Dispatcher: A Message-Passing Approach To Language Modelling.

Alberto Cetoli †
alberto.cetoli@fractalego.io

Abstract

The transformer architecture has achieved state-of-the-art performance on language modelling. Nonetheless, a more efficient algorithm would allow for a larger number of tokens, thus a wider context and better grounding of the model's predictions. In this spirit, we introduce a more efficient layer type that aims to substitute self-attention for unidirectional sequence generation tasks. The system is shown to be competitive with existing methods: Given N tokens, the computational complexity is $\mathcal{O}(N \log N)$ and the memory complexity is $\mathcal{O}(N)$ under reasonable assumptions. The Dispatcher layer is seen to achieve comparable perplexity to self-attention while being more efficient[1].

1 Introduction

The introduction of self-attention (Vaswani et al., 2017) has produced a considerable surge of language models (Devlin et al., 2018; Liu et al., 2019; Yang et al., 2019; Lan et al., 2020). Originally, self-attention had been envisioned as a three elements algorithm (key, query, and value) to be applied onto an encoder-decoder framework for machine translation. It soon became evident that the Transformer architecture can successfully master the most relevant NLP tasks (Devlin et al., 2018) with some marginal modifications. One key application of self-attention has been language generation (Radford et al., 2019), where typically the model attempts to predict the next token given a limited window of prior elements. A full text can thus be generated word by word. Self-attention - bidirectional in nature - needs to be masked in order to avoid backward propagation of information.

Until a few years ago recurrent models (Sutskever et al., 2011; Graves, 2014; Merity et al., 2017; Melis et al., 2018) outperformed every other method for language modelling and generation. This has changed with the introduction of masked self attention (MSA) models, which have achieved the state-of-the-art in language generation, culminating in some unexpected results for multi-task zero-shot learning (Radford et al., 2019) as well as intriguing few-shot abilities (Brown et al., 2020).

The main argument of this work is to show that language modelling can efficiently rely on a message passing approach to perform, proposing a method that *does not leverage upon self-attention*. Instead, the system builds a tree-like structure of forward message passing weighed by *dispatching coefficients*. In the end, the Dispatcher architecture can generate texts as well as the original Transformer model, more efficiently. The main contributions of the paper are to introduce the novel algorithm as well as compare perplexity to the "standard" self-attention on the task of language modelling.

2 Model

The original Transformer architecture is composed of a number of self-attention, skip connection, and feed-forward layers. Given N tokens, the self-attention block has a computational and memory complexity of $\mathcal{O}(N^2)$ and is therefore problematic for long sequences. Here we propose to substitute each self-attention layer with a different algorithm.

Within the Dispatcher layer, information is pushed forward onto the next tokens in a recursive fashion. The algorithm is given a list of embeddings as *input*, with the aim to create *output* embeddings that contain a mixture of the tokens that precede them, without any leakage from the tokens that follow. The system achieves this goal by summing the tokens with themselves shifted by a power of two, iteratively. Each of these steps is labelled *shift and sum* in Fig. 1.

In the pseudo-code shown in Alg. 1 the *dis-*

---

[1] The code is available at https://github.com/fractalego/dispatcher

† Work done while at QBE Europe.

Algorithm 1: The Dispatcher Layer Algorithm

$c \leftarrow$ Sigmoid(Linear1(input));
$c \leftarrow c \odot$ mask;
$V \leftarrow$ Linear2(input);
for $row = 0 \rightarrow log_2 N - 1$ **do**
    $V \leftarrow V + c[row] \odot$ RollRight(V, 2^{row});
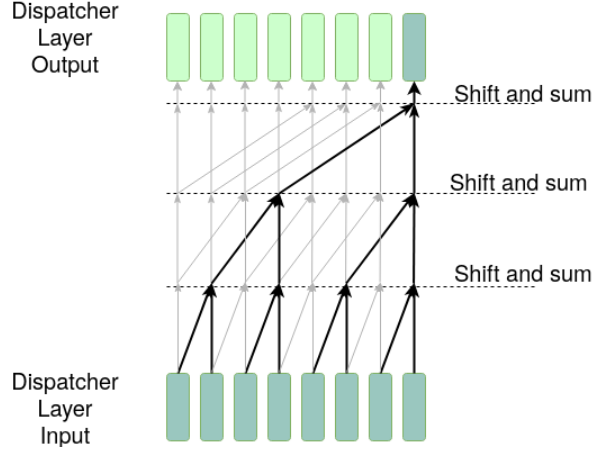**end**
output $\leftarrow$ Linear3(V);

