Stitching Neural Networks in the NLP domain

Robert Adrian Minut
minut.1942806@studenti.uniroma1.it

Abstract

We explore the properties of the GPT-2 architecture when model stitching is performed, with or without additional learned parameters. In the experiments we found that the head of the transformer, which projects learnt features onto dictionary word pieces, plays a big role in how the resulting model performs. Surprisingly, even without adding new parameters we can stitch two models together given certain conditions.

1 Problem statement

An intriguing method that has recently garnered attention is the practice of "model stitching"[1] or, more generally, the technique of substituting model weights.

Unlike traditional model fusion or ensemble approaches, weight substitution involves replacing the learned parameters of one model with those from another while maintaining the same underlying architecture.
This novel approach challenges conventional wisdom by exploring the notion that it's not always necessary to build larger, more complex models to achieve improved performance. Instead, the judicious transfer of knowledge between models with different strengths offers a promising avenue for advancement.

The idea behind it is that models encode important information about the inputs through internal representations (intermediate outputs), which are then used to perform a certain task, i.e. classification. We can look at a network as a set of weights which are used to calculate these representations. So, we should be able to transfer representations from one model to another by simply substituting its weights. These representations may not actually be compatible with the other model. In this case we may introduce a stitching layer which acts as a translator.
If we define $A_{<i}$ as all layers before depth i of the neural network A, $B_{>j}$ as all layers after depth j of the network B, and S is a stitching layer, then $C = B_{>j}(S(A_{<i}))$ represents the stitched network.

This report embarks on a comprehensive exploration of weight substitution within the context of GPT-2. We delve into the rationale underpinning this technique and examine the actual impact of weight substitution on GPT-2's performance metrics, analyzing whether this unconventional approach yields benefits in terms of model effectiveness.

Note: when we introduce additional parameters through a stitching layer, we refer to this operation as "**stitching**". Instead, when we perform the substitution of weights without adding a stitching layer, we call this operation "**merge**".

1.1 Real-world applications

The authors of [7] proposed interesting practical applications of model stitching to improve, at inference time, the responsiveness of a server hosting an AI model. Their method consists of stitching different model architectures, and deciding at inference time which path to take through the various models. This allows, for example, to go from a bigger sized model to a smaller sized one in order to finish the prediction's computation earlier.

Another interesting application is to transfer the learned knowledge, on a specific task or data distribution, from one model to the other.

2 Related work

The authors of [5] conjectured that good Vision Neural Networks may learn similar **representations**. But different networks may encode the information differently, so the representations of two networks may not be compatible. The authors' hypothesis is that by **permuting** and **interpolating** the feature channels we should be able to obtain compatible representations. These two operations can be performed through a convolutional or linear layer, without non-linear activation functions. The work of [3] goes more in-depth to show how stitching two models at different depths can have an important impact on the performance of the resulting models.

In [7] the authors discuss how we can stitch models from different architectures, but their work is limited to the Vision domain. The same applies to the others, whereas we study how to best adapt this approach to models in the Natural Language Processing (NLP) domain.

3 Datasets and benchmarks

TinyShakespeare [4] This dataset was used to train small-scale models which have a similar architecture as GPT-2 but resized for character-level inference.

OpenWebText [2] This dataset was used to train LLMs with the same architecture of GPT-2 Small. Thet total size is around 17 million documents collected from internet pages dating from 2005 to 2020.

Benchmark To evaluate the model we used the validation loss, which takes into account the perplexity of the language model on a set of samples.

4 Existing tools, libraries, papers with code

For weight substitution we simply load the weights using the PyTorch library, perform the necessary modifications and then load the weights into the model.

For model stitching we used parts of the code by [7], which shows how to initialize and train stitching layers that translate the feature space of one model to the other's, and adapted it to the GPT-2 architecture.

5 State-of-the-art evaluation

Stitched neural networks are usually evaluated on:

1. the loss of the model;
2. a downstream task;
3. a reconstruction loss, to see whether the stitch layer is correctly mapping the feature space of one model to the other's.

6 Setup and Evaluation

Datasets We used the Tiny Shakespeare and OpenWebText datasets (Section 3) to evaluate the stitched or merged models.

