On the Effect of Pre-training for Transformer in Different Modality on Offline Reinforcement Learning

Shiro Takagi *
Independent Researcher
takagi4646@gmail.com

Abstract

We empirically investigate how pre-training on data of different modalities, such as language and vision, affects fine-tuning of Transformer-based models to Mujoco offline reinforcement learning tasks. Analysis of the internal representation reveals that the pre-trained Transformers acquire largely different representations before and after pre-training, but acquire less information of data in fine-tuning than the randomly initialized one. A closer look at the parameter changes of the pre-trained Transformers reveals that their parameters do not change that much and that the bad performance of the model pre-trained with image data could partially come from large gradients and gradient clipping. To study what information the Transformer pre-trained with language data utilizes, we fine-tune this model with no context provided, finding that the model learns efficiently even without context information. Subsequent follow-up analysis supports the hypothesis that pre-training with language data is likely to make the Transformer get context-like information and utilize it to solve the downstream task.

1 Introduction

The past few years have witnessed the tremendous success of pre-trained Transformer-based models [1] on versatile natural language processing (NLP) tasks. This is because pre-training with a large corpus enables Transformer-based models to gain rich universal language representations transferable to vast downstream tasks [2, 3, 4]. Remarkably, recent studies demonstrate that Transformers pre-trained with language data can efficiently solve even non-NLP tasks [5, 6, 7, 8, 9, 10]. Because of its versatility, it has been suggested that Transformers pre-trained with a large corpus could be called a kind of *universal computation engines* [5].

One area worthwhile attempting to study the benefits of pre-trained Transformer is reinforcement learning because the applicability of pre-training to this area still has room to be explored [11, 12, 13]. In this vein of research, Reid et al. studied the effect of pre-training of Transformers on offline reinforcement learning (offline RL) tasks [9]. Their study demonstrates a surprising contrast that pre-training with image significantly deteriorates performance, while that with language data does not negatively affect downstream tasks, rather it even improves performance for some datasets.

However, it is not yet known how the presence or absence of pre-training and its content leads to differences in the performance of downstream tasks. What information does/doesn't the model pre-trained on language data (*language-pre-trained model*) leverage to solve downstream tasks, and why does the model pre-trained on image data (*image-pre-trained model*) result in catastrophic performance? In this paper, we approach these questions by analyzing the inside of the Transformer

*https://t46.github.io/
The code for the experiments is available at https://github.com/t46/pre-training-different-modality-offline-rl.
The LaTeX source is available at https://github.com/t46/paper-pre-training-different-modality-offline-rl.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

that is fine-tuning to Mujoco [14] offline RL tasks. Analyzing the internals of Transformer, we study how randomly initialized, language-pre-trained, and image-pre-trained models acquire different representations, and what the differences could be attributed to. Our contributions are as follows:

- We study the internal representation of pre-trained and randomly initialized Transformers and find that the pre-trained models largely change their representation, while they encode less information of data during fine-tuning.
- The analysis of parameter change and gradient shows that the pre-trained models do not change parameters much perhaps due to gradient confusion and that the performance of the image-pre-trained model might be partly affected by gradient clipping on the large gradient.
- We examine context-dependence by fine-tuning models with giving no context information, finding that the language-pre-trained Transformer efficiently learns even without context and this could be the case by replacing just a single Transformer block with a pre-trained one.
- Subsequent analysis of the change in how far away the model process the tokens of input supports the possibility that some contextual equivalent information has been positively transferred from language pre-training.

2 Related Works

The study of transferability of neural representation has been one of the most important research topics in machine learning research. A bunch of studies have investigated how pre-training influences the downstream tasks, such as computer vision tasks [15, 16, 17, 18] and NLP tasks [19, 20, 21]. Recently, several studies have investigated the transferability of Transformers to various NLP tasks since understanding its high transferability is of great interest. Previous research reveals that Transformer representation encodes some syntax [22], semantics [23], cross-lingual information [24], and world knowledge [25]. Also, it has been pointed out that Transformers obtains not only language-specific information but also more abstract inter-token dependencies [26, 27, 28]. This finding suggests that pre-trained Transformers may apply to a wider variety of tasks than just NLP tasks.

In recent years, studies have emerged that apply the Transformer to tasks other than NLP tasks, reporting the versatility of the Transformer even in such multi-modal cases. Such prior works have shown that language-pre-trained Transformers can efficiently learn image classification [5], symbolic mathematics [6], simple arithmetic [4], reinforcement learning tasks [7, 8, 10, 9], and more generally, sequence modeling problems [5]. Our study is in line with the research that examines the applicability of pre-trained Transformers on reinforcement learning.

Most prior works applying pre-trained Transformer on reinforcement learning explicitly express goals, observations, or actions in a language [29, 7, 8, 10]. On the other hand, the work of [9] applies pre-trained Transformers on the offline RL tasks that language is unrelated to (e.g. Mujoco [14]) and still reports that pre-training is helpful. This study is interesting because it suggests that language-pre-trained Transformers seem to exploit some similarity between the structure of natural language data and trajectory data, which is more abstract and fundamental than linguistic information. Also, exploring the applicability of pre-trained Transformers, which have already been successful, is of great engineering significance since pre-training in offline RL has room for improvement compared to other modalities. Examining the internals of Transformers, we further delve into the findings from this study by Reid et al. [9] and explores the applicability of Transformers to offline RL.

3 Background

3.1 Transformer

Transformers are sequence-to-sequence models composed of stacked identical blocks, called *Transformer blocks*. Each block consists of self-attention [1], multi-layer perceptron, skip-connection [30], and layer normalization (layer norm) [31]. Denoting the linear projection of input sequence of length n to query, key, values of length m by $Q \in \mathbb{R}^{n \times d_Q}$, $K \in \mathbb{R}^{m \times d_K}$, and $V \in \mathbb{R}^{m \times d_V}$, respectively, the self-attention is $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_K}}\right)V$. We use GPTs [32], and hence the autoregressive language model, where queries head to only keys before their position.

3.2 Offline Reinforcement Learning and Decision Transformer

Reinforcement learning is the problem to learn to make optimal decisions in a dynamic environment. For ease of mathematical handling, the environment is usually described as a Markov decision process (MDP). The MDP consists of states $\boldsymbol{s} \in \mathcal{S}$, actions $\boldsymbol{a} \in \mathcal{A}$, transition probability $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, and reward function $r(\boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The interaction of the agent in the environment is represented as K length trajectory, or sequence of state, action, and reward $(\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}_1, \boldsymbol{a}_1, r_1, ..., \boldsymbol{s}_K, \boldsymbol{a}_K, r_K)$, where $\boldsymbol{s}_t$, $\boldsymbol{a}_t$, and r_t are state, action, and reward at time step t. The goal is to find an optimal policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ to maximize the expected cumulative rewards $\mathbb{E}[\sum_{t=0}^{K} r_t]$ for the trajectory.

Offline RL aims to achieve objectives of reinforcement learning from the trajectory data collected by some policy [33]. Because offline RL is purely characterized by trajectory data, several studies have proposed formulating offline RL as a sequence modeling problem [34, 35]. Decision Transformer is a seminal work that attempts to use a causal transformer (in other words, GPT architecture or decoder of BERT [2]) to solve such a sequence modeling problem [35]. In Decision Transformer, input data is a sequence $\tau = (\hat{R}_0, \boldsymbol{s}_0, \boldsymbol{a}_0, \hat{R}_1, \boldsymbol{s}_1, \boldsymbol{a}_1, ..., \hat{R}_K, \boldsymbol{s}_K, \boldsymbol{a}_K)$ of length K, where return-to-go $\hat{R}_t = \sum_{i=0}^{t-1} r_i$ is the cumulative reward until time step t. The previous studies [9, 35] employ the above problem set up to discuss the applicability of Transformer for offline RL. We also follow these studies and use the setup for our analysis.

