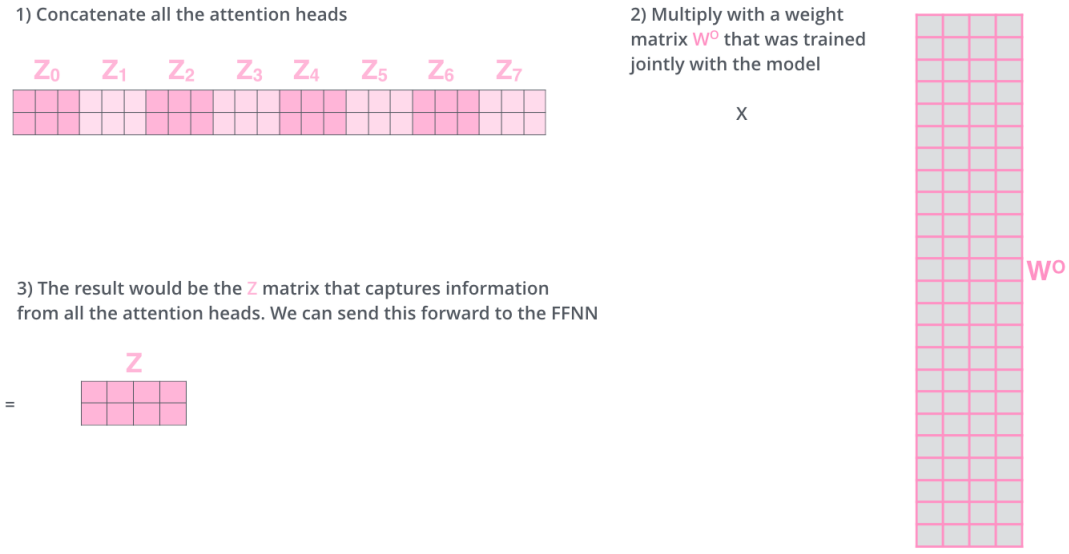
where, i = index with vector, p = position of token and d is the length of the vector.
The input that goes into the attention block is,

$$V = word\ vector + positional\ encoding + sentence\ id$$

2. **Multi Head Attention:** We've seen how attention mechanism works. Now this current model can handle considerable amount of context, but one problem that arises is that it can attend to one token at a time since token with the greatest value attached is attended to. Take the following sentence, for instance,

I spoke to the teacher last afternoon

with **I** as the focus. The model is expected to attend words such as **speak, teacher, afternoon** all at once. To solve this, we introduce the concept of multi head attention, where each head is devoted to a focal word. Again when computing the self attention, it is obvious that the word **I** or any other word will assign the maximum significance to itself, which is redundant. To get rid of both the flaws Multihead attention models are introduced. Multihead attention models feed the output of each previous single head attention block into the next block, each block has its own triplet of key, query and value matrices that are updated during back propagation. Output of every layers are than concatenated into a single matrix of dimension lxh, l = length of input and h = number of heads.



In the original paper, the authors propose an 8 headed architecture.

Whats so special about the number 8? Well that's an arbitrary number. The question that however demands an answer is ***does more heads mean better results?*** It is true that the more the number of heads, the more patterns are learnt since each head operate independently. But high er number of heads also indicate rise in training time and wastage of resources as most of the heads sit idle sometimes.

Base accuracy	83.59%											
Delta accuracy on MNLI dev set when all but one head are masked												
Layer \ head	1	2	3	4	5	6	7	8	9	10	11	12
1	-0.43%	-0.41%	-0.19%	-0.26%	-0.46%	-0.23%	-0.47%	-0.18%	-0.01%	-0.28%	-0.30%	-0.28%
2	0.07%	0.07%	-0.05%	-0.05%	-0.02%	-0.06%	-0.01%	0.03%	0.10%	-0.05%	0.03%	-0.46%
3	-0.18%	-0.27%	-0.26%	-0.21%	-0.28%	-0.38%	-0.36%	-0.32%	-0.19%	-0.15%	-0.14%	-0.39%
4	-0.53%	-1.22%	-1.24%	-1.12%	-1.38%	-1.35%	-1.27%	-1.46%	-1.18%	-1.27%	-1.29%	-1.36%
5	-0.57%	-0.92%	-0.97%	-0.29%	-1.09%	-1.03%	-1.02%	-1.05%	-1.06%	-1.22%	-1.11%	-0.94%
6	-0.61%	-1.03%	-0.93%	-0.80%	-1.43%	-0.87%	-1.01%	-1.06%	-0.52%	-0.86%	-0.88%	-1.06%
7	0.05%	-1.49%	-1.43%	-1.67%	-1.43%	-1.71%	-1.60%	-1.77%	-1.17%	-1.50%	-1.47%	-1.53%
8	-0.72%	-0.89%	-1.08%	-1.35%	-1.07%	-1.11%	-1.21%	-1.09%	-0.95%	-0.85%	-1.21%	-1.16%
9	-2.04%	-1.04%	-2.13%	-1.80%	-0.96%	-2.17%	-2.00%	-1.65%	-1.55%	-1.97%	-2.09%	-2.13%
10	-0.12%	-0.42%	-0.26%	-0.44%	-0.40%	-0.79%	-0.35%	-0.44%	0.07%	-0.29%	-0.48%	-1.31%
11	-0.84%	-1.15%	-1.01%	-1.21%	-1.12%	-1.13%	-1.06%	-1.22%	-1.04%	-1.22%	-0.56%	-0.19%
12	-0.29%	-0.35%	-0.50%	-0.38%	-0.12%	-0.35%	-0.34%	-0.42%	-0.25%	-0.15%	-0.25%	-0.38%

As can be seen in the figure, how each of the head show a standalone performance. It is seen that one head can alone account for 12 heads without any drastic fall in accuracy and more surprisingly in some cases (layers 2, 7, 10) the performance enhances after removing the heads.