Evaluation Protocol We measured the validation loss L of the model after the stitching or the merge operation is performed, and calculated the loss penalty as: $L_{\text{penalty}} = (L_{\text{base}} - L) \,/\, L_{\text{base}}$, where L_{base} is the loss of the original model.

GPT-2 Character Level This tiny version of GPT-2 has around 10.65 millions parameters, and its vocabulary size is 65 token ids, which represents the ASCII alphabet, digits and punctuation. We used this toy architecture to perform a series of stitching experiments.

GPT-2 Small GPT-2 Small has around 123.59 million parameters and a vocabulary size of 50304 token ids. These tokens encode both Unicode characters and pieces of words obtained through Byte Pair Encoding (BPE). We used these models to test whether our understanding of what is happening when merging or stitching models at larger scales is correct.

Optimizers We used as optimizers AdamW [6] and SGD with momentum and weight decay. Adam (short for Adaptive Moment Estimation) is an optimization algorithm commonly used for training artificial neural networks. It combines techniques from stochastic gradient descent (SGD) and momentum methods to efficiently update model weights during training by adapting learning rates for each parameter individually. AdamW introduces weight decay in Adam, whereas before it was implemented through L_2 regularization, which was demonstrated to perform worse.

GPT Head Experiments We found that the head of the GPT model plays an important role for the performance of a stitched network. We define a stitched model with its head preserved as $C = A_h(B_{i \leq j < h}(S(A_{<i})))$, where:

1. A_h is the head of model A;
2. $B_{i \leq j < h}$ are layers from model B from depth i up to right before the head of the model;
3. S is the stitching layer;
4. $A_{<i}$ are layers from model A up to depth i.

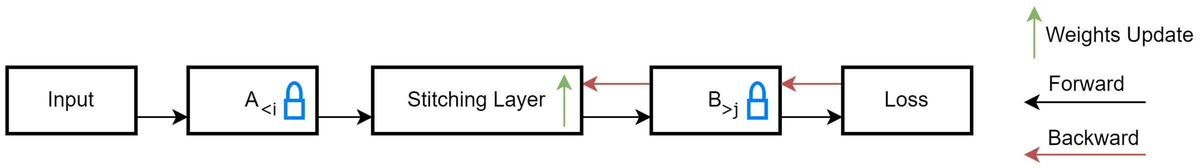

Figure 1: High level view of a stitched neural network.

7 Results

7.1 GPT-2 Character Level Results

GPT-2 character-level models are toy examples trained on the Tiny Shakespeare dataset. They allowed us to perform extensive experiments on the behaviour of stitched or merged models.

7.1.1 Different Initialization

We trained multiple models using distinct seeds for random generation and then tried to merge them at different levels. In Fig. 3 we can see that the performance loss is very high when using AdamW as an optimizer, but we found that preserving the head weights of the first model when substituting the weights of the second one greatly improves the performance. Also, note that in this case the loss penalty is clearly linearly dependent to the percentage of layers we substitute.
The performance loss is almost none when merging SGD initializations (Fig. 5), but this may be due to the small scale of the dataset.
Through stitching, we can improve the performance even when merging AdamW initializations (Fig. 4), and as seen for the merging operation we can get even better results by keeping the original head weights of the model.

7.1.2 Different Optimizer

We also trained models using different optimizers (Fig. 2) and tried to merge them, the results can be seen in Fig. 6. We can improve the performance using stitching and by preserving the head weights (Fig. 7), as in the previous experiment.

7.2 GPT-2 Small Results

GPT-2 small models were trained on OpenWebText, and they are representative of larger-scale models.

7.2.1 Different Initialization

We first trained two models using distinct seeds. Then we performed the merge (Fig. 8) and stitch (Fig. 9) experiments. In (Fig. 8) we can see that even without finetuning, we can merge models and have a low loss penalty.

7.2.2 Different Optimizer

We trained models using different optimizers. Then we performed the merge (Fig. 10) and stitch (Fig. 9) experiments.

8 Discussion

The phenomenon of two models, trained using different initializations, exhibiting almost no performance loss when merged together without any fine-tuning is intriguing and highlights some interesting aspects of the GPT architecture:

1. Transferability of Knowledge: Our results suggests a good degree of transferability of knowledge between models with different weight initializations. It implies that the architectural patterns and learned representations in both models are highly compatible, allowing them to seamlessly work together.