4 Experimental Setup

Unless otherwise noted, all of our experimental setups, including model configuration, dataset, and hyperparameters follow the previous work [9] [2]. The sanity check that our trained models achieve comparable performance with the previous work is shown in Table 1. Details are in Appendix A.

We compare the model with GPT2 architecture [3] that is randomly initialized (*randomly initialized model*), pre-trained with language data (*GPT2*), and with image data (*iGPT*) to study the effect of pre-training. Pre-trained models are from the `Transformers` library from HuggingFace [36]. Like the previous study, the model code for GPT2 and iGPT are `gpt2` and `openai/imagegpt-small`.

For offline RL tasks, we use some of Mujoco tasks [14] in OpenAI Gym [37] provided by D4RL [38] [3], an offline RL dataset library. Specifically, we employ *medium* datasets of *HalfCheetah*, *Walker2d*, and *Hopper* environments. Unless otherwise specified, we show the result for *Hopper-medium*. We put the results for other environments in Appendix J.

Following the previous study [9], we train the models for 40 epochs, each of which consists of 2500 steps. In the following sections, we refer to the 40th epoch checkpoint as the *post-fine-tuning model* and the 0th epoch model (the initial state) as the *pre- fine-tuning model*. The training details, including hyperparameters and the optimizer, are detailed in Appendix B.

Table 1: Normalized mean return.

Dataset	Environment	GPT2 ([9])	iGPT ([9])	DT ([9])	GPT2 (ours)	iGPT (ours)	Random Init (ours)
Medium	Hopper	79.1 ± 1.1	5.7 ± 1.5	67.6	79.5 ± 1.3	2.6 ± 0.3	67.7 ± 3.0
	HalfCheetah	42.8 ± 0.1	1.5 ± 0.1	42.6	48.4 ± 0.0	1.3 ± 0.1	48.6 ± 0.2
	Walker 2D	78.3 ± 1.5	0.4 ± 0.4	74.0	71.2 ± 1.3	4.3 ± 1.9	71.0 ± 0.5

5 Results and Analysis

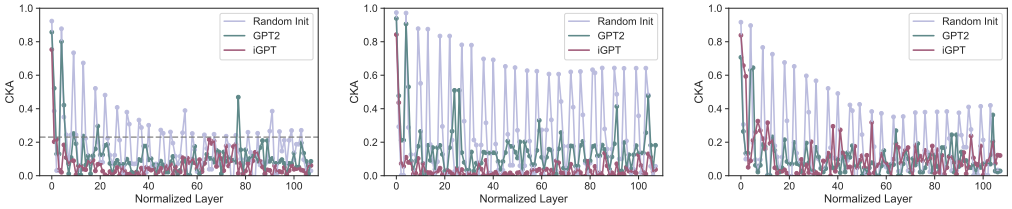
5.1 Activation Similarity

We first study how the pre-trained and randomly initialized models shape the representation in fine-tuning. To that end, we compare the centered kernel alignment (CKA) [39] of each layer's activation between pre and post-fine-tuning, investigating how each layer's representation changes by learning

[2] We use the following code for experiments: https://github.com/machelreid/can-wikipedia-help-offline-rl

[3] https://github.com/rail-berkeley/d4rl

offline RL data [4]. The CKA has been used in various studies and has provided numerous insights into the understanding of neural networks [40, 41, 42, 43, 44]. In particular, we compute the linear CKA, following previous studies [39, 45], for a subset of the dataset. We use the activation of each layer from the last time step of the context for return-to-go, state, and action, respectively. We show the result of state, putting the results for the others in Appendix J.1. Following the previous study [43], we compute CKA of activation not only for Transformer block output but for all internal activation of the Transformer blocks. The definition of the CKA and the details are in Appendices D.2 and D.1.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 1: CKA similarity of each layer between pre and post-fine-tuning.

Fig. 1 shows the result. The y-axis is each layer's CKA between representation at pre and post-fine-tuning. The x-axis is the normalized layer index: because the number of layers of iGPT (216) is twice the others (108), we average CKA over two adjacent elements starting from the 0th layer for iGPT and align the x-axis for different models for comparison. Higher CKA means higher similarity.

We observe that the CKAs of pre-trained models (GPT2 and iGPT) are uniformly low across layers except for a few layers, indicating that pre-trained models largely change representation. On the other hand, for the randomly initialized model (Random Init), the representations have a higher similarity to the initial state from the middle to the shallow layers. Also, the CKAs of the randomly initialized model seem to be higher than pre-trained models across layers. These observations indicate that the randomly initialized and pre-trained models structure their representations in very different ways. Further support for this conclusion is shown in Appendix D.4.

To identify which layers have relatively higher CKA for the randomly initialized model, we obtain layer names with CKA values above a threshold (dashed line at 0.23 in Fig. 1 (a)). Then, we find that most of them are layer normalization layers (the list of the layer names are put in Appendix D.3). Therefore, we can speculate that the difference in CKA similarity of layer normalization can be related to the effect of pre-training. This possibility could be supported by previous research [5] that reports that layer norm parameters are the most important to fine-tune.

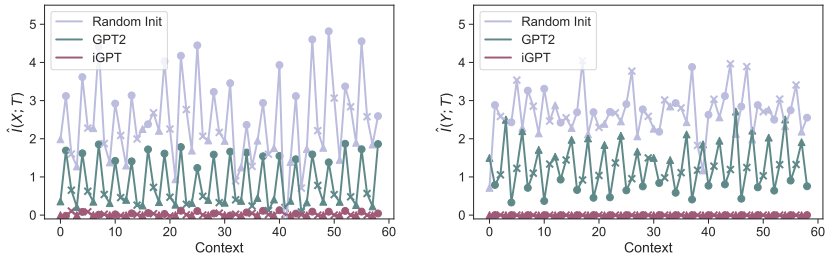
In uni-modal cases, where the statistical nature of the input data does not change significantly, previous studies have a consensus that the shallow layer acquires a more universal representation and the deep layer acquires a task-specific representation for a wide range of architectures and tasks [46, 47, 48, 49, 50, 39, 51, 42, 44]. The finding that the language-pre-trained Transformer performs as well as or better than the baseline but largely changes the representation suggests that in multi-modal cases, something different may be happening in forming the representation than in prior studies.

5.2 Mutual Information Between Hidden Representation and Data

In the previous section, we observe that pre-trained models drastically change their internal representation during fine-tuning, attaining largely different representations. One possible consequence of this observation is that the pre-trained model may have adapted better to the data in the downstream tasks. To explore this possibility, we calculate estimated values of mutual information between hidden representation and data and compare them among different models. Mutual information is a well-known measure of the mutual dependence between the two random variables. Many previous studies have used this metric to investigate the extent to which neural representation encodes input and label information and provided profound insight into how neural networks work [52, 53, 54, 55, 56].

[4] We use the following code to compute CKA: https://github.com/google-research/google-research/tree/master/representation_similarity [39]

In particular, we calculate the estimated mutual information $\hat{I}(X;T)$ between some layer representation T and input data X, and that $\hat{I}(Y;T)$ between T and label Y. We employ Mutual Information Neural Estimation (MINE) [57] to estimate mutual information since this method can estimate mutual information even when random vectors are continuous and have different dimensions [5]. Since Decision Transformers are trained to minimize the gap between predicted action $\hat{\boldsymbol{a}}_t$ and actual action $\boldsymbol{a}_t$, we define the label as $Y = \boldsymbol{a}_t$. Unlike non-sequential models, the causal Transformer exploits past context information. Thus, denoting the input token except for $\boldsymbol{a}_K$ by $\boldsymbol{x}_t$ and the internal activation of Transformer block l of the input by $T_l(\boldsymbol{x}_t)$, we compute $\hat{I}(\boldsymbol{x}_t; T_l(\boldsymbol{s}_K))$ as $\hat{I}(X;T)$ and $\hat{I}(\boldsymbol{a}_K; T_l(\boldsymbol{x}_t))$ as $\hat{I}(Y;T)$ for all token positions, $t = 0, ..., K$. We calculate these values for the shallow, middle, and deep Transformer blocks, respectively. We show the result of the middle block here and leave the remaining in Appendix J.2. The other details are described in Appendix E.1.