3. **Addition and Normalization:** In the encoder, there is a skip connection that goes directly from the input bypassing the attention block and is added to output of the multi head attention block. Since the addition may result in values going out of bounds, layer normalization is used.
   Layer normalization transforms every vector of the output matrix in such a way that mean and variance of each vector are 0 and 1 respec-

tively. Following equations,

$$\mu_i = \frac{1}{m}\sum x_{ij}$$
$$\sigma_i = \frac{1}{m}\sum (x_{ij} - \mu_i)^2$$
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta = LN_{\gamma,\beta}(\hat{x}_i)$$

4. **Feed Forward Network:** This is the simple feed forward Neural Network that is applied to every attention vector, it's main purpose is to transform the attention vectors into a form that is acceptable by the next encoder or decoder layer. Feed Forward Network accepts attention vectors "one at a time". And the best thing here is unlike the case of RNN, here each of these attention vectors are independent of each other. So, parallelization can be applied here, and that makes all the difference. The equation summarizes the FFN,

   $$FFN(x) = relu(xW_i + b_1)W_2 + b_2$$

   The output of the FFN is added to output of attention block channeled through a skip connection and layer normalized as before.

2.1.2 Decoder

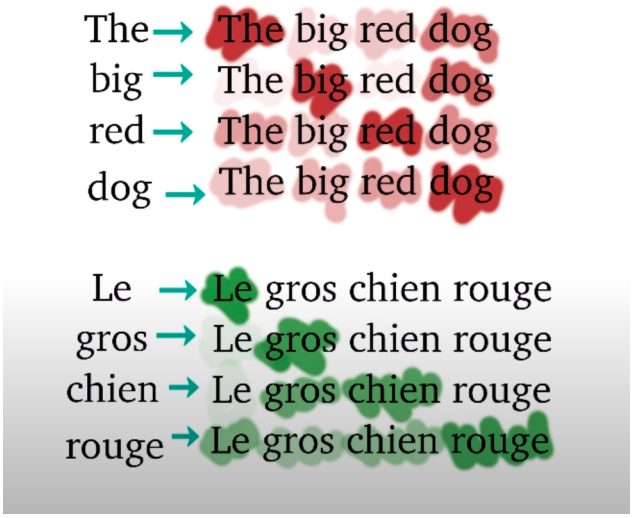
The is quite similar to the encoder in terms of the components present with some essential differences. The decoder feeds on the target sentence.

1. **Positional Encoding:** Same as that of an encoder.

2. **Masked Multihead Attention:** Same as ins encoder. **But why the word masked though?** In order to learn the target sentence well, the decoder attempts to predict the next token based on the sequence it has received till now, so the tokens ahead must be kept masked, or it would just copy the tokens from the training data.

3. **Encoder-Decoder Attention:** The multihead attention block present inside teh decoder is called the Encoder-Decoder Attention, because

this is where the actual mapping happens, the decoder passes its previous state to the encoder as query and receives the id of the token demanding the most attention.



The above figure depicts a sample translation from English to French, where the English feeds into the encoder and French to the decoder. The encoder representation for every token, shown in different shades of red, is essentially its relative significance with each other tokens. The decoder representation (from Masked Miltihead Attention) is almost same as that of encoder except for the words after the current one are masked, hence not colored.

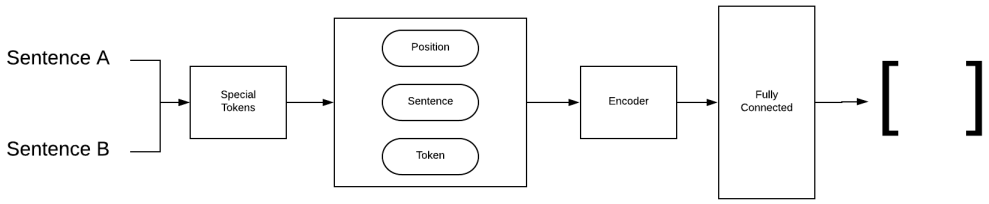
4. **Linear layer and Softmax:** The decoder stack outputs a vector of floats. How do we turn that into a word? That's the job of the final Linear layer which is followed by a Softmax Layer.
   The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.
   Let's assume that our model knows 10,000 unique English words (our model's "output vocabulary") that it's learned from its training dataset. This would make the logits vector 10,000 cells wide – each cell corresponding to the score of a unique word. That is how we interpret the output of the model followed by the Linear layer.

The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

2.2 BERT

Bidirectional Encoder Representation from Transformer, BERT, as the name suggests is essentially a stack of transformer encoders that aims at better capturing the context by reading the input in both directions. The components of a BERT are discussed as follows,

1. **Masked Language Model:** The BERT paper proposes a fresh token prediction approach, that randomly masks a random number of words and later attempts to predict the masked word based on the bidirectional context provided by the other words. More on this topic will be presented later in this paper.
2. **Next Sentence Prediction:** BERT brings to the table an additional feature that predicts if a pair of sentences is likely to follow on the same context, i.e. given a pair of sentences, it outputs a binary value of true if second sentence follows the first and false otherwise.