Figure 1: A representation of how information is passed from the input tokens to the output within the Dispatcher Layer. At every vertex of this directed graph the embeddings are summed together, each sum weighed by the *dispatching coefficients*. These weights determine how much of the message from the left needs to be passed onto the right. For clarity the paths to the last output item are painted in a darker color.

patching coefficients are written as $c \in \mathbb{R}^{N \times \log_2 N}$, whereas $V \in \mathbb{R}^{N \times d}$ is the tensor containing the hidden states used as a working memory in the main loop, with embedding dimension d. The Linear functions are dense layers, while RollRight shifts the tokens to the right.

The message coming from the prior tokens follows a binary tree structure, as depicted in Fig. 1. The sum is weighed by the *dispatching coefficients*, which effectively decide whether information coming from the left of the tree should propagate further, and by what amount. These weights are computed through a dense layer applied to the original tokens. A constant mask is applied to the tensor c after it has been computed to avoid leakage after the RollRight operation.

The algorithm presented above describes a single-head unit. As with self-attention, this layer can be split into a set of Dispatcher heads to improve performance. The number of heads then becomes another hyper-parameter to tune during training. Finally, if the number of input embeddings is not a power of two, the loop stops when the shift value is greater than the input length.

2.1 Dispatcher Dropout

A quick modification of Alg. 1 can introduce an effective dropout by randomly skipping a *shift and sum* step in training with a probability given by a dropout value between 0 and 1. Notice that in this procedure dropout makes the algorithm quicker, albeit with the same computational complexity.

2.2 Computational complexity

The creation of the *dispatching coefficients* is linear in time, as a dense layer is a applied to every input token. The main algorithm repeats $\log_2 N$ times a weighted sum. If d is the dimension of the embeddings, the computational complexity is $\mathcal{O}(d \, N \log N)$.

2.3 Memory complexity

The system computes the dispatching coefficients in every layer with a space complexity of $\mathcal{O}(N \log N)$. In addition, the algorithm uses at every step a set of embeddings V with complexity $\mathcal{O}(N \times d)$. In a typical scenario $d \gg \log_2 N$, yielding an effective asymptotic linear memory consumption $\mathcal{O}(N \times d)$.

3 Evaluation

3.1 Datasets

The algorithm is evaluated on the following datasets: PTB (Mikolov and Zweig, 2012), WikiText2 and WikiText103 (Merity et al., 2017), and One Billion Word (Chelba et al., 2013). A simple pre-processing step uses the special token <EOS> to indicate the end of each sentence.

Among the sets, PTB and WikiText2 are the smallest, with only 4.9MB and 11MB of text data for training respectively. This is to be compared to the 515MB training set of WikiText103 and 3.9GB of 1BW. While the larger dataset, 1BW only models short-term dependency because the sentences have been shuffled. An additional corpus called OpenWebText (Cohen and Gokaslan, 2020) is used to train the larger Dispatcher model. This set was created as an open alternative to the one used when training GPT-2 (Radford et al., 2019) and consists of about 40 GB of text data.
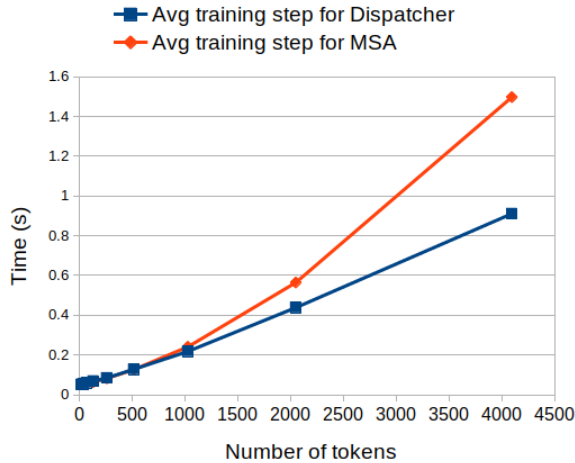
Figure 2: Average time in seconds for a single training step on WikiText2 as a function of the number of input tokens. Both models are trained on the same single-GPU instance. The asymptotic behavior appears remarkably different.