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 2: Estimated mutual information between data and hidden representation.

Fig. 2 is the result, where Fig. 2 (a) and (b) are results of $\hat{I}(X;T)$ and $\hat{I}(Y;T)$, respectively. The markers *triangle*, *circle*, and *cross* represent return-to-go, state, and action, respectively. The x-axis is the position of tokens in the context and the y-axis is the estimated value of mutual information.

We observe that the randomly initialized model consistently has higher mutual information both for input and label than pre-trained models. Given the catastrophic performance of iGPT, it is understandable that iGPT representations have almost no information. The result comparing GPT2 to random initialization indicates that the changed representation observed in Section 5.1 does not encode more information about the input or the label than the randomly initialized model. That is, the language-pre-trained model does well probably not because it acquires as much or more information about the data as the random initialization. This result suggests that although the representation has changed, there remains something unchanged, and the language-pre-trained model may leverage it to solve the offline RL task. Note that this might be affected by limitations of mutual information.

We also notice that for label the language-pre-trained model has consistently higher mutual information for a specific token type (return-to-go), while that is not always the case for the randomly initialized model. This result implies that the language-pre-trained model may process the information of a specific token type at each layer for prediction.

5.3 Parameter Similarity

In the previous section, we discuss the possibility that the language-pre-trained Transformer exploits some pre-acquired information. To investigate this issue on a finer scale, we turn our eye from representation to parameter. We conduct parameter-level similarity analysis and investigate to what extent parameter change between pre and post-fine-tuning. In concrete, we compute $l2$ distance and cosine similarity of parameters between pre and post-fine-tuning. The details are in Appendix F.1.

The result for $l2$ distance is shown in Fig. 3 and for cosine similarity in Fig. 4, where the top is the result for GPT2, the middle is for iGPT, and the bottom is for random initialization. The x-axis is the parameter set index, where, from left to right, the ticks correspond to the shallowest to deepest layer parameters. For visibility, we do not show the labels for ticks and put them in Appendix F.2.

[5] We use the following code for mutual information estimation: https://github.com/gtegner/mine-pytorch

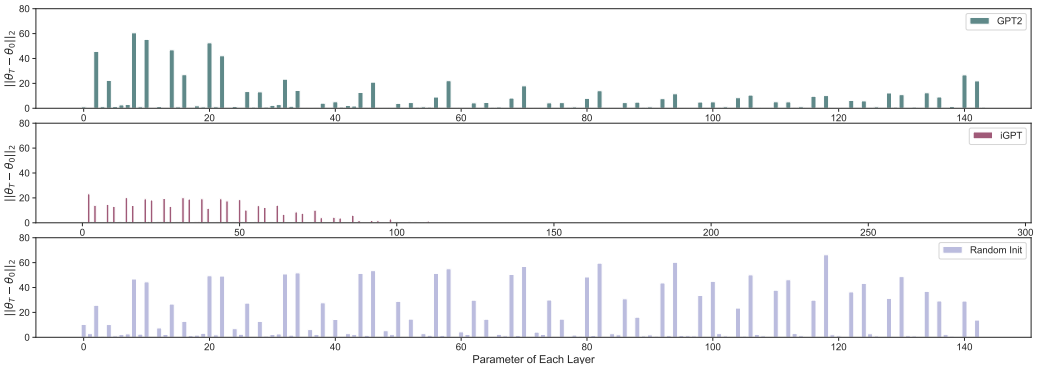


Figure 3: $l2$ distance of each parameter between pre and post-fine-tuning.

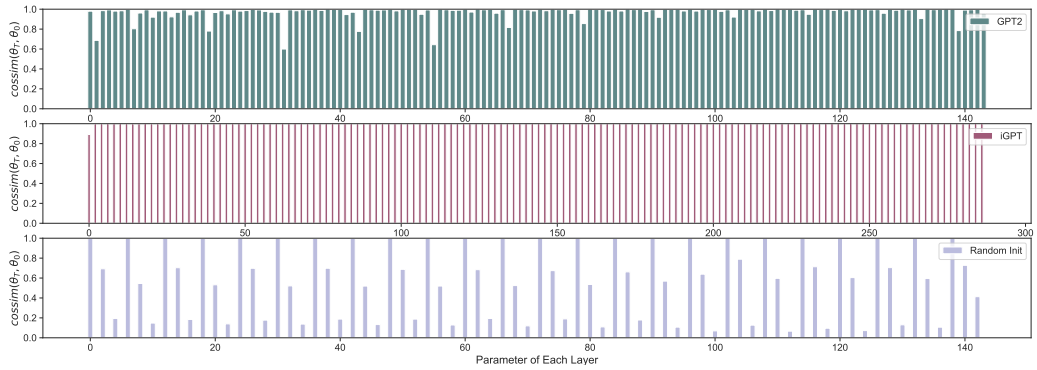


Figure 4: Cosine similarity of each parameter between pre and post-fine-tuning.

Overall, We observe that $l2$ distance is small and cosine similarity is large for pre-trained models, meaning that the parameters of pre-trained models do not change that much. At first glance, this appears to contradict the results of Section 5.1. However, prior research has reported that most of the information in Transformers propagates through skip connections [43]. This means that, as long as the parameters of the shallow layers change, the representation of each layer may change significantly without changing the parameters of each layer. We indeed observe that the shallow layer $l2$ distance of the pre-trained model shows relatively large changes. The existence of unchanged parameters means that knowledge acquired through pre-training could be stored in those parameters.

5.4 Gradient Analysis

In Section 5.3, we observe that pre-trained models' parameters don't change that much. We dig into this issue, investigating why GPT2 and iGPT do not change parameters. To this end, we compare the gradient norm and *gradient confusion* [58] at the early phase of the training (1st epoch), discussing the training difficulty. When parameter gradients of loss for two different samples are negatively correlated, we say that there is gradient confusion. Empirically, we can measure this by the minimum gradient cosine similarity among all pairs of gradients: when the value is close to 0, gradient confusion is low. The previous study showed that training is easier when gradient confusion is low [58]. Because gradient clipping is used in the previous work [9] and this study (the maximum norm is 0.25), we compute the values for clipped gradients. The details are described in Appendix G.1.

Fig. 5 is the result for the gradient confusion and Fig. 6 is that for gradient norm. Fig. 5 shows that the minimum cosine similarity of iGPT and GPT2 is similarly smaller than the randomly initialized model. This high gradient confusion probably makes pre-trained models harder to train, which explains why they change parameters less than the randomly initialized model. Fig. 6, on the other hand, shows that the gradient norm of iGPT is concentrated around the clipping threshold (0.25), while that of the others is below the bound. That is, the differences in the magnitude of the gradient are collapsed for iGPT, reducing the informational value of the gradients.

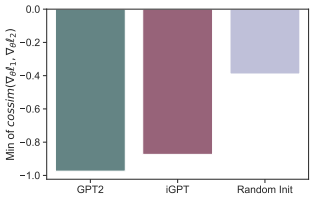


Figure 5: Grad. confusion.

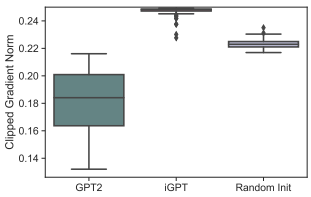


Figure 6: Grad. norm.