Next Sentence Prediction

The model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the

original document. During training, 50 percent of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50 percent a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

(a) A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

(b) A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.

(c) A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

T predict the whether it is likely to follow,

(a) The concatenated input goes through a transformer.

(b) Each token outputs a vector of size 768

(c) The output of [CLS] transforms into a 2 length vector using a classification layer

(d) This vector is passed though a softmax and the decision is inferred from here on

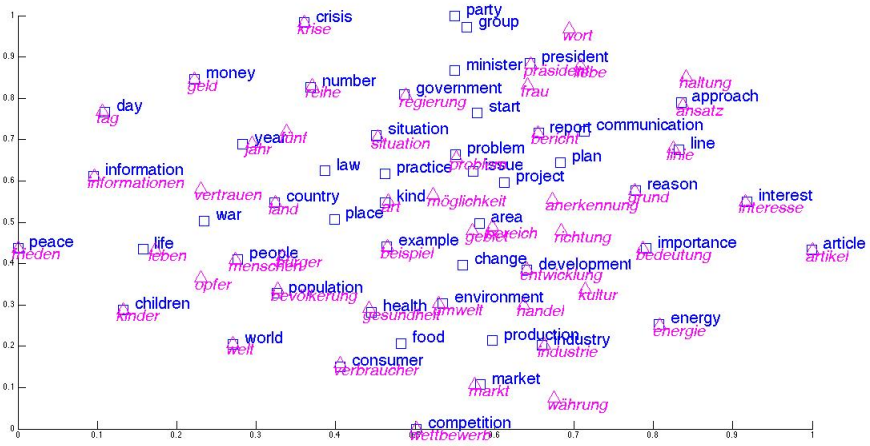
3 XLM Architecture

The architecture of XLM is in a sense similar to that of BERT. The main components of the XLM architecture are elaborated as follows,

3.1 Shared Sub-word Vocabulary

As mentioned earlier, XLM uses Byte Pair Encoding to ensure words with similar meanings stay close in the embedding space irrespective of the parent language they come from. Our goal is to learn a shared embedding space

between words in all languages. Equipped with such a vector space, we are able to train our models on data in any language. By projecting examples available in one language into this space, our model simultaneously obtains the capability to perform predictions in all other languages. Following is a figure that depicts similar words, plotted on a vector, from English with their German counterpart.



The reason for such graphical proximity is the way words are embedded to vectors. Different embedding strategies are used such as Glove, Word2Vec, we shall however discuss a very simple but intuitive form.
Say we've got a small corpus of six words, which are
V = { man, woman, king, queen, apple, orange }
We can select some features to differentiate each word from other. For this example lets select three features such as ***gender, royality, fruit***. We could then embed the words as,

Feature	Man	Woman	King	Queen	Apple	Orange
Gender	1	-1	0.93	-0.95	0	0
Royality	0.01	0.03	1	-1	0	0
Fruit	0	0	0	0	0.98	0.91

There can be the following major takeaways from this embedding scheme,

1. Notice that performing a dot product between any similar pair, say man and woman produces a large value of -0.9997, while that of a pair that is not similar like man and apple would produce a 0.

2. The representation is language independent. Even if we replace the words man woman by hombre and mujer, which are the Spanish counterparts, we're still likely to get a similar vector.

3. Since we're using BPE, described earlier, this allows to reduce OOV encounters by a large factor. For example, say we train the model with the above corpus and encounter the word **manly** during test. Obviously this is a word out of our corpus, but by tokenizing it into **man + ly**, we can now map it to the word **man** that is well within our training data.
   This phenomenon is not limited to English or any monolingual case, but can be extended to modeling languages that have a similar root. For example, the German counterpart for the English word **bring** is **bringen** which obviously splits to **bring + en** and can be handled if the corpus consists of the word **bring**.

The above is just an illustration of how words can be converted into their vector counterparts. The length of each vector in our hypothetical scheme is 3, whereas in actual word embedding may figure in thousands if not millions. This vector can be transformed into a 2 length, vector suitable for graphical plots, by multiplying with an appropriate matrix.

In recent years, various models for learning cross-lingual representations have been proposed. In the following, we will order them by the type of approach that they employ.

- **Monolingual Mapping:** These models initially train monolingual word embeddings on large monolingual corpora. They then learn a linear mapping between monolingual representations in different languages to enable them to map unknown words from the source language to the target language.

    1. **Linear Projection:** Mikolov et al. have popularised the notion that vector spaces can encode meaningful relations between

words. In addition, they notice that the geometric relations that hold between words are similar across languages, e.g. numbers and animals in English show a similar geometric constellation as their Spanish counterparts in the following figure.