3.2 Training

At first, the Dispatcher algorithm is compared against a masked self-attention model. Rather than trying to optimize for the MSA and the Dispatcher separately, we choose an identical set of hyperparameters (embedding size, number of heads, layers) and compare their perplexity. While this approach might not give the best results for each model, it helps to show that the two algorithms perform similarly under similar conditions.

Secondly, a slightly bigger Dispatcher model (Sec. 3.4) is trained for *one epoch only* on the OpenWebText corpus. The goal is not to achieve state-of-the-art results, rather to prove that the proposed architecture can reach comparable perplexity to Transformer-based models. In the spirit of simplicity, we use a single head for all the models, which are trained on the same single-GPU machine. All the models are implemented in *PyTorch* (Paszke et al., 2019).

3.3 Masked self-attention (MSA) and Plain Dispatcher

These models have embedding and inner dimension size 512, 6 layers, and only 1 head. The training batch size is 20 and dropout is set to 0.2, using 512 tokens. The only difference between the two models is the self-attention/Dispatcher layer. Tokenization is done using a pre-trained WordPiece tokenizer made available by HuggingFace (Wolf et al., 2020). On training and evaluation the vo-

cabulary is further restricted on each dataset to improve performance, as a consequence the number of parameters changes depending on the relevant dataset's vocabulary size.

3.4 Dispatcher after OpenWebText

This model has a single head, 480 embedding and inner dimension size, and 12 layers with a minibatch size of 5 and no dropout. A BPE tokenizer - pre-trained on OpenWebText - is used. The number of tokens used for this model is 1024. First pre-trained on OpenWebText, the model is then fine-tuned onto the relevant sets.

3.5 Discussion

The MSA model and the Plain Dispatcher are evaluated against four different datasets, as shown in the first two rows of Table 1. The results are quite similar, with the Dispatcher architecture seen performing better on the smaller sets PTB and WikiText2. This is arguably due to the model having fewer parameters and being less prone to overfitting. Conversely, the larger MSA model wins on WikiText103. The Dispatcher overtaking MSA on 1BW is more challenging to explain in terms of model size and seems to suggest its enhanced ability to model short-term dependencies, at least in this one-headed configuration.

A striking difference between the MSA and the Dispatcher is however shown in Fig. 2, which plots the average time for a single training step as a function of the number of input tokens. While the recorded times are configuration-specific, the asymptotic behavior looks radically different, suggesting the Dispatcher architecture as a better candidate for longer sequences.

A single epoch of training onto the OpenWebText dataset boosts the Dispatcher performance into competitive results for a model of this size, after fine-tuning on the relevant corpus. This is shown in the third row of Table 1, presenting our top results.

The rest of Table 1 is a showcase of the most recent self-attention based models. Notably, our results on PTB and WikiText2 are among the best in the literature, surpassing the results in (Wang et al., 2019) which are obtained by fine-tuning a pre-trained BERT model. This is most likely due to the OpenWebText corpus being a better set for language generation than BookCorpus (used by BERT), but it bodes well for the algorithm presented here that it can compete against models with one order of magnitude more parameters. The last

Model Type	PTB	WikiText2	WikiText103	1BW
Masked Self-Attention (18M / 30M / 39M / 41M)	40.58	55.05	22.56	66.52
Plain Dispatcher (17M / 27M / 36M / 38M)	35.40	50.23	24.39	53.32
Dispatcher after OpenWebText (59M)	18.95	22.74	20.38	36.76
(Fan et al., 2020) (44M)	-	-	22.4	-
(Wang et al., 2019) (395M)	31.34	34.11	20.42	-
(Tay et al., 2021) (100M)	-	-	-	21.5
(Dai et al., 2019) (257M / 0.8B)	-	-	18.3	21.8
(Radford et al., 2019) (1.5B)	35.7	18.34	17.48	42.16
(Shoeybi et al., 2020) (355M)	-	-	19.31	-
(Shoeybi et al., 2020) (8.3B)	-	-	10.81	-

Table 1: Top: The Dispatcher architecture is evaluated concurrently with a masked self-attention model yielding similar results. Bottom: The Dispatcher pre-trained on OpenWebText compared to some recent results achieved using a variant of the Transformer architecture. All the results refer to the test perplexity.

three rows relate to zero-shot results. Omninet's impressive result (Tay et al., 2021) is achieved by extending self-attention to all tokens in all the layers, while here the dispatcher layer is only aware of the embeddings within a single layer.

