ENSAE NLP Project : Language Modeling

Salomé Do, Gabriel Romon

I. Introduction

We chose to work on the language modeling task. Language modeling aims at assigning a probability to a sentence. For a long time, language modeling has been done with n-gram models, and their smoothing variants which are now well-known. Recently, neural models, especially recurrent neural networks, have been used for the task, giving interesting results in language modeling.

The language modeling task is truly at the heart of many other NLP tasks, as it has been at the origin of the late 2018's NLP developments on transfer learning. Pre-training deep neural networks on a language modeling task (ULM-fit [Howard & Ruder, 2018], and GPT [Radford *et al.* , 2019]), or a masked language model task (BERT [Devlin *et al.* , 2018]), and fine-tuning these models on other tasks as text classification, question answering, etc. has enabled to achieve state-of-the-art results in many tasks of the GLUE benchmark with a single model.

II. Task, datasets, implementation

A. Neural Language Modeling

In this work, we focus only on the neural language modeling task : we train the neural models by feeding an input token sequence to the model, which must predict the next tokens. This way, we can evaluate the model by computing the perplexity directly as the exponential of the cross-entropy loss.

B. Datasets and example generation

We used the word-levels datasets given for the task, `wiki`, and `tatoeba`. These datasets were already cleaned, tokenized, lowercased, and infrequent words were replaced by the token `<unk>`.

The different models we built were fed two types of data, generated from `wiki` or `taoteba` :

- **Sentence-level data** : for a sentence `<s> the police were ready to arrest sami .</s>` for instance, we generated the following (input, output) couples : (`<s>`, `the`), (`<s> the`, `police`),(`<s> the police`, `were`), etc. (see Figure 2 in Appendix). This example generation enables to learn sentences, but cannot generalize to multi-sentences text generation.
- **Cross-sentence n-grams** : in order to have a complementary multi-sentence level for our text generation, we generate couples of (input, output) by generating the n-grams (we took $n = 5$) over the corpus, taking the first $n-1$ tokens as an input and guessing the n-th token.

In order to have both long-term sentence dependencies and short-term cross-sentence dependencies, we mix these two ways of generating examples.

Finally, examples are batched, padded, but **not** shuffled, as shuffling examples has detrimental impact on model performance, as described in [Press, 2019]. We decided to follow a different scheme, by not shuffling batches, but mixing our two ways of generating examples.

C. Implementation

The implementation framework we used is Tensorflow 1.13. The code, available online [1], has been designed to be entirely reproducible, and modular. It only requires a dataset and a conda environment, and it runs entirely from the command line. The shell script `train.sh` trains all the models presented here (runtime is over 5-6 hours with 8 CPU cores and a GTX 1080 Ti GPU). We do not provide checkpoints for the model, as they are heavy. Results can be visualized with Tensorboard. A predict script is given, in order to play with the text generation feature.

III. Model Architectures

We investigated many architectures, which are generally composed of an embedding layer, a deep encoder, and a dense classification output. We detail the composition of these layers.

A. Embedding Layer

The embedding layer was made of two distinct parts :

1) Word embedding: The word embedding layer is a simple dense layer, transforming the one-hot word vectors, dimension `vocabulary size`, to a dense output, dimension `embedding size`. In our early modeling, we compared the advantages of using a pre-trained word embedding such as Google's Word2Vec or FastText over a simple embedding trained on the fly, and found that there was no significant need for pre-trained word embeddings.

2) Character-level embedding: An option for our model is to add a character-level embedding as proposed in [Ling *et al.* , 2015]. This character-level embedding aims at giving a representation a word's form, by splitting the word into characters, passing characters in a Bi-LSTM,

1. `https://github.com/sally14/language-model-ENSAE`

and retrieving final states. This enables to create an open-vocabulary embedding, which is complementary with the token-level word embedding, as it focuses on the token's form.

The representations are eventually concatenated, word by word.

B. LSTM Base Architecture

As we have variable-length inputs, due to the sentence-level example generation, we need to have a variable-length layer which produces a fixed-length representation of the sequence. A recurrent neural network, such as an LSTM, is appropriate for this task, as suggested in [Merity *et al.* , 2017]. Thus, our word embeddings pass in a multi-layer LSTM, which returns the last LSTM output state for each sequence. Each sequence is then represented by an `hidden size LSTM`-dimensional vector.

