Does data augmentation matter more than architecture design for small datasets?

Stanford CS224N Default Project

Max Sobol Mark
Department of Computer Science
Stanford University
maxsobolmark@stanford.edu

Abstract

The Transformer-XL paper ([**?**]) introduced an innovative architecture that enabled transformers to take advantage of unbounded context lengths, surpassing SOTA performance in several datasets such as Penn Treebank, WikiText-103, and One Billion Word. However, these results are controversial, since it turns out that, without being mentioned in the paper, two data augmentation techniques are absolutely essential for getting good performance in small datasets. In particular, the two methods are discrete embedding dropout, and standard dropout on the input embeddings.
The goal of this project is to answer the question: are data augmentation techniques more influential than the model architecture itself for small datasets? In particular, would a LSTM-based architecture also benefit from discrete and standard embedding dropout? Could it perform competitively with a transformer-based architecture?
Our testing shows that LSTM-based architectures indeed can benefit from discrete dropout embedding. Unfortunately, due to resource constraints we were unable to get our Transformer-XL implementations to reach convergence and exhibit performance comparable to the LSTM-based baseline, since the Transformer-XL is a very big model requiring a lot of resources and time to train. Future work could focus on extending our evaluation with more epochs until convergence to perform a fair comparisson between the architectures.

1 Key Information to include

- Mentor: Ben
- External Collaborators (if you have any): None.
- Sharing project: No.

2 Introduction

Reproducibility is a pillar of science, and is necessary to guarantee efficient resources allocation in a field of study. In Natural Language Processing, bigger models usually get better performance observing a power-law, and the trend in the field has been that bigger and bigger models get trained each year in the pursuit of better performance, albeit hurting reproducibility, since training costs quickly can become prohibitly expensive.

Adding to this challenge that the NLP community faces, it unfortunately is common for novel Machine Learning methods to leave implementation details unmentioned in their paper, when these implementation details might be key to achieving good performance, and as such might cast doubt to the relevance of the method.

In this work, we focus on the problem setting of learning an NLP task with a small dataset. Current state-of-the-art methods make use of a Transformer-based architecture ([**?**]), but as we will discuss in the related work section, even landmark papers cited thousands of times can have the reproducibility problems we mentioned. Specifically, the Transformer-XL method, widely regarded for its ability to model long-term dependencies in text and its SOTA performance, relies on on unmentioned data-augmentation techniques. In fact, when de-activating these techniques, independent repreductions found that Transformer-XL doesn't even match LSTM baselines, concluding that architecture choice might be less relevant in small-dataset regimes than data-augmentation designs. In this work, we set out to answer this question: when dealing with small datasets, **does data augmentation matter more than architecture design?** Can we get better performance increases from simpler LSTM-based models by applying different dropout techniques than by changing the architecture to an attention-based model?

3 Related Work

3.0.1 Transformer-XL architecture

Transformer architectures ([**?**]) have advanced the field of NLP by solving some key challenges that traditional RNN architectures faced, namely the difficulty of learning relationships between distant words because of diminishing gradients. Particularly when long-term dependencies are present in the dataset were working with (more than 200 context words), Transformer architectures can be expected to perform significantly better than LSTM or other RNN variants, and indeed have established state of the art results in many benchmarks. Transformers, however, have a strict limitation with regards to how back they can look in the text to search for context; they have a limited window for attention, which makes it so that dependencies longer than this window cannot be modelled.

Transformer XL ([**?**]) directly tackles this problem by merging the good things from both Transformers and RNNs by implementing recurrence in a Transformers design. Previous hidden states are kept when moving the context window, such that the model can decide to pay attention to previous hidden states as well as the current one. In theory, this novel architecture would allow transformers to have an unbounded context window, such that long-term dependencies in the data can be discovered by the model.