2. Robustness of Neural Networks: Neural networks, particularly large and deep models like GPT-2, can exhibit a certain degree of robustness. This means that their performance can be resilient to certain perturbations, such as weight initializations, and still perform well. Understanding this robustness can have implications for model generalization and training stability.

3. Implicit Regularization: Weight initialization methods are known to have an implicit regularization effect on training. The compatibility observed between models with different initializations may reflect the regularization properties of those initializations. This result could inspire further research into understanding the impact of different initialization strategies.

4. Architectural Redundancy: It's possible that the architecture of GPT-2, along with the

self-attention mechanism and layer normalization, contains a degree of redundancy or flexibility that allows it to adapt effectively to different weight distributions. This architectural robustness can be beneficial in practical applications where model deployment or adaptation is needed.

5. Practical Implications: This discovery has practical implications, particularly in scenarios where model deployment needs to be efficient and rapid. If models with different initializations can be stitched together seamlessly and without performance loss, it could simplify the deployment pipeline, reduce training costs, and expedite model adaptation to specific tasks.

Another interesting result is the importance of the Head of the GPT architecture for language modeling. We show that preserving or changing the weights of the Head of the model during the stitch or merge operation can greatly affect the performance. We think that GPT models are very sensible to changes in the weights of the Head, while more robust to changes in the middle layers.

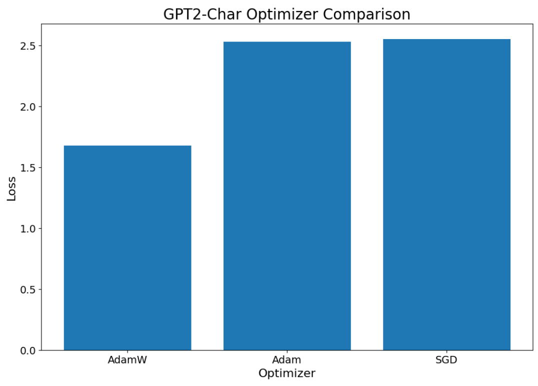

Figure 2: GPT-Char Loss when using different optimizers.

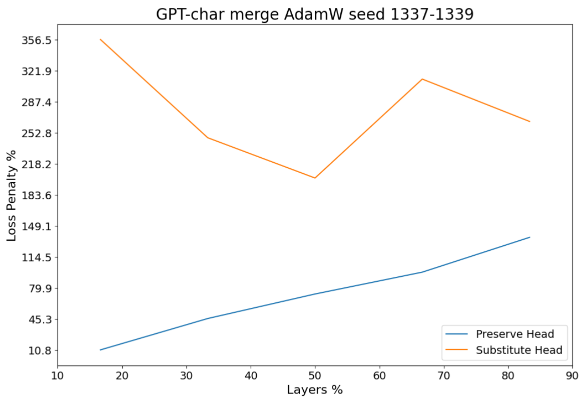

Figure 3: GPT-Char Loss Penalty when **merging** tiny GPT models trained using AdamW with different initializations.

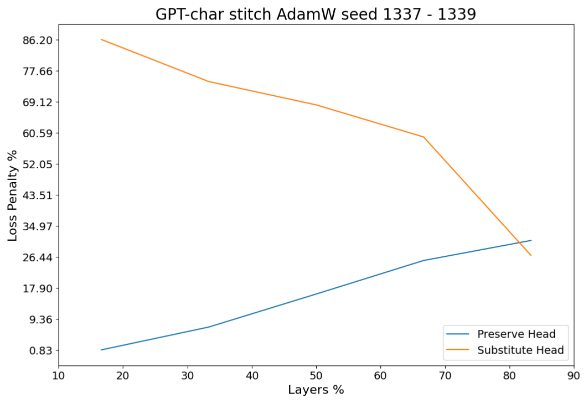

Figure 4: GPT-Char Loss Penalty when **stitching** models trained using AdamW with different initializations.