C. Dense Layer

The `hidden size LSTM`-dimensional vector generated by the multi-layer LSTM is then passed to a (eventually multi-layer) dense network, with final dimension `vocabulary size`. While the inner layers are activated by a leaky ReLU function, the last layer of the dense block passes through the softmax function to produce logits probabilities.

D. Additional experimental modules

We experimented with a multi-headed attention encoder based on [Vaswani *et al.* , 2017], which was placed between word embeddings and the multi-layer LSTM. An ideal feature would have been to use only attention, however, we should have used n-grams examples generation only, in order to have a fixed output dimension. This has unfortunately not been implemented.

E. Losses

We tried many losses during modeling :

1) Cross-entropy loss: The cross-entropy loss takes as input the logits generated by a softmax on the last layer of the Dense block, and compares it to the one-hot true labels. Unfortunately, it suffers from the softmax computation, which is expensive when `vocabulary size` grows. Still, this was our best loss. Perplexity can be computed by taking the exponential of losses means.

2) Weighted loss: As some words are more frequent than others, and can thus induce some biases, we also tried to train the model on a weighted loss. This weights were computed the following way : if we have a vocabulary $V = x_1, ..., x_n$, with associated corpus frequencies $F = f_1, ..., f_n$, then the weight w_i of a token x_i is :

$$w_i = c \times \frac{1/f_i}{\sum_{j=1}^{n} 1/f_j}$$

Frequent words are thus given a smaller importance than rare words. As weights are very small, which is not good for optimizers, they are scaled by a constant $c \approx 10^5$ (this was chosen empirically). However, the weighted loss did not yield encouraging results, as averaging it on batch sizes of 10 or 20 might still be too large.

3) Candidate sampling loss: Finally, we also tried candidate sampling, which is a common technique when the output dimension is too large, and the softmax computation is too expensive. As in [Mikolov *et al.* , 2013], we tried negative sampling, but also noise-contrastive estimation loss. This unfortunately did not give good results.

IV. Results

We report training results for $n = 4$ epochs for `wiki` and $n = 6$ epochs for `taoteba`, with RMSPROP optimizer and learning rate $lr = 0.001$. The hyperparameters were empirically fixed to `embedding size`$= 300$, `hidden size LSTM` $= 600$, `num LSTM layers` $= 2$, `dropout` $= 0.5$ during training, and for the optional blocks : `hidden size char` $= 25$, `num chars LSTM` $= 2$, `num attention heads` $= 10$.

Base LSTM stands for the model with only on-the-fly word embedding, multi-layer LSTM and a dense block with 3 layers. Char stands for the bi-LSTM character embedding, and Encoder LSTM stands for the experiment with the attention encoder followed by the LSTM.

A. `wiki` corpus

Arch.	Train PP	Acc.	Test PP	Acc.
Base LSTM	66.68	29.43%	**71.16**	**29.27**%
Char + Base LSTM	**60**	**30**%	73.69	29.53%
Char + Encoder LSTM	73.6	28.31%	81.5	27.96%

B. `taoteba` corpus

Arch.	Train PP	Acc.	Test PP	Acc.
Base LSTM	24	35.96 %	**36.59**	**33.14** %
Char + Base LSTM	**22.19**	**38.41**%	43	31.61%
Char + Encoder LSTM	25.75	36.94%	41.67	32.5§%

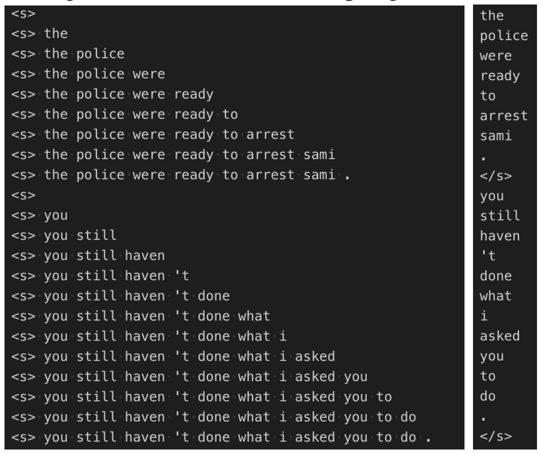
V. Conclusions

Our best model seems to be our simplest model. While this might be quite surprising that the model with the character-level embedding has poorer results than the base model, it is no surprise that the Encoder LSTM model has the worse results. Attention transformers models are better when not mixed with other types of networks, as it can be seen in purely attention-based models as [Devlin *et al.* , 2018] . Thought, building and training such attention-based models requires huge computational resources, in contrast to our base model, which is light and quick to train (<1h). The character-level embedding implementation, still, should be reviewed.