This architecture shows empirically very strong results, matching or surpassing state-of-the-art for several benchmarks. Surprisingly, not only do they show state-of-the-art results in datasets with long-range dependencies, but they also surpass state-of-the-art on the **One Billion Word benchmark, which doesn't preserve long-term dependencies in the data**, since it truncates the context. This observation is key, so let's discuss it in more detail.

Long-term dependencies happen in a text when, while trying to predict the next word, that word depends somehow on a token that occurred much earlier in the input text. For example, in the beginning of a book a character might be introduced by name, and by the end of the book that character might be mentioned again, so if we see the sentence "She ran into her friend from the start of the story, _", this will represent a very long-term dependency. The entire motivation of the Transformer-XL and related methods such as Compressive Transformers ([**?**]) or the Routing Transformer ([**?**]) is that their architectures are better suited for modelling this kind of dependencies.

The One Billion Word dataset, however, doesn't have these dependencies at all, because the context is truncated. The fact that the Transformer-XL achieves SOTA performance even on this dataset is a key observation for two reasons:

- It points to the architecture not being the only reason for getting such strong performance, since in the One Billion Word dataset the main feature of the architecture (very long dependencies modelling) is not used at all.
- It points to potential benefits of using this architecture in the Question-Answering domain where contexts are smal enough to fit into one segment's attention. This is the case in the SQuAD dataset.

3.0.2 Data augmentation

Data augmentation is a common technique when dealing with the small-data regime; When powerful models are trained on a small dataset, it is normal to experience overfitting. Data augmentation can help by artificially generating samples starting from real samples from the dataset. Many examples of data augmentation techniques are found in Computer-Vision literature, where flipping, cropping, scaling, rotating, or adding artificial noise improve generalization performance of algorithms ([**?**], [**?**]). In the NLP domain, we also see significant performance gains by using data augmentation in low-resources domains ([**?**]). Notice that dropout ([**?**]), when applied to the embedding layer of the model, acts as a data augmentation technique, similar to adding noise to images in the computer-vision domain.

3.0.3 Unmentioned Data-Augmentation in Transformer-XL

If the Transformer-XL cannot take advantage of long-term dependencies in the One-billion word dataset, then **how can it achieve state-of-the-art performance** on it, vastly surpassing other attention-based architectures? What sets them appart from other methods? This same question was asked by the community when several people were unable to reproduce the results from the paper in this dataset, even offering a bounty for anyone who could replicate their results.

It turns out that, without mentioning it in the paper, the authors made use of key data augmentation techniques to achieve such strong results in these benchmarks, especially the ones in which relatively little amounts of data is available. In particular, the authors made use of two types of dropout: discrete embedding dropout, which drops out entire word embeddings randomly from the context; and standard dropout on the input embeddings. These techniques improve performance drastically in low-data regimes by making the dataset noisier, which makes it less likely for the model to overfit the dataset, but in turn much more training time is required to achieve good performance, which is a key disadvantage over previous works, which conveniently went unmentioned in the paper.

4 Approach

In this paper, we implemented from scratch the Transformer-XL architecture, and experimented two kinds of data-augmentation on both the Transformer-XL and the baseline BiDAF (a RNN based on the LSTM architecture.) In this section, we thoroughly describe the Transformer-XL architecture that was implemented, and the dropout techniques that we implemented.

4.0.1 Baseline model

The core question of this study is whether dropout data augmentation techniques can also improve performance of LSTM-based architectures, so as our baseline we use the BiDAF model ([**?**]), which is an LSTM-based architecture. Notice that this is included in the starter code, so we don't implement it ourselves, though we do implement the data augmentation techniques.