This suggests that it might be possible to transform one language's vector space into the space of another simply by utilising a linear projection with a transformation matrix. In order to achieve this, they translate the 5,000 most frequent words from the source language and use these 5,000 translations pairs as bilingual dictionary. They then learn W using stochastic gradient descent by minimising the distance between the previously learned monolingual representations x_i of the source word w_i that is transformed using W and its translation z_i in the bilingual dictionary. This model initially learns the monolingual embeddings with,

$$\sum_{i=1}^{N} \sum_{-C \leq j \leq C, j \neq 0} logP(w_{i+j}|w_i)$$

which is a skip-gram that skips j tokens. The probability is calculated with the softmax,

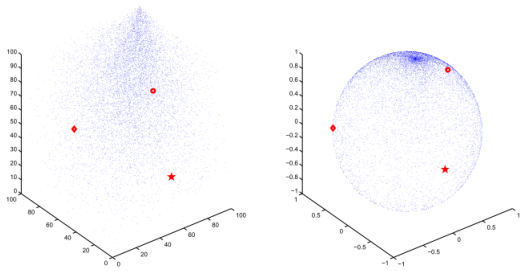
$$\frac{exp(v_{w_{i+j}}^T v_{w_i})}{\sum exp(v_w^T v_{wi})}$$

where v_w is the word vector for the word w.
Then the transformation is learnt as,

$$min \sum_i |Wx_i - z_i|^2$$

where W is learnt such that the squared distance minimizes.

2. **Normalization and Orthogonal Projection:** This section addresses the following inconsistencies of linear projection,
   (a) Every inner product used to gain maximum likelihood, $c_w^T c'_w$ is normalized as $\frac{c_w^T c'_w}{|c_w^T||c'_w|}$, which makes the inner product the same as cosine similarity performs better than inner product when considering only the angle between the vectors. This shape shifts the embedding space into a hyper sphere.



(b) The squared distance minimisation object is also replaced by a cosine similarity maximization objective, since cosine similarity outperforms Euclidean distance metrics in case of higher dimensional vectors. The new objective is,

$$max \sum_i (Wx_i)^T z_i$$

where W is constrained to be orthogonal, so that Wx_i is normalized, with a separate optimisation objective.

3. **Max Margin:** Yet another modification that can be made to linear projection is to eliminate *hubness*. This is where some words may appear to be neighbors of many other words. It is suggested to use the max margin scheme here, where the distance is calculated as,

$$d = \gamma + cos(\hat{y}_i, y_i) - cos(\hat{y}_i, y_j)$$

where γ is the chosen maximum distance, $\hat{y}_i$ is the correct translation, y_i is the source vector, y_j are other vectors that are not correct translation.

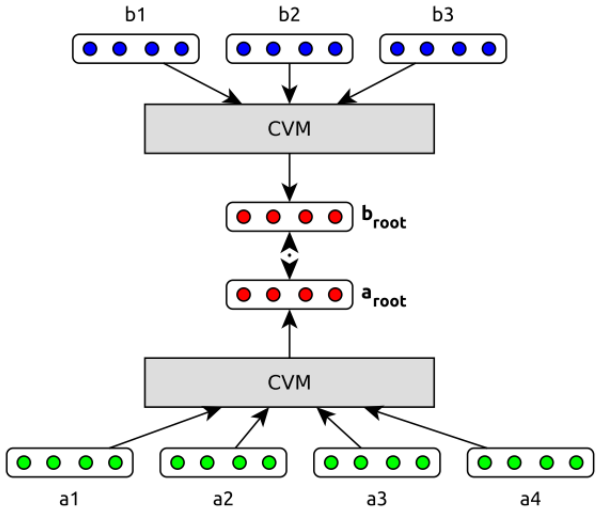
- **Cross-lingual Mapping:** Cross-lingual training approaches focus exclusively on optimising the cross-lingual objective. These approaches typically rely on sentence alignments rather than a bilingual lexicon and require a parallel corpus for training.

  1. **Bilingual compositional sentence model:** The first approach that optimizes only a cross-lingual objective is the bilingual compositional sentence model by Hermann and Blunsom. They train

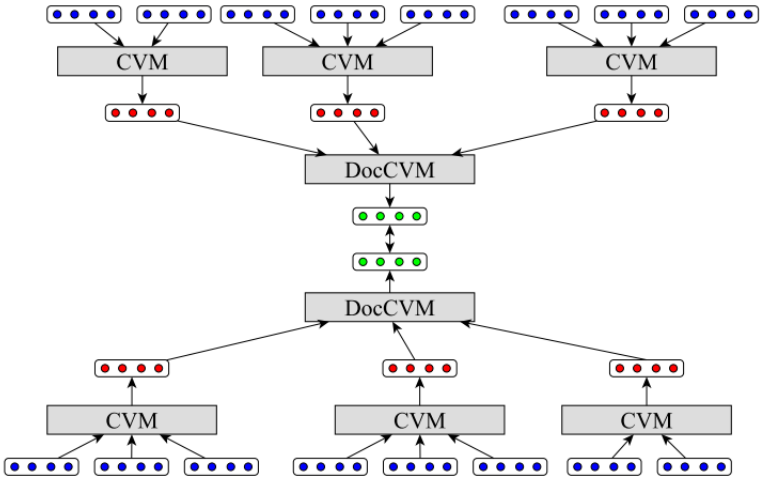
two models to produce sentence representations of aligned sentences in two languages and use the distance between the two sentence representations as objective. They minimise the following loss:

$$Dist(a, b) = |a_{root} - b_{root}|^2$$

where a_{root} and b_{root} are vector representations of aligned sentences from two languages. **How's the representation formed?** This is just the sum of word wise vector representation of each sentence. The entire model is depicted in the following diagram,



2. **Bilingual compositional document model:** The above approach could be extended to documents, by applying the composition and objective function recursively to compose sentences into documents. First, sentence representations are computed as before. These sentence representations are then fed into a document-level compositional vector model, which integrates the sentence representations in the same way as can be seen in the following figure.