APPENDIX

EXAMPLES GENERATION

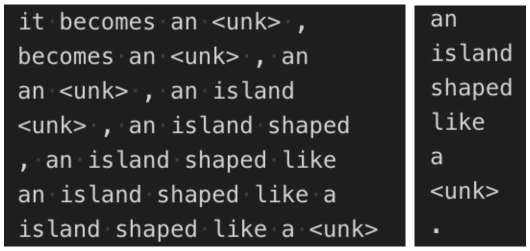
Fig. 1: Sentence level example generation



```
<s>
<s> the
<s> the police
<s> the police were
<s> the police were ready
<s> the police were ready to
<s> the police were ready to arrest
<s> the police were ready to arrest sami
<s> the police were ready to arrest sami .
<s>
<s> you
<s> you still
<s> you still haven
<s> you still haven 't
<s> you still haven 't done
<s> you still haven 't done what
<s> you still haven 't done what i
<s> you still haven 't done what i asked
<s> you still haven 't done what i asked you
<s> you still haven 't done what i asked you to
<s> you still haven 't done what i asked you to do
<s> you still haven 't done what i asked you to do .
```

```
the
police
were
ready
to
arrest
sami
.
</s>
you
still
haven
't
done
what
i
asked
you
to
do
.
</s>
```

Fig. 2: Cross-sentence 5-grams example generation



```
it becomes an <unk> ,
becomes an <unk> , an
an <unk> , an island
<unk> , an island shaped
, an island shaped like
an island shaped like a
island shaped like a <unk>
```

```
an
island
shaped
like
a
<unk>
.
```

REFERENCES

[Devlin *et al.* , 2018] DEVLIN, JACOB, CHANG, MING-WEI, LEE, KENTON, & TOUTANOVA, KRISTINA. 2018. BERT : pre-training of deep bidirectional transformers for language understanding. *Corr*, **abs/1810.04805**.

[Howard & Ruder, 2018] HOWARD, JEREMY, & RUDER, SEBASTIAN. 2018. Fine-tuned language models for text classification. *Corr*, **abs/1801.06146**.

[Ling *et al.* , 2015] LING, WANG, LUÍS, TIAGO, MARUJO, LUÍS, ASTUDILLO, RAMÓN FERNANDEZ, AMIR, SILVIO, DYER, CHRIS, BLACK, ALAN W., & TRANCOSO, ISABEL. 2015. Finding Function in Form : Compositional Character Models for Open Vocabulary Word Representation. 1520–1530.

[Merity *et al.* , 2017] MERITY, STEPHEN, KESKAR, NITISH SHIRISH, & SOCHER, RICHARD. 2017. Regularizing and optimizing LSTM language models. *Corr*, **abs/1708.02182**.

[Mikolov *et al.* , 2013] MIKOLOV, TOMAS, CHEN, KAI, CORRADO, GREG, & DEAN, JEFFREY. 2013. Efficient Estimation of Word Representations in Vector Space. 1–12.

[Press, 2019] PRESS, OFIR. 2019. Partially shuffling the training data to improve language models. *Corr*, **abs/1903.04167**.

[Radford *et al.* , 2019] RADFORD, ALEC, WU, JEFF, CHILD, REWON, LUAN, DAVID, AMODEI, DARIO, & SUTSKEVER, ILYA. 2019. Language models are unsupervised multitask learners.

[Vaswani *et al.* , 2017] VASWANI, ASHISH, SHAZEER, NOAM, PARMAR, NIKI, USZKOREIT, JAKOB, JONES, LLION, GOMEZ, AIDAN N., KAISER, LUKASZ, & POLOSUKHIN, ILLIA. 2017. Attention is all you need. *Corr*, **abs/1706.03762**.