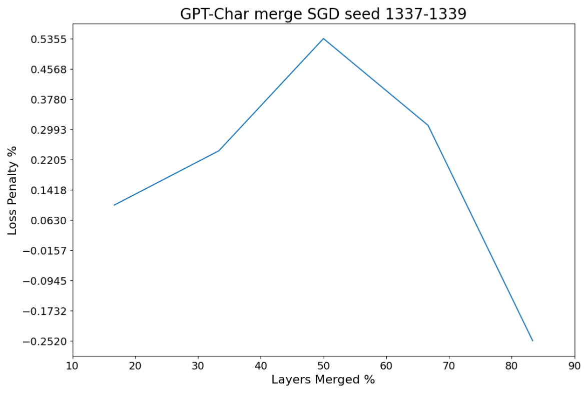

Figure 5: GPT-Char Loss Penalty when **merging** tiny GPT models trained using SGD with different initializations. Note that we have a very low loss penalty, and in some cases the resulting model even performs better than the source.

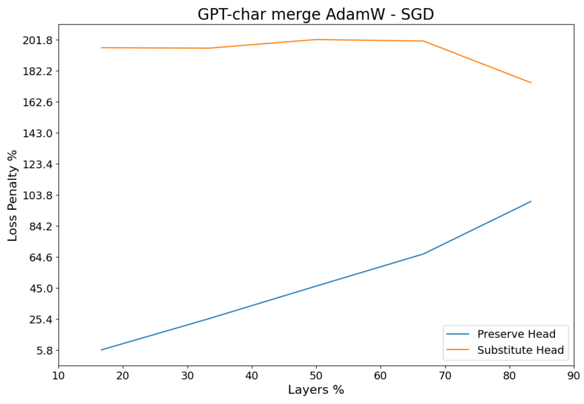

Figure 6: GPT-Char Loss Penalty when **merging** models trained with different optimizers.

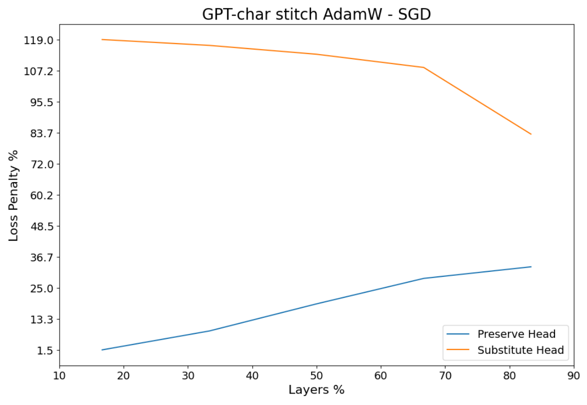

Figure 7: GPT-Char Loss Penalty when **stitching** models trained with different optimizers.

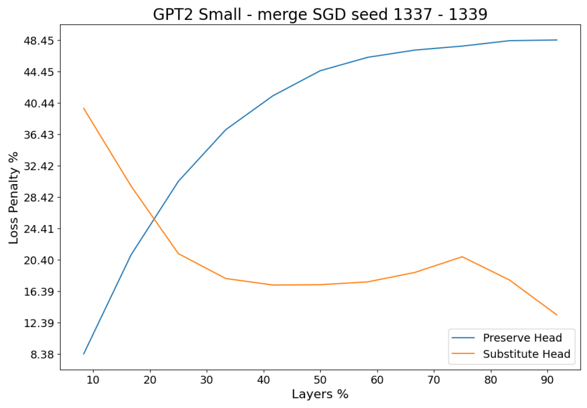

Figure 8: GPT-2 Small - Loss Penalty when **merging** models trained with different seeds.

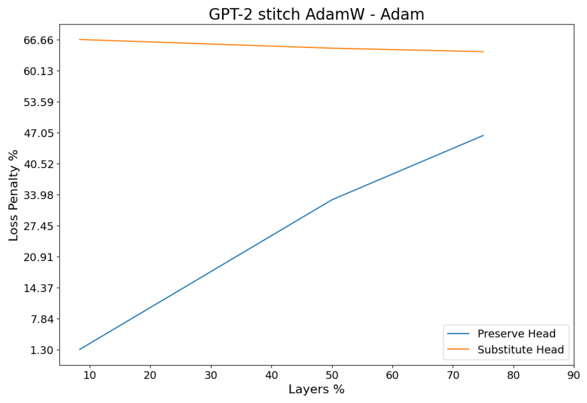

Figure 9: GPT-2 Small - Loss Penalty when **stitching** models trained with different seeds.