The advantage of this method is that weaker supervision in the form of document-level alignment can be used instead of or in conjunction with sentence-level alignment. The authors run experiments both on Europarl as well as on a newly created corpus of multilingual aligned TED talk transcriptions and find that the document signal helps considerably. The proposed technique to compose words here is, however, changed to one involving *tanh* function instead of just adding as that helps keep the figures within the range of $(-1, 1)$. The composition function is,

$$f(x) = \sum_{i=1}^{n} tanh(x_{i-1} + x_i)$$

3.2 Input Embedding

The input to the model is composed of three components added together. These are,

1. **Token embedding:** The input sentence is first reduced to the available BPE tokens. These tokens are then processed with various embedding techniques to convert into a word vector representation.

2. **Position embedding:** The position embedding is added to enforce the model to care more about position of the token within a sentence.

Position embedding is similar to that of BERT and uses *sin*/*cos* pairs, as discussed before.

3. **Language embedding:** Since the input may originate from virtually any language around the globe, so a unique identifier for each language is added to inform the model where the sequence originates from.

3.3 Causal Language Modeling

The causal language model consists of a transformer that predicts the next token given the context up until now. This model is autoregressive in nature. Given a text sequence $\mathbf{x} = [x_1, x_2, ..., x_T]$, the model aims to maximize the following objective function,

$$\textbf{max}\ log\ p(x) = \sum_{t=1}^{T}\ log\ p(x_t \mid x_{<t}) = \sum_{t=1}^{T}\ log\ \frac{exp(h(x_{1:t-1}^{T})e(x_t)))}{\sum_{x'} exp(h(x_{1:t-1}^{T}e(x')))}$$

This holds because the probability of a sequence x to exist is equal to the product of the probabilities of each of constituent tokens of x to hold. That is,

$$p(x) = p(x_1).p(x_2)...p(x_T) = \prod_{t=1}^{T} p(x_t)$$

At a very high level, what the model is attempting is to predict the sequence of highest probability. The function h here is a function that returns the context representation produced by an encoder. This is then multiplied with $e(x)$ which is the vector representation of the word x. Say the encoder representation till $t-1$ and word embedding for t^{th} token are,

$$\begin{bmatrix} 2 & 3 & 4 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} 1 & 2 \end{bmatrix}$$

Then the numerator of the objective becomes,

$$\begin{bmatrix} 2 & 4 \\ 3 & 5 \\ 4 & 6 \end{bmatrix} . \begin{bmatrix} 1 & 2 \end{bmatrix} = 39$$

3.4 Masked Language Modeling

This is the BERT like language modeling problem, that uses ***Denoising Autoencoder. What that is?*** The model randomly masks some of the tokens and tries to predict those based on the surrounding context. That is it deliberately introduces noise to the input and tries to reconstruct the sequence, this provides the model to have a better grasp of the language. This is carried out in the following manner,

- Take the BPE tokens as input
- **80%** times replace 15% tokens with the [**MASK**] token
- **10%** times replace 15% tokens with the random other token
- **10%** times keep them unchanged

This model is bidirectional, so is expected to perform better by taking into account both left and right context. The only difference with BERT, however, is that BERT allows a pair of sentence to be intaken ata a time, while this model allows more than two. Like **CLM**, **MLM** also has an objective function to optimize, that is,

$$\textbf{max}\ log\ p(\bar{x}|\hat{x}) \approx \sum_{t=1}^{T} m_t\ log\ p(x_t|\hat{x}) = \sum_{t=1}^{T} m_t\ log\ \frac{H(\hat{x})_t^T.e(x_t)}{\sum_{x'} H(\hat{x})_t^T.e(x')}$$

Explanation Say we've the sequence $\mathbf{x} = [x_1, x_2, ..., x_T]$. We pass this input sequence through a function,

$$f : x \rightarrow \hat{x}$$

such that it masks a definite portion of the tokens (15% here) and produces the corrupt sequence . The aim is to reconstruct x from $\hat{x}$. m_t is a flag variable that is 1 if the x_t is a masked token. $H(x)$ is a transformer that

forms the hidden vectors as $H(x) = [H(x)_1, H(x)_2, ..., H(x)_T]$.
Let's break it down. The first part, **max** $log\ p(\bar{x}|\hat{x})$, asks the question what is the reconstruction of $\bar{x}$ with highest probability? The second part computes the probability for each x_t to be the masked token, given the context $\hat{x}$ and maximizes that. That is it looks for an x_t that would best fit in to the given context. How is the best fit computed? Through inner product. We take the inner product of the context representation $H(\hat{x})_t^T$ with the embedding for x_t and pass that through a softmax to transform that into a vector that sums up to 1, that is a vector of probability, where i^{th} index represents the probability of x_i being the masked token.

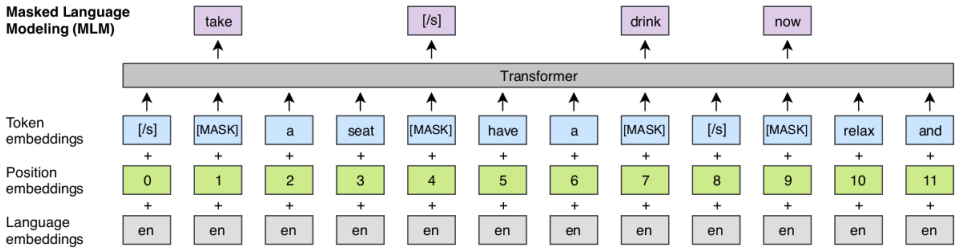
Following factors are to be noted here,

1. **Masked word prediction:** In the second statement, $\bar{x}$ has been replaced by x_t, that is the operation is carried out for each token, which is a contradiction to our initial purpose of only predicting the masked ones, so a flag variable is introduced. The flag variable m_t forces the model so that the prediction is performed only on the masked tokens.