4 Related works

The way information is funneled to higher layers in Fig. 1 is reminiscent of convolutional neural networks (CNN) (Liu et al., 2020; Gehring et al., 2017). It is especially evocative of *dilated convolutions* as presented in (van den Oord et al., 2018). While similar, the method presented here is not technically a convolution, which by definition requires the same operator being translated over the input elements. In this paper the dispatching coefficients are *local* to the tokens.

Another way to visualize the Dispatcher algorithm is as a set of overlapping Recursive NN acting on binary trees (Goller and Kuchler, 1996; Socher et al., 2011) which share parameters where the trees overlap. It is however important to keep in mind that the *shift and sum* iteration only performs a weighted sum of the input embeddings, achieving competing performance only when repeated within a multi-layer structure.

The computational cost of large models has become a source of concern in terms of scalability as well as energy consumption (Strubell et al., 2019). For this reason, a growing number of approximations (Wang et al., 2020; Kitaev et al., 2020; Zaheer et al., 2021; Choromanski et al., 2020; Zhai et al., 2021) have appeared in the literature, suggesting modifications to the main self-attention layer.

These approximations tend to leverage linear algebra properties to speed up calculations, capturing the essence of the Transformer architecture into more efficient algorithms. In many cases the approximation makes the model irreducibly bidirectional, thus hindering language generation tasks.

More recently, inductive biases alternative to self-attention have been shown to achieve comparable results on language tasks using the Fourier Transform in place of the MSA layer (Lee-Thorp et al., 2021) and on vision tasks by means of spatial MLPs (Yu et al., 2021).

Finally, the concept of message passing is understood to describe Graph Convolutional Networks (Kipf and Welling, 2017; Geerts et al., 2020) and by extension the self-attention mechanism in the Transformer architecture. The Dispatcher algorithm makes message passing explicit by keeping the routing topology constant while relying on the coefficients to distribute the message within a set of binary trees.

5 Conclusions

A novel architecture dedicated to language modelling is introduced and shown to achieve comparable perplexity with self-attention based models, requiring less computational and memory resources. A larger number of tokens can allow for a wider context window and more detailed grounding of the model's predictions.

Finally, low perplexity in the task of language modelling is often predictive of high-quality text generation. This intriguing possibility will be pursued in a future work.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Rethinking attention with performers.

Vanya Cohen and Aaron Gokaslan. 2020. Opengpt-2: Open language models and implications of generated text. *XRDS*, 27(1):26–30.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2020. Accessing higher-level representations in sequential transformers with feedback memory.

Floris Geerts, Filip Mazowiecki, and Guillermo A. Pérez. 2020. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. *CoRR*, abs/2004.02593.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR.

C. Goller and A. Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352 vol.1.

Alex Graves. 2014. Generating sequences with recurrent neural networks.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR 2017*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2021. Fnet: Mixing tokens with fourier transforms.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Zhouyong Liu, Shun Luo, Wubin Li, Jingben Lu, Yufan Wu, Chunguo Li, and Luxi Yang. 2020. Convtransformer: A convolutional transformer network for video frame synthesis. *CoRR*, abs/2011.10185.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*.

Stephen Merity, Caiming Xiong, James Bradbury, and R. Socher. 2017. Pointer sentinel mixture models. *ArXiv*, abs/1609.07843.

Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. 2018. Parallel WaveNet: Fast high-fidelity speech synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926. PMLR.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning

library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-lm: Training multi-billion parameter language models using model parallelism.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 129–136, Madison, WI, USA. Omnipress.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1017–1024, Madison, WI, USA. Omnipress.

Yi Tay, Mostafa Dehghani, Vamsi Aribandi, Jai Prakash Gupta, Philip Pham, Zhen Qin, Dara Bahri, DaCheng Juan, and Don Metzler. 2021. Omninet: Omnidirectional representations from transformers. In *ICML 2021*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Chenguang Wang, Mu Li, and Alexander J. Smola. 2019. Language models with transformers. *ArXiv*, abs/1904.09408.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame,

Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32, pages 5753–5763. Curran Associates, Inc.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. 2021. Metaformer is actually what you need for vision.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2021. Big bird: Transformers for longer sequences.

Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. 2021. An attention free transformer.

29