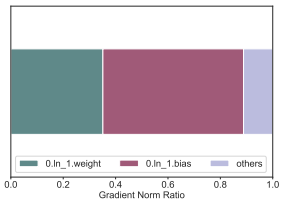
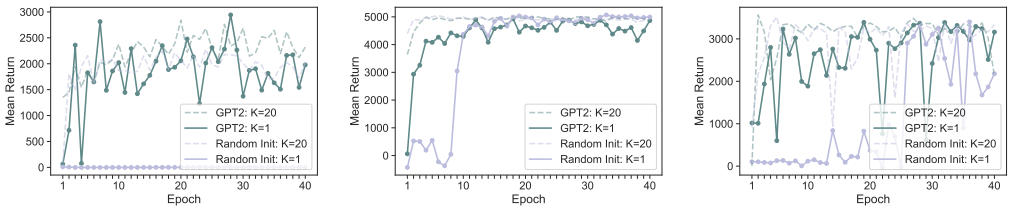


Figure 7: iGPT's grad. norm ratio.

To identify which parameter of iGPT dominates the gradient norm, we compare the gradient norm of each parameter set. We calculate the gradient norm ratio of weights and biases of all parameters in all Transformer blocks. The result is shown in Fig. 7: for visibility, we show the parameters of the first layer norm module (weights `0.ln_1.weight` and biases `0.ln_1.bias`) and lump the remaining results together as `others`. The full result is shown in Appendix G.2. We observe that weights and biases of the first layer norm module dominate the norm. This result implies that the difficulty of training iGPT might partially come from the use of gradient clipping on large gradients dominated by few parameters. A complementary analysis is described in Appendix G.3.

5.5 Fine-Tuning with No Context Information

In the sections up to this point, we find the possibility that the language-pre-trained model may exploit some pre-acquired information. Then what information does the language-pre-trained model exploit to solve the tasks? One possibility is the ability to handle context because Transformer is skilled at handling the relations of tokens, which enables it to perform well in a variety of NLP tasks [22]. Therefore, we train the language-pre-trained and the randomly initialized model with no access to the context ($K = 1$) to dig into this possibility. If a model learns efficiently without context, the model is already likely to have context-equivalent information. Details of experiments are in Appendix H.1.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 8: Mean return throughout fine-tuning when access to the context information is prohibited.

The mean return throughout training time is shown in Fig. 8 and the normalized scores of return [6] are shown in Table 2. The solid lines in Fig. 8 are results without context and the dashed lines for reference are those with context. From Fig. 8, we observe that when the context is not provided ($K = 1$), the GPT2 reaches the high mean return much faster than the randomly initialized model. Furthermore, Table 2 reveals that the randomly initialized model fails to learn the trajectory for a dataset (*Hopper*), while the language-pre-trained model consistently achieves a comparable performance with that of the baseline result ($K = 20$). That is, we observe that the language-pre-trained Transformer consistently performs more efficiently when no context is provided. This is somewhat surprising, given that previous studies have pointed out that Decision Transformer performs significantly worse with no context [35]. The result that the language-pre-trained Transformer learns efficiently as well without

Table 2: Normalized return of $K = 1$.

Dataset	Environment	GPT2	Random Init
Medium	Hopper	83.73	−0.24
	HalfCheetah	47.7	49.0
	Walker 2D	69.0	69.1

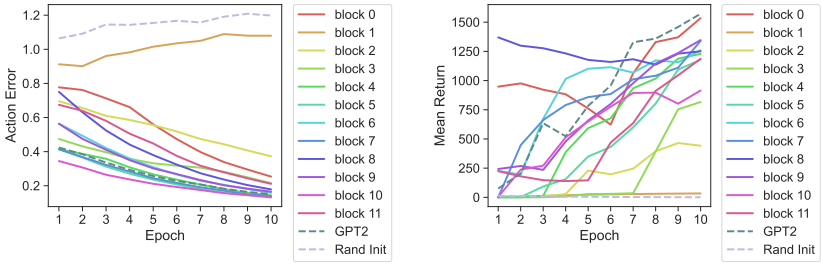
[6] Following the previous studies [38, 35, 9], we report normalized scores: $100 \times \frac{\text{score} - \text{random score}}{\text{expert score} - \text{random score}}$.

context as it does with context suggests that the model may have already acquired some kind of context-like information through pre-training and is utilizing it to solve the task.

5.6 More In-Depth Analysis of Context Dependence

5.6.1 Replacement by the Pre-Trained Block

In the previous sections, we find that the language-pre-trained Transformer exploits context-related information to solve the task. To identify which block contains the useful information, we conduct a block replacement experiment, where we replace a Transformer block of the randomly initialized model with that of the language-pre-trained model and then fine-tune the model without context. If replacement of a block does not help training, the block probably did not learn effective information in pre-training. Because we observe in Section 5.5 that the early phase characterizes the difference of the models, we focus on the epochs up to 10th epoch. We explain the other details in Appendix I.1.1.



(a) Action error **(b)** Mean return

Figure 9: Learning curve when a single Transformer block is replaced with a pre-trained one.

The result is shown in Fig. 9, where (a) is the prediction error of action and (b) is the mean return [7]. To get a rough trend, we take the exponential moving average. We observe that only when block 1 is replaced with the pre-trained block, does the action error (Fig. 9 (a)) and the mean return (Fig. 9 (b)) behave similarly to the randomly initialized model. Thus, we conclude that all blocks other than block 1 contain context-equivalent useful information to solve tasks. The findings that replacing a single block with the pre-trained one improves learning efficiency and that effective information is distributed across layers alone are surprising enough. This could be related to the dominant role of skip connections for information propagation [43] or the nature of non-monotonic change in the content each layer processes [20] in Transformers. We show the result for Hopper since the trend explained above is most pronounced. For other environments, although the similarity between the random model and block1 is much weaker, the observation that block 1 is not good is consistent. Results for other environments are in Appendix J.6.1.

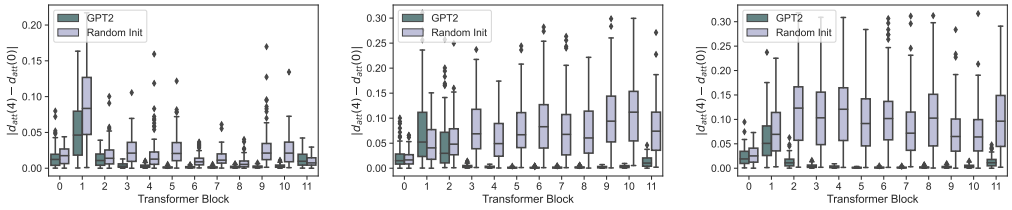
5.6.2 Attention Distance Analysis

In previous sections, we find that the information improving learning efficiency is preserved in all blocks only in the varying amount and that is probably context-related. To further confirm that the context-like information is indeed preserved, we investigate how the model processes context by calculating the attention distance [59] of each Transformer block for the result of Section 5.5. Attention distance is the average distance between key and query weighted by attention weight. Previous studies successfully used this measure to reveal how locally Transformers process context [43, 59]. Attention distance will elucidate how far away each model processes the tokens of input.

In particular, we compute the gap ($|d_{att}(epoch) - d_{att}(0)|$) between attention distance at the initial state ($d_{att}(0)$) and that at an epoch ($d_{att}(epoch)$) to study to what extent the model preserves the way to utilize context. If the gap is smaller, it means that the attention distance is more preserved. To clarify the difference between the language-pre-trained model and the randomly initialized model, we compare the gap between epoch 0 and 4 since at epoch 4 for all environments the randomly initialized

[7] Note that since the medium dataset we use is corrected by a sub-optimal policy, a better action error does not always result in a better return. We explain this more in detail in Appendix O.

model still produces a low return, while the GPT2 achieves a high return (Fig. 8). The detail of the setup and results for other epochs are shown in Appendices I.2.1 and J.6.2.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 10: Attention distance gap between epoch 0 and epoch 4.