2. **Context dependency:** The prediction is performed given the entire corrupt sequence $\hat{x}$, that is the bidirectional context is available. The difference between $h(x_{1:t-1}$ of CLM and $H(x)_t$ of MLM is that $H(x)_t$ contains a bit more information about the input sequence.

3. **Independence assumption:** It should be noted that the equation consists of $\approx$ rather than equality. The reason for that is it assumes every masked token is being predicted separately, because $\hat{x}$ is being altered for every new x_t. CLM objective however doesnt make such an assumption and uses the product rule to factorize.

4. **Input noise:** MLM consists of artificial tokens such as [**MASK**] which occur during the pretraining but not in testing phase. This creates a train-test discrepancy. It has been shown that the prediction is often influence by the position of the masked token. Tokens masked earlier in the sequence are less likely to be predicted correctly as compared to the later ones.



One thing that remains unanswered is there are many words that occur in overwhelmingly greater frequencies than others, **How is the imbalance between rare and frequent words countered?** The frequent outputs are subsampled and are assigned a weight such as,

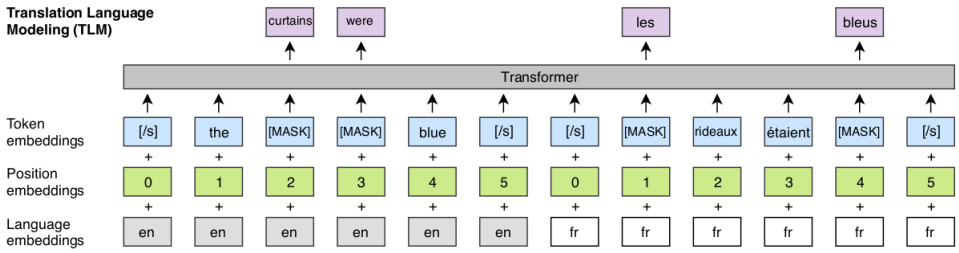
$$w_i = \frac{C}{\sqrt{n_i^2}}$$

where w_i is the weight for i^{th} output, n_i the frequency and C just a constant for proportionality.

Feature	CLM	MLM
Context dependency	Upto $< t$ for every t	Entire context captured
Factorization	Using product rule	Independence assumption
Prediction style	Every next word	Only the masked words
Noise influence	Absent	Present

3.5 Translation Language Model

Models we so far have discussed only require monolingual unsupervised data. They cannot leverage parallel data when that is available. For this, authors introduce the Translation Language Model(TLM). This is an extension of the MLM objective. The input to the model is fed as a concatenation of parallel sentences in different languages. Tokens are masked randomly and model predicts from the surrounding context. In the following example, the model is fed an English sentence concatenated with its French counterpart. Interesting point about this is that the model leverages the context from the French portion of the input to predict the masked tokens in English and vice-verse if the former is not enough.



3.6 Cross-lingual Language Models

The paper considers language modeling in two approaches, which are CLM-MLM approach used for unsupervised monolingual corpora and MLM-TLM approach for unsupervised parallel copora. For CLM-MLM objective the model is trained with input length of 256 tokens, zero-padding used for shorter inputs that compose a mini-batch of size 64. At each iteration, 64 sentences are chosen from corpus of same language, the following probability distribution is used to select the sentences,

$$q_i = \frac{p_i^\alpha}{\sum_{j=1}^{N} p_j^\alpha}, p_i = \frac{n_i}{\sum_{k=1}^{N} n_k}$$

Where q_i is the probability of a sentence being selected, n_i the length of the corpus C_i and α a constant used to control the weights.
But how does that work? Consider this set of numbers where i^{th} index denotes the length of C_i, for illustration,

$$C = [35, 5000, 251, 39]$$

Here is how α plays a role in putting all the different lengths to the same platform,

α	q_i
0.7	{0.026, 0.842, 0.103, 0.028}
0.5	{0.044, 0.529, 0.118, 0.046}
0.1	{0.205, 0.337, 0.249, 0.207}

The way the α parameter regulates the distribution is as α is reduced, the probability distribution come more and more closer.

3.7 CLM MLM Implementation

Here is an implementation of CLM and MLM approaches from Hugging Face.

CLM

```
[ ] import torch
    from transformers import GPT2Tokenizer, GPT2LMHeadModel
```

```
[ ] tokeniser = GPT2Tokenizer.from_pretrained('gpt2')
    model = GPT2LMHeadModel.from_pretrained('gpt2')
```

```
text = "i am very exited to present to you this"

def generate_text(text, n_words=3):
  for i in range(n_words):
    indexed_tokens = tokeniser.encode(text)
    tokens_tensor = torch.tensor([indexed_tokens])

    outputs = model(tokens_tensor)
    predictions = outputs[0]


    predicted_index = torch.argmax(predictions[0, -1, :]).item()
    print(predictions[0, -1, :])
    text = tokeniser.decode(indexed_tokens + [predicted_index])

  return text

generate_text(text)

# [output]: i am very exited to present to you this new book.
```