4.0.2 Transformer-XL architecture

A main part of this project is reimplementing the Transformer-XL architecture to evaluate first-hand the differences in performance we get using different dropout data-augmentation techniques. I implemented the Transformer-XL model using the Pytorch-Lightning framework to make it easier to train with multiple gpus. Note that Pytorch Lightning is only a wrapper around Pytorch, so I still had to implement Transformer-XL from scratch.
Transformer-XL has two main features that distinguish it from the vanilla Transformer:

- **Segment-Level Recurrence with State Reuse.** In the vanilla Transformer architecture, the self-attention mechanism can only pay attention to the current context of tokens. The Transformer-XL architecture, on the other hand, is also able to look at the state of the previous segments of the text. In particular, let $s_\tau = [x_{\tau,1}, \dots, x_{\tau,L}]$ and $s_{\tau+1} = [x_{\tau+1,1}, \dots, x_{\tau+1,L}]$ be two consecutive segments of text, and $\mathbf{h}_\tau^n \in \mathbb{R}^{L \times d}$ be the hidden state for the τ-th segment s_τ, then the self-attention mechanism is modified as

follows:

$$\tilde{\mathbf{h}}_{\tau+1}^{n-1} = [\text{StopGradient}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}]$$
$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \tilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \tilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top}$$
$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}) \quad (1)$$

The $\circ$ denotes concatenation. Notice that we make use of the previous contexts $\mathbf{h}_{\tau}^{n-1}$ cached from previous segments, so this composes the recurrent architecture of the Transformer-XL.

- **Relative Positional Encodings.** When we reuse states from previous segments, we must include a new way of encoding positioning of the tokens relative the the current position. For example, there should be a way of distinguishing between a token from the current segment and a token from several segments before. To do this, the authors present a relative positional encoding scheme in which learnable parameters are introduced to encode the position of tokens. We refer the reader to the Transformer-XL paper to get the full details of the Relative Positional Encoding.

The original Transformer-XL paper only considered the task of language modeling, so to use this architecture for the SQuAD dataset we will adapt it to use the decoder architecture, and output two indices for the start and the end locations of the answer. The output head is the same one as the starter code for BiDAF, which outputs probabilities to each token of the context, and minimizes the negative-log-likelihood of the right answers.

4.0.3 Data-augmentation techniques to be considered

In this study we focus our experimentation on two Dropout techniques that greatly improved performance on the Transformer-XL architecture: discrete embedding dropout, and standard input embedding dropout.

- **Discrete embedding dropout.** In this technique, we apply dropout to entire word embeddings with some probability. The input to the transformer model is shaped [sequence_length, batch_size, hidden_size], so in the sequence_length dimension, we choose some rows and zero them completely. This serves to make the dataset more noisy, and acts as data-augmentation in a similar way as applying noise to images in an image dataset would.
- **Standard dropout for input embeddings.** Instead of dropping the whole embedding for a token, this technique drops out elements of the embeddings randomly.

5 Experiments

5.1 Data

We use the SQuAD 2.0 dataset ([**?**]) of question-answering. 130k training samples, 6k dev samples, and 6k test examples are provided. Each sample consists of a context paragraph, and a query for a question, whose answer is contained in the context paragraph. The output of the model should be the indices of the start and the end of the answer in the context paragraph.

5.2 Evaluation method

The main evaluation metric for this work is averaged EM and F1 scores on the SQuAD dataset for every method. Notice that the Transformer-XL method was meant for language modeling and not for question-answering, so further work could extend experimentation to other transformer-based question-answering methods.

5.3 Experimental details

We were unable to get access to our Azure project credits because on the Azure Portal the only subscription that showed up was "Microsoft Azure Sponsorship", which did not enable access to GPU instances. Because of this and in response to the time constraints and slow responses from the Azure support team, our team decided to use Google Colab Pro to run the experiments, which

enabled us to use GPUs for training, but meant that the length of training would be very limited. We would also like to note that being a one-person team made it additionally difficult to run experiments for longer.

Model hyperparameters for the Transformer-XL architecture are as follows:

- **Dimensionality of model** (dimensions of the embeddings and hidden dimension of each layer. As in the original paper, every layer has the same dimensions): 256.
- **Number of layers**: 6.
- **Number of attention heads**: 4.
- **Batch size**: 64.
- **Learning rate**: 0.00025.
- **Discrete embedding dropout rate**: 0.1 or 0 depending on the setting.
- **Standard dropout for input embeddings**: 0.1 or 0 depending on the setting.
- **Number of epochs**: 8. Unfortunately, after running for 12 hours our Colab runtime was disconnected for each experiment, which limited the length of our training.