Results are shown in Fig. 10, where the x-axis is the Transformer block index and the y-axis is the gap between attention distances. Fig. 10 (a), shows that the attention gaps of GPT2 except for block 1 are small, indicating that attention distances are kept from the pre-training in these blocks. Taken together with the finding of Section 5.6.1 that blocks other than block 1 improve learning efficiency, these preserved attention distances are likely to positively affect fine-tuning. This result further supports the hypothesis that language-pre-training brings the model context-like information and the use of this information allows efficient learning.

We also notice the outstanding contrast that gaps of language-pre-trained models are almost zero except for block 1 (and block 2 for HalfCheetah), while those for the randomly initialized model are large. This trend is clearer from middle to deep blocks (3 to 10) for all environments. Turning an eye to Fig. 9 (a), we notice that the action error of these blocks decreases faster than the other block (0 to 2 and 11). Thus, we conclude that the pre-trained context-like information could help the training by enabling the model to predict action more accurately.

We summarize findings of Section 5 with particular attention to the implication for performance:

- 5.1: Re-using representation is not the cause of the good performance of GPT2.
- 5.2: Fitting to data better is not the cause of the good performance of GPT2.
- 5.3: The good performance of GPT2 might come from some unchanged parameters.
- 5.4: Gradient clipping and large gradient might be a cause of the bad performance of iGPT.
- 5.5: GPT2 can learn efficiently even without the context provided.
- 5.6.1: Even a single pre-trained Transformer block makes training more efficient.
- 5.6.2: A cause of the good performance of GPT2 is a pre-acquired way to use context.

6 Discussion

Conclusion We have examined how pre-training on data of different modalities influences fine-tuning to offline RL. Internal representation analysis shows that pre-trained Transformer largely changes its representation while attaining less knowledge of the downstream task. By the analysis of change in parameters, we find that pre-trained Transformers do not change parameters that much and that the bad performance of the image-pre-trained model might partially come from gradient clipping on the large gradient dominated by a few parameters. Fine-tuning models with no context, we find that the language-pre-trained model can efficiently solve offline RL tasks even without context. Follow-up analysis supports the hypothesis that language pre-training probably gives the Transformer context-like information and the model exploits it to tackle the offline RL tasks.

Discussion The finding in Section 5.6.2 of the usefulness of middle blocks is interesting, given that the contextual model's middle layers contain syntactic information [60, 61] and are most transferable [20]. This implies that the syntactic capability of Transformers could be related to the performance on offline RL. The role of layer norm is also noteworthy as we observe that it may characterize the behavior in fine-tuning (Sections 5.1 and 5.4). Studying the pre-training on other RL tasks is also critical to understanding the benefit of pre-training in RL. Many studies of human evolution and

development point to a close relationship between language and behavior [62, 63]. Hence, studying the relationship between these modalities matters to create human-like intelligence that handles language. A more detailed analysis of this relationship is a promising research direction.

Limitation and Impact We consider only a random seed and a few datasets from OpenAI Gym for analysis. Thus, using diverse data is important and checking if the claim holds with the average results with many more seeds is crucial. Also, the validity of the metrics we used should be critically examined since different results could be obtained by using more carefully designed ones. Because this work is fundamental research, no immediate serious negative societal impact is expected.

Acknowledgments and Disclosure of Funding

We thank DEEPCORE Inc. for providing the computational resource.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[5] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

[6] Kimia Noorbakhsh, Modar Sulaiman, Mahdi Sharifi, Kallol Roy, and Pooyan Jamshidi. Pretrained language models are symbolic mathematics solvers too! *arXiv preprint arXiv:2110.03501*, 2021.

[7] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

[8] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[9] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[10] Allison C Tam, Neil C Rabinowitz, Andrew K Lampinen, Nicholas A Roy, Stephanie CY Chan, DJ Strouse, Jane X Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *arXiv preprint arXiv:2204.05080*, 2022.

[11] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2021.

[12] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[13] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[15] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, 2014.

[16] Josip Djolonga, Jessica Yung, Michael Tschannen, Rob Romijnders, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Matthias Minderer, Alexander D'Amour, Dan Moldovan, et al. On robustness and transferability of convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[17] Yifan Ding, Liqiang Wang, and Boqing Gong. Analyzing deep neural network's transferability via frechet distance. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021.

[18] Romain Orhand, Hiba Khodji, Amarin Hutt, and Anne Jeannin-Girardon. Quantification of the transferability of features between deep neural networks. *Procedia Computer Science*, 192:138–147, 2021.

[19] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in NLP applications? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[20] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[21] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[22] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*, 2019.

[23] Allyson Ettinger. What bert is not: Lessons from a new suite of psycholinguistic diagnostics for language models. *Transactions of the Association for Computational Linguistics*, 8:34–48, 2020.

[24] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, July 2020.

[25] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[26] Ryokan Ri and Yoshimasa Tsuruoka. Pretraining with artificial language: Studying transferable knowledge in language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022.

[27] Cheng-Han Chiang and Hung-yi Lee. On the transferability of pre-trained language models: A study from artificial datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[28] Ethan A. Chi, John Hewitt, and Christopher D. Manning. Finding universal grammatical relations in multilingual BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[29] Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep CALM and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[31] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[32] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[33] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[34] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[35] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

[36] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020.

[37] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[38] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[39] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[40] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.

[41] John Wu, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Similarity analysis of contextual word representation models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[42] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems*, 2020.

[43] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In *Advances in Neural Information Processing Systems*, 2021.

[44] Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *Proceedings of the 38th International Conference on Learning Representations*, 2021.

[45] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2021.

[46] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, 2017.

[47] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, 2018.

[48] Ari S. Morcos, David G.T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *International Conference on Learning Representation*, 2018.

[49] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *Advances in Neural Information Processing Systems*, 2019.

[50] Elena Voita, Rico Sennrich, and Ivan Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[51] Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. What happens to BERT embeddings during fine-tuning? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, 2020.

[52] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, 2015.

[53] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[54] Hassan Hafez-Kolahi and Shohreh Kasaei. Information bottleneck and its applications in deep learning. *arXiv preprint arXiv:1904.03743*, 2019.

[55] Ziv Goldfeld and Yury Polyanskiy. The information bottleneck problem and its applications in machine learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):19–38, 2020.

[56] Bernhard C Geiger. On information plane analyses of neural network classifiers–a review. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[57] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[58] Karthik Abinav Sankararaman, Soham De, Zheng Xu, W. Ronny Huang, and Tom Goldstein. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[60] John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

[61] Yoav Goldberg. Assessing bert's syntactic abilities. *arXiv preprint arXiv:1901.05287*, 2019.

[62] K. S. Lashley. The problem of serial order in behavior. *Cerebral Mechanisms in Behavior; The Hixon Symposium*, pages 112—-146, 1951.

[63] Patricia M Greenfield. Language, tools and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behavioral and Brain Sciences*, 14(4):531–551, 1991.

[64] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[65] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[66] Arthur Gretton, Kenji Fukumizu, Choon Teo, Le Song, Bernhard Schölkopf, and Alex Smola. A kernel statistical test of independence. In *Advances in Neural Information Processing Systems*, 2007.