MLM

```
from transformers import pipeline
import operator

model = pipeline('fill-mask')
preds = model('I am <mask> this lecture')

def print_sentence(preds):
  scores = list(map(lambda x: x['score'], preds))
  max_index, _ = max(enumerate(scores), key=operator.itemgetter(1))
  pred_sentence = preds[max_index]['sequence']
  return pred_sentence

print_sentence(preds)

# [output]: I am giving this lecture
```

4 Experiments and results

4.1 Cross-lingual classification

	en	fr	es	de	el	bg	ru	tr	ar	vi	th	zh	hi	sw	ur	Δ
Machine translation baselines (TRANSLATE-TRAIN)																
Devlin et al. (2018)	81.9	-	77.8	75.9	-	-	-	-	70.7	-	-	76.6	-	-	61.6	-
XLM (MLM+TLM)	85.0	80.2	80.8	80.3	78.1	79.3	78.1	74.7	76.5	76.6	75.5	78.6	72.3	70.9	63.2	76.7
Machine translation baselines (TRANSLATE-TEST)																
Devlin et al. (2018)	81.4	-	74.9	74.4	-	-	-	-	70.4	-	-	70.1	-	-	62.1	-
XLM (MLM+TLM)	85.0	79.0	79.5	78.1	77.8	77.6	75.5	73.7	73.7	70.8	70.4	73.6	69.0	64.7	65.1	74.2
Evaluation of cross-lingual sentence encoders																
Conneau et al. (2018b)	73.7	67.7	68.7	67.7	68.9	67.9	65.4	64.2	64.8	66.4	64.1	65.8	64.1	55.7	58.4	65.6
Devlin et al. (2018)	81.4	-	74.3	70.5	-	-	-	-	62.1	-	-	63.8	-	-	58.3	-
Artetxe and Schwenk (2018)	73.9	71.9	72.9	72.6	73.1	74.2	71.5	69.7	71.4	72.0	69.2	71.4	65.5	62.2	61.0	70.2
XLM (MLM)	83.2	76.5	76.3	74.2	73.1	74.0	73.1	67.8	68.5	71.2	69.2	71.9	65.7	64.6	63.4	71.5
XLM (MLM+TLM)	**85.0**	**78.7**	**78.9**	**77.8**	**76.6**	**77.4**	**75.3**	**72.5**	**73.1**	**76.1**	**73.2**	**76.5**	**69.6**	**68.4**	**67.3**	**75.1**

Figure 1: **Results on cross-lingual classification accuracy**. Test accuracy on the 15 XNLI languages. We report results for machine translation baselines and zero-shot classification approaches based on cross-lingual sentence encoders. XLM (MLM) corresponds to our unsupervised approach trained only on monolingual corpora, and XLM (MLM+TLM) corresponds to our supervised method that leverages both monolingual and parallel data through the TLM objective. $\triangle$ corresponds to the average accuracy.

This figure 1 shows the XNLI benchmark on 15 different languages. XNLI is a machine translation baselines and zero-shot classification using cross-lingual sentence encoders. In this case, the authors are trying to classify the different languages and if we look at the bottom row here, we'll see that XLM using the MLM and TLM pre-training objective performs the best across the board in comparison to other methods like devlin at all 2018(BERT model). The key here is that using the MLM and TLM objective performes the best.

4.2 Unsupervised Machine Translation

		en-fr	fr-en	en-de	de-en	en-ro	ro-en
Previous state-of-the-art - Lample et al. (2018b)							
NMT		25.1	24.2	17.2	21.0	21.2	19.4
PBSMT		28.1	27.2	17.8	22.7	21.3	23.0
PBSMT + NMT		27.6	27.7	20.2	25.2	25.1	23.9
Our results for different encoder and decoder initializations							
EMB	EMB	29.4	29.4	21.3	27.3	27.5	26.6
-	-	13.0	15.8	6.7	15.3	18.9	18.3
-	CLM	25.3	26.4	19.2	26.0	25.7	24.6
-	MLM	29.2	29.1	21.6	28.6	28.2	27.3
CLM	-	28.7	28.2	24.4	30.3	29.2	28.0
CLM	CLM	30.4	30.0	22.7	30.5	29.0	27.8
CLM	MLM	32.3	31.6	24.3	32.5	31.6	29.8
MLM	-	31.6	32.1	**27.0**	33.2	31.8	30.5
MLM	CLM	**33.4**	32.3	24.9	32.9	31.7	30.4
MLM	MLM	**33.4**	**33.3**	26.4	**34.3**	**33.3**	**31.8**

Figure 2: **Results on unsupervised MT**. BLEU scores on WMT'14 English-French, WMT'16 German-English and WMT'16 Romanian-English. For our results, the first two columns indicate the model used to pretrain the encoder and the decoder. " - " means the model was randomly initialized. EMB corresponds to pretraining the lookup table with cross-lingual embeddings, CLM and MLM correspond to pretraining with models trained on the CLM or MLM objectives.

In figure 2 we show results for unsupervised machine translation. These are the blue scores. The table does is that it looks at the performance of the encoder and decoder pre-training for XLM. XLM retrains both the encoder and decoder which is unlike BERT which is only the encoder and unlike GBT which is only the decoder. We see here is that EMB is the using the lookup table with cross-lingual embeddings. The dash is random initialization of the weights of the model and CML is a casual language model, MLM is the mass language model. If we look at the best numbers which are in bold we'll see that for the encoder and decoder, the MLM objective works the best. If we mix it any other way, it wouldn't be as good across the board. So,

this shows that the different pre-training schemes used in XLM, the mass language model performs the best.