5.4 Results

We report the train curves of our Transformer-XL implementation in Figure 1. Notice that the training losses for the baseline method without dropout and with discrete dropout of the input embeddings are essentially the same, and appear almost indistinguishable in the figure, whereas the loss for the standard embedding dropout is slightly higher. Since the original Transformer-XL doesn't report the dropout rate that they used, it is possible that using a 0.1 dropout rate is too low to see noticeable differences of using discrete dropout.

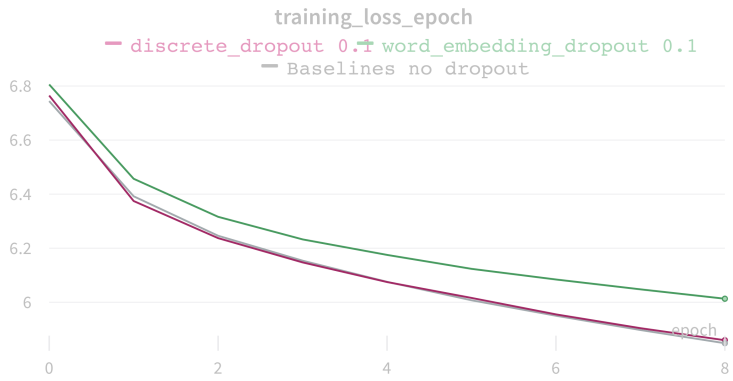

Figure 1: Training curves of Transformer-XL experiments with and without dropout.

Now we report the development-set performances of our Transformer-XL methods:

- **Transformer-XL No Dropout**: Dev NLL: 08.36, F1: 12.80, EM: 10.87, AvNA: 47.86
- **Discrete Dropout 0.1**: Dev NLL: 08.53, F1: 03.37, EM: 01.23, AvNA: 47.86.
- **Standard Embedding Dropout 0.1**: Dev NLL: 08.32, F1: 06.78, EM: 04.54, AvNA: 47.86

Now we report out dev-set performances on the baseline BiDAF model, modificated to make use of the discrete embedding dropout method: **Results**: : **Dev NLL: 2.94, EM: 57.217; F1: 60.393**

5.4.1 Comments on the quantitative results

The results for the Transformer-Xl architecture with or without dropout are much lower than expected. This can be explained by the fact that this is a very complex and big model, and the training time for these experiments simply was not enough to achieve reasonable performance. The use

of dropout accentuates this weakness even more, since it is known that using dropout trades-off better performance for longer training times. Furthermore, notice by the training loss curves that the models have not converged, which demonstrates that longer training times would directly benefit these models.
When looking at the BiDAF results though, we can see an improvement over the baseline of not using discrete dropout, which suggests that data augmentation techniques such as discrete embedding dropout are in fact promising for the small-data regime.

6 Analysis

When conducting the qualitative evaluation of the Transformer-XL model, it became very clear that the training times that we carried out in this project were simply not enough for the model to correctly learn the task. The answers provided by the model look largely random, and are clearly much worse than the BiDAF model, which provides more reasonable answers.

7 Conclusion

A key limitation of our work was access to resources to run the experiments for longer such that the transformer-xl models could actually achieve convergence. Since we were not able to do this, we cannot make claims comparing the Transformer-XL architecture to LSTM-based ones such as BiDAF, since the comparison would be unfair, because the BiDAF requires less compute time to reach convergence. What we can claim is that discrete embedding dropout seems to benefit LSTM-based architectures in small-data regimes. This is an encouraging finding that hopefully will direct future research to focus on data-augmentation details that might impact performance, and encourage other researchers to publish all their implementation details to foster reproducibility.