[67] 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. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Section 6
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] https://github.com/t46/pre-training-different-modality-offline-rl
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Appendix B
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix K
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All existing assets we used are mentioned in footnote or References.
   (b) Did you mention the license of the assets? [Yes] Appendix L
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] Appendix L
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Table of Contents

A Fine-Tuning Result for Sanity Check **17**

B Training Details **17**

C Model Architecture and Module Names **18**

D Activation Similarity **20**
D.1 Details of Experiments 20
D.2 Centered Kernel Alignment (CKA) 20
D.3 Layer Names Whose CKA is Above a Threshold 20
D.4 CKA Between Different Models 21
D.5 CKA Between Different Layers in a Model 21

E Mutual Information Between Hidden Representation and Data **22**
E.1 Details of Experiments 22
E.2 Mutual Information Neural Estimation (MINE) 23
E.3 Mutual Information Without Considering Context 23

F Parameter Similarity **24**
F.1 Details of Experiments 24
F.2 Ticks' Labels of Figures 3 and 4 24

G Gradient Analysis **27**
G.1 Details of Experiments 27
G.2 Gradient Norm for Each Parameter 27
G.3 Analysis of the Effect of Gradient Clipping 27

H Fine-Tuning with No Context Information **28**
H.1 Details of Experiments 28
H.2 Analysis of Why Randomly Initialized Model Fails for Hopper with No Context . 29

I More In-Depth Analysis of Context Dependence **29**
I.1 Replacement by the Pre-Trained Block 29
I.2 Attention Distance Analysis 30

J Results for Other Conditions **31**
J.1 Activation Similarity 31
J.2 Mutual Information Between Hidden Representation and Data 36
J.3 Parameter Similarity 37
J.4 Gradient Analysis 40
J.5 Fine-Tuning with No Context Information 43
J.6 More In-Depth Analysis of Context Dependence 44

K Details for Computation **46**

L Licences of Assets Used for Our Experiments **46**
L.1 Dataset 46

L.2 Code 46

M Note on the Role of Each Sub-Sections in Section 5 **47**

N Note on Relationship Between Parameter Change and Representation Change **47**

O Note on Why Better Action Prediction Does Not Always Result in Better Return **47**

A Fine-Tuning Result for Sanity Check

We compare our performance results to those of previous studies to ensure that they are not far off from the results of the prior study [9], the result of which is shown in Table 1. Following the previous studies [38, 35, 9], we report the normalized score of mean return: $100 \times \frac{\text{score} - \text{random score}}{\text{expert score} - \text{random score}}$, where *random score* is mean return generated by a random policy and *expert score* is generated by a policy trained by Soft Actor-Critic [64]. Mean return is the sum of rewards averaged over trajectory. For further details of the datasets and metrics, please refer to the paper that proposes D4RL [38]. Although the previous work [9] used several techniques to improve the performance, e.g. language model co-training, the extension of positional embedding, and similarity encouragement, we do not employ any of these techniques so that we study the pure effect of pre-training. The training details are described in Appendix B

The result for the previous work is the average and standard deviation of three random seeds, while our result is those of two random seeds. The aim of this comparison is just to confirm that our result is not too pathological, checking soundness with two seeds would be valid enough. For reference, we also include the results of the Decision Transformer (DT) since the randomly initialized model (Random Init) is a large Decision Transformer. Following the previous study [9], which says that it conducted early stopping, we show the result of the best checkpoint in 40 epoch checkpoints. The result shows that our result is not too far away from the previous work, ensuring that the models of the subject of our analysis are valid enough.

B Training Details

Hyperparameters are determined following previous study [9]. The hyperparameters and other setups for fine-tuning the model are summarized in Table 3. The number of layers is 12 for GPT2 and randomly initialized model and 24 for iGPT. The other setup not specified, including β of Adam optimizer [65] and model initialization scheme follows Pytorch default ones or the default values of optional arguments in the scripts that our experiment is based on [8]. We do not do any pre-training ourselves, but use publicly available pre-trained models explained in Section 3.

Each model is trained to minimize the mean squared loss between its predicted actions $\hat{\boldsymbol{a}}_t$ and the true actions $\boldsymbol{a}_t$. To get the big picture of how fine-tuning of GPT-based models to offline RL data is conducted, please refer to the pseudo-code of the previous work [35].

The D4RL mujoco task used as offline RL data for our analysis has *expert*, *medium*, *medium-replay*, *medium-expert*, and *random* datasets. The *expert* dataset is the one trained by Soft Actor-Critic [64], *medium* is the one partially trained by Soft Actor-Critic and was stopped early, *medium-replay* is the one accumulated in the replay buffer before the model reached *medium*'s level, and *medium-expert* is a mixture of *medium* and *expert* results. The *random* dataset is the trajectory collected by the random policy. We conducted our experiments on the *medium* data set since it was used by the previous study [9] and our analysis is based on the observations of this previous study. The *medium* dataset we used is 1 million time steps collected by the policy explained above. For the detail of the dataset, please refer to the original paper [38].

[8] `experiment.py` in the following repository:
https://github.com/machelreid/can-wikipedia-help-offline-rl/blob/main/code/experiment.py

Table 3: Training configuration.

# Layers	12
Emb. Dim.	768
# Attention Heads	1
Batch size	64
Context	20
Return-to-go conditioning	6000 HalfCheetah 3600 Hopper 5000 Walker
Dropout	0.2
Learning rate	$1e-4$
LR Warmup	5000 steps
Epoch	40
# Steps per Epoch	2500
Optimizer	Adam
Weight Decay	$1e-4$

C Model Architecture and Module Names

In the following sections, we use the module names of our models (`gpt2` and `openai/imagegpt-small`) to describe the experimental details. Thus, for the reference, we show the model architecture we use and the names of the modules of the model. We show that of `gpt2` but the configuration is the same for `openai/imagegpt-small` except for the number of Transformer blocks. For the detailed configurations of pre-trained models, please refer to the following links:

- `gpt2`: https://huggingface.co/gpt2
- `openai/imagegpt-small`: https://huggingface.co/openai/imagegpt-small