4.3 Supervised Machine Translation

Pretraining	-	CLM	MLM
Sennrich et al. (2016)	33.9	-	-
ro → en	28.4	31.5	35.3
ro ↔ en	28.5	31.5	35.6
ro ↔ en + BT	34.4	37.0	**38.5**

Figure 3: **Results on supervised MT**. BLEU scores on WMT'16 Romanian-English. The previous state-of-the-art of Sennrich et al. (2016) uses both back-translation and an ensemble model. ro ↔ en corresponds to models trained on both directions.

Figure 3 is for supervised machine learning. The blue scores for Romanian to English. The mass language model here with the bold shows the best and the highest performance across the board. The key takeaway is that mass language model performs the best for XLM.

4.4 Low-resource language model

The author have investigated the impact of cross-lingual language modeling for improving the perplexity of a Nepali language model. Perplexity measures how well a language model is or performs. The perplexity is a positive number.

Perplexity: Assume that we have some test data sentences $x^{(1)}$, $x^{(2)}$,....., $x^{(m)}$. Each test sentence $x^{(i)}$, for i $\in \{1...m\}$ is a sequence of words $x_1^{(i)}$, $x_2^{(i)}$,...., $x_{n_i}^{(i)}$, where n_i is the length of the i'th sentence. We assume that every sentence ends in the STOP symbol.

It is critical that the test sentences are "held out", in the sense that they are not part of the corpus used to estimate the language model. In this sense,

they are examples of new, unseen sentences.
For any test sentence $x^{(i)}$, we can measure its probability $p(x^{(i)})$ under the language model. A natural measure of the quality of the language model would be the probability it assigns to the entire set of test sentences, that is:

$$\prod_{i=1}^{m} p(x^{(i)})$$

The intuition is as follows: the higher this quantity is, the better the language model is at modeling unseen sentences.
The perplexity on the test corpus is derived as a direct transformation of this quantity. Define M to be the total number of words in the test corpus. More precisely, under the definition that n_i is the length of the i'th test sentence,

$$M = \sum_{i=1}^{m} n_i$$

Then the average log probability under the model is defined as

$$\frac{1}{M} \log_2 \prod_{i=1}^{m} p(x^{(i)}) = \frac{1}{M} \sum_{i=1}^{m} \log_2 p(x^{(i)})$$

This is just the log probability of the entire test corpus, divided by the total number of words in the test corpus. Here we use $\log_2(z)$ for any $z>0$ to refer to the log with respect to base 2 of z. Again, the higher this quantity is, the better the language model.
The perplexity is then defined as

$$2^{-l}$$

where

$$l = \frac{1}{M} \sum_{i=1}^{m} \log_2 p(x^{(i)})$$

Thus we take the negative of the average log probability, and raise two to that power. (Again, we're assuming in this section that $\log_2$ is log base two). The perplexity is a positive number. The smaller the value of perplexity, the better the language model is at modeling unseen data.

Training languages	Nepali perplexity
Nepali	157.2
Nepali+ English	140.1
Nepali+ Hindi	115.6
Nepali+ Hindi+ English	109.3

Figure 4: **Results on language modeling**. Nepali perplexity when using additional data from a similar language (Hindi) or a distant one (English).

Nepali and English are distant languages, but Nepali and Hindi are similar as they share the same Devanagari script and have a common Sanskrit ancestor. After training with Nepali, if we train XLM on English and Nepali data that perplexity for Nepali goes down. Again, these are different languages that are trained on the same cross-lingual model, but there's no parallel data between them, the XML is just training in Nepali, then train with English. If we train Nepali and then Hindi, we get an even lower perplexity because Hindi based on Sanskrit is more closely related to Nepali. Now, if we train XLM on all three, Nepali and in English and Hindi, you will get the perplexity is even lower.
The cross-lingual language model can thus transfer the additional context provided by the Hindi or English monolingual corpora through these anchor points to improve the Nepali language model.

4.5 Unsupervised cross-lingual word embedding

	Cosine sim.	L2 dist.	SemEval'17
MUSE	0.38	5.13	0.65
Concat	0.36	4.89	0.52
XLM	**0.55**	**2.64**	**0.69**

Figure 5: **Unsupervised cross-lingual word embeddings** Cosine similarity and L2 distance between source words and their translations. Pearson correlation on SemEval'17 cross-lingual word similarity task of Camacho-Collados et al. (2017).

Cross-lingual word embeddings models train their embeddings on a parallel corpus and optimize a cross-lingual constraint between embeddings of different languages that encourages embeddings of similar words to be close to each other in a shared vector space. One way of measuring it is to use different similarity scores. We can see in figure 5, the first column is a cosine similarity score. The higher the cosine similarity which is a dot product is the closer the difference. The closer the word embeddings for the different languages the better they align.
The L2 distance is the distance measure. The smaller it is also the better it aligns which in this case the XLM does do the best here. SemEval is a pearson correlation coefficient. The higher the correlation coefficient is the better which in this case it does here too. SemEval 17 is another benchmark but uses the piercing correlation coefficient. XLM compares to two other models MUSE and Concat. MUSE is based on fast text multilingual word embeddings. Fast text is like Word to Vec but trained using subword tokens. Concat is also fast text word embedding but it's trained by concatenating monolingual corpora from different languages. So, in figure 5, the XLM has been shown to do better performance.