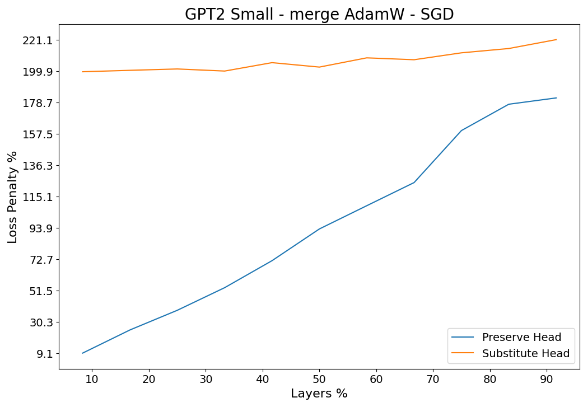

Figure 10: GPT-2 Small - Loss Penalty when **merging** models trained with different optimizers.

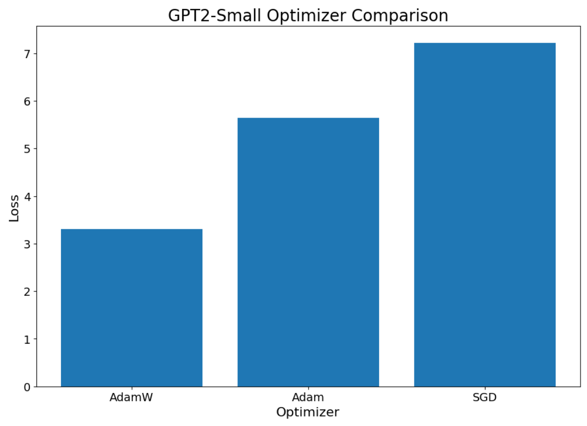

Figure 11: GPT-2 Small - Loss when using different optimizers.

8.1 Limitations

Evaluating models after stitching or merging their parameters is very challenging. The main concerns are:

1. Transferability: The success of weight transfer depends on the similarity of architectures and pre-training objectives between the source and target models. If the models have substantial architectural differences or were trained with different objectives, the effectiveness of weight transfer may be limited.

2. Fine-Tuning Challenges: While some stitched models may perform reasonably well without fine-tuning, others may require fine-tuning to align the transferred knowledge with specific tasks. Identifying when and how to fine-tune a stitched model can be non-trivial and may require additional data and computational resources.

3. Task-Specific Performance: Stitched models may excel in certain tasks but underper-

form in others. Evaluating their performance across a wide range of tasks can be challenging, and it may be necessary to assess their suitability for each task individually.

4. Computational Resources: Training large language models and evaluating their performance can be computationally intensive. Stitched models, which often involve multiple large models, can exacerbate these resource requirements, making them less accessible for many researchers and organizations.

5. Interpretability and Explainability: Stitched models may be more complex and difficult to interpret due to their composite nature. Understanding the contributions of each source model to the overall performance can be challenging, and this lack of interpretability can be a significant limitation.

6. Lack of Theoretical Foundation: although we shed some light on the properties of stitched models, the theoretical underpinnings for the effectiveness of these models are not as well-established as other machine learning techniques. This can make it difficult to predict when and why stitching will work effectively.

While this phenomenon is fascinating, it may not be universally applicable to all GPT models. Further research is needed to understand the boundaries and conditions under which this holds true. It's also important to consider the scale and complexity of the models involved, as the results may vary.

9 Conclusions

The fact that two models trained with different initializations can be stitched together with a small performance loss highlights the robustness and transferability of knowledge within this family of models. We also show that the Head of the Language Model plays an important role in how the stitching should be performed. Careful consideration should be given to the specific use cases and scenarios where this technique is most effective, but also to how it is applied.

References

[1] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 225–236. Curran Associates, Inc., 2021. 1

[2] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019. 2

[3] Adriano Hernandez, Rumen Dangovski, Peter Y. Lu, and Marin Soljacic. Model stitching: Looking for functional similarity between representations, 2023. 2

[4] Andrej Karpathy. Tiny shakespeare. https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt. 2

[5] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence, 2015. 1

[6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. 2

[7] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Stitchable neural networks. In *CVPR*, 2023. 1, 2