We show the model architecture below, following the Pytorch format:

```
DecisionTransformer(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0): GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.2, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (1): GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
```

```
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.2, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
      (2): GPT2Block(
      .
      .
      .
      )
      (11): GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.2, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.2, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (embed_timestep): Embedding(1000, 768)
  (embed_return): Linear(in_features=1, out_features=768, bias=True)
  (embed_state): Linear(in_features=11, out_features=768, bias=True)
  (embed_action): Linear(in_features=3, out_features=768, bias=True)
  (embed_ln): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

Listing 1: Model architecture.

Among the above modules, for our analysis, we focus on the modules in `DecisionTransformer.transformer.h`. We summarize the module-related notations used in this paper below for reference:

- *Transformer blocks*: modules directly under `DecisionTransformer.transformer.h`
  - e.g. `DecisionTransformer.transformer.h[0]`.
- *Outputs of blocks*: outputs of `mlp.dropout` of blocks.
  - e.g. `DecisionTransformer.transformer.h[0].mlp.dropout`.
- *Layers*: modules under Transformer blocks.
  - e.g. `DecisionTransformer.transformer.h[0].attn.c_attn`.
- *Parameter set*: parameters of each layer.
  - e.g. `DecisionTransformer.transformer.h[0].ln_1.weight`.

Note that these notations are sometimes used interchangeably as long as it doesn't significantly deteriorate the readability.

D Activation Similarity

D.1 Details of Experiments

We randomly sample 100 samples and compute unbiased estimators [39] of linear CKA for these 100 activation vectors. The activation to be analyzed are outputs from all *layers*. In the Decision Transformer, the effective total input length is context length K times the number of token types $(\hat{R}, \boldsymbol{s}, \boldsymbol{a})$. As a result, the shape of the activation obtained at each layer is `(batch size, 3x context length, embedding dimension)`. In this study, we compute the CKA for the activity of shape `(batch size, embedding dimension)` that corresponds to the last time step in the context of return-to-go, state, and action, respectively. In other words, noting activation as in the form of python NumPy array `activation`, we obtain `activation[:, -1, :]`, `activation[:, -2, :]`, and `activation[:, -3, :]`. In the previous work analyzing the internal representation of Transformer [41], the dimension of context and representation seem to be concatenated: `activation.contiguous().view(activation.shape[0] -1)`. Since the representation obtained in this way is hard to interpret, we decide to use the way we described above to obtain activation. A descriptive diagram for activation that we compute CKA about is shown in Fig. 11. The design of the diagram is based on a previous study [35].

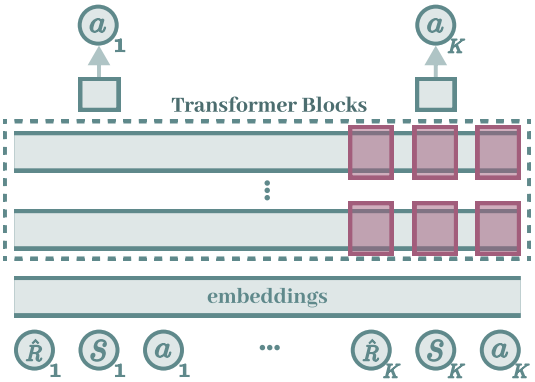


Figure 11: Activation we consider to compute CKA.

Each rectangle colored in red is an activation we consider. The left-most rectangles are the activation with return-to-go as input, the middle rectangles are the that with status as input, and the right-most rectangles are that with action as input. Since we consider only the last in the context, the rectangles are drawn only where the tokens of the K-th step are received as input. In Section 5.1, we show the results of activation that $\boldsymbol{s}_K$ is fed in the diagram.

D.2 Centered Kernel Alignment (CKA)

We show the definition of Centered Kernel Alignment (CKA). Given two activation vectors $\boldsymbol{X} \in \mathbb{R}^{m \times p_1}$ and $\boldsymbol{Y} \in \mathbb{R}^{m \times p_1}$, where m is data size and p_1 and p_2 are dimensions of hidden layers, the CKA of the two representations is as follows:

$$\text{CKA} = \frac{\text{HSCI}(\boldsymbol{K}, \boldsymbol{L})}{\sqrt{\text{HSCI}(\boldsymbol{K}, \boldsymbol{K})\text{HSCI}(\boldsymbol{L}, \boldsymbol{L})}}, \tag{1}$$

where HSCI is the Hilbert-Schmidt independence criterion [66] and $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{L}_{ij} = l(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are kernels. Particularly, linear CKA is defined as follows:

$$\text{linear CKA} = \frac{||\boldsymbol{Y}^\top \boldsymbol{X}||_F^2}{||\boldsymbol{X}^\top \boldsymbol{X}||_F ||\boldsymbol{Y}^\top \boldsymbol{Y}||_F}, \tag{2}$$

where $|| \cdot ||_F$ is the Frobenius norm. For further details, please refer to the previous work [39].

D.3 Layer Names Whose CKA is Above a Threshold

In Section 5.1, we say we observe that most of the high CKAs of Fig. 1 (b), which are greater than 0.38, are those of layer normalization. The reason we set this seemingly arbitrary threshold is that we

observe higher layers in a block than other layers in Fig. 1 and want to identify these layers. By eye inspection, we find the threshold over which all these points are included in Fig. 1 is around 0.38 and so we select this value as a threshold. Here, we show the raw result of the observation. The layer name list is shown below:

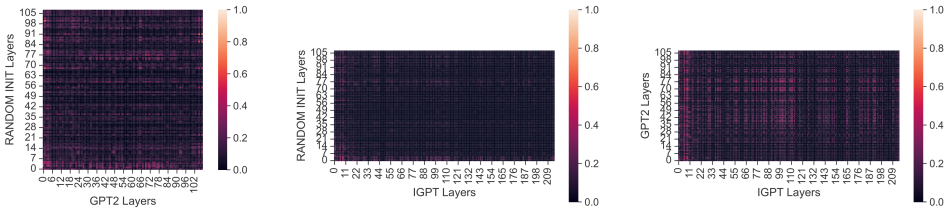
```
0.ln_1, 0.attn.c_attn, 0.ln_2, 0.mlp.c_fc, 0.mlp.c_proj, 0.mlp.dropout
1.ln_1, 1.attn.c_attn, 1.ln_2
2.ln_1,                2.ln_2
3.ln_1,                3.ln_2, 3.mlp.c_fc
4.ln_1, 4.attn.c_attn, 4.ln_2
5.ln_1, 5.attn.c_attn, 5.ln_2
6.ln_1, 6.attn.c_attn, 6.ln_2
7.ln_1, 7.ln_2
8.ln_1,                8.ln_2
9.ln_1, 9.ln_2
10.ln_1, 10.attn.c_attn, 10.ln_2
11.ln_1.                 11.ln_2
```

Listing 2: Layer names whose CKA is above a threshold

Note that, for example, `0.ln_1` is the first layer normalization layer (`ln_1`) of the first transformer block (`0`). We observe that 24 of 34 layers are layer norm modules (`ln_1` and `ln_2`).

D.4 CKA Between Different Models

To check if pre-trained models and the randomly initialized model converge to a similar representation or not, we compute the CKA between representations of fine-tuned these models. The result is shown in Fig. 12, where (a) is the CKA between GPT2 and iGPT, (b) is that between GPT2 and random initialization, and (c) is that between iGPT and random initialization. We observe that CKA values in these heat maps are small, confirming that different models learn different representations.



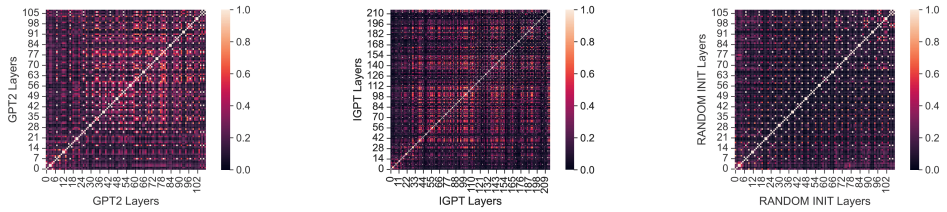
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 12: CKA similarity across layers between different fine-tuned models.

D.5 CKA Between Different Layers in a Model

D.5.1 Post-Fine-Tuning

In Appendix D.4, we find that each layer of the pre-trained model and randomly initialized model learn representation differently. In this section, we delve into what representation these models acquire. For that purpose, we compute CKA values of the different layers in the same model after fine-tuning. The result for the state input token with the *Hopper-medium* dataset is shown in Fig. 13. The results for other environments and input tokens are in Appendix J.1.



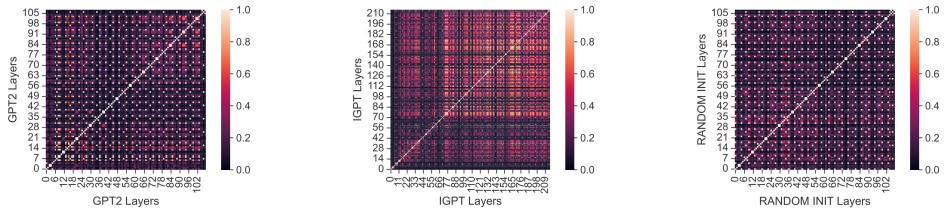
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 13: CKA of different layers in the same models (epoch 40).

Comparing the CKA matrix of pre-trained models (Fig. 13 (a) and Fig. 13 (b)) with that of the randomly initialized model (Fig. 13 (c)), we observe that the randomly initialized model has an almost equally divided lattice structure, while pre-trained models have some block-like structure. In particular, the shallow blocks and the middle to top blocks of language-pre-trained model have a bit similar representation, respectively. This result indicates that probably layers of random initialization process information separately, while some layers of language-pre-trained models coordinate to represent information.

D.5.2 Pre-Fine-Tuning

For just reference, we put the CKA matrix of pre-trained models before fine-tuning as well. The result is shown in Fig. 14. We observe the grid structure at the initial state as well.



(a) GPT2 **(b)** iGPT **(c)** Random Initialization

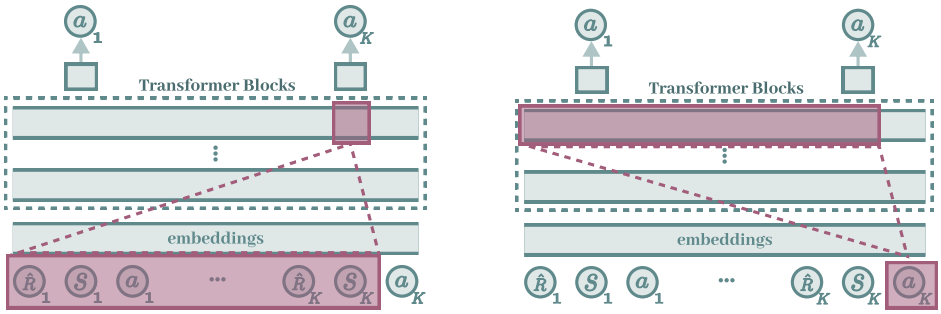
Figure 14: CKA of different layers in the same models (epoch 0).

E Mutual Information Between Hidden Representation and Data

E.1 Details of Experiments

MINE, the definition of which is explained in Appendix E.2, estimates mutual information by training a neural network. The neural network we use is a feed-forward ReLU network with two hidden layers of width 400. We train the neural network for 1000 iterations by Adam optimizer with a learning rate of $1e-4$. Same as in Appendix D.1, we randomly sample 100 trajectories and use them to obtain the activation. Thus, the dataset for mutual information calculation is the pair between these 100 activation vectors and 100 trajectories for $\hat{I}(X;T)$ and 100 last action vectors for $\hat{I}(Y;T)$. If the estimated value is `NaN`, we exclude the point from the figure.

The descriptive diagram that shows hidden activation and data we consider to compute estimated mutual information is Fig. 15, where (a) is for $\hat{I}(X;T)$ and (b) is for $\hat{I}(Y;T)$. This is an example of mutual information for the deep Transformer block. For the middle block, we use block 6 for GPT2 and random initialization and block 12 for iGPT. Specifically, the outputs of the shallow, middle, and deep Transformer blocks are the outputs of `0.mlp.dropout`, `6.mlp.dropout`, and `11.mlp.dropout` for GPT2 and randomly initialized models and `0.mlp.dropout`, `12.mlp.dropout`, and `23.mlp.dropout` for iGPT.



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 15: Data and activation we consider to compute estimated mutual information.

E.2 Mutual Information Neural Estimation (MINE)

Given random variables X and T, whose joint probability is $\mathbb{P}_{XT}$ and marginal distributions are $\mathbb{P}_X = \int_{\mathcal{T}} \mathrm{d}\mathbb{P}_{XT}$ and $\mathbb{P}_T = \int_{\mathcal{X}} \mathrm{d}\mathbb{P}_{XT}$, the mutual information of X and T are as follows:

$$I(X;T) = \int_{\mathcal{X}\times\mathcal{T}} \log \frac{\mathrm{d}\mathbb{P}_{XT}}{\mathrm{d}\mathbb{P}_X \otimes \mathbb{P}_T} \mathrm{d}\mathbb{P}_{XT}. \tag{3}$$

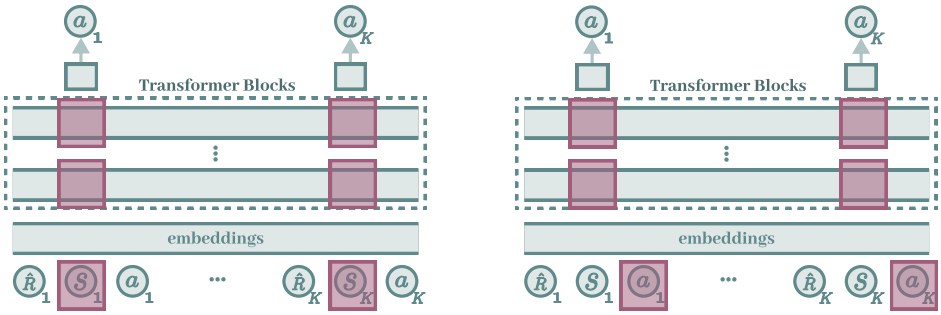
Denoting neural network with parameters $\boldsymbol{\theta} \in \Theta$ by $\mathrm{NN}_{\boldsymbol{\theta}}$ and the empirical distribution of $\mathbb{P}$ characterized by n i.i.d. samples by $\hat{\mathbb{P}}^{(n)}$, MINE is defined as follows [57]:

$$\hat{I}(X;T)_n = \sup_{\theta\in\Theta} \mathbb{E}_{\mathbb{P}^{(n)}_{XT}} [\mathrm{NN}_\theta] - \log \left(\mathbb{E}_{\mathbb{P}^{(n)}_X \otimes \hat{\mathbb{P}}^{(n)}_T} \left[e^{\mathrm{NN}_\theta} \right] \right). \tag{4}$$

Please refer to the previous study [57] for the details.

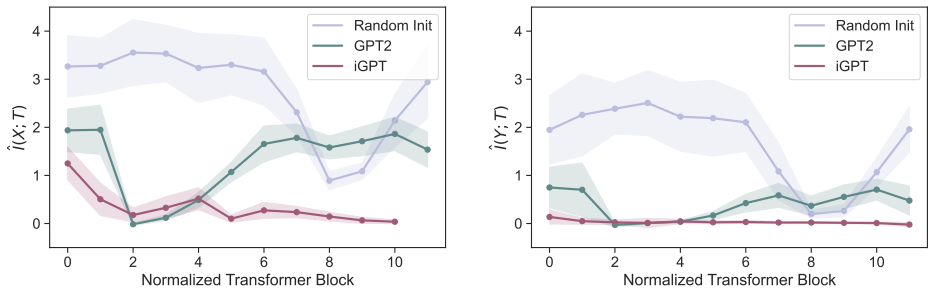
E.3 Mutual Information Without Considering Context

As a complementary analysis, we also calculate the estimated mutual information between hidden representation and $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$. In other words, we compute $\hat{I}(\boldsymbol{s}_t; T_l(\boldsymbol{s}_t))$ and $\hat{I}(\boldsymbol{a}_t; T_l(\boldsymbol{s}_t))$ as $\hat{I}(X;T)$ and $\hat{I}(Y;T)$ for all time steps in the context $t = 1, ..., K$. Then, we take an average of them over the context. The descriptive diagram is shown in Fig. 16.



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 16: Data and activation we consider to compute estimated mutual information.



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 17: Estimated mutual information between hidden representation and state and action.

The result for *Hopper-medium* is shown in Fig. 17. Each point indicates the estimated mean mutual information and the shaded area is the standard deviation. Just as we did in Section 5.1, we take an average over two adjacent elements for iGPT for comparison. We again observe that the randomly initialized model generally has more information on input and label, and iGPT has almost nothing about label-related information.

F Parameter Similarity

F.1 Details of Experiments

The parameters considered in the analysis are those in Transformer blocks (`DecisionTransformer.transformer.h`). We concatenate all these parameters into a single vector and compute $l2$ distance and cosine similarity for this vector. For the post-fine-tuning model in Section 5.3, we use the model trained with the *Hopper-medium* dataset. The results for other environments are in Appendix J.3.

F.2 Ticks' Labels of Figures 3 and 4

The labels of ticks of Figs. 3 and 4 in Section 5.3 correspond to all *parameter sets*. For example, one of the parameter sets is `0.ln_1.weight`: this is the weight vector of the layer normalization in the first (`0`) transformer block (`h`). The labels for ticks of GPT2 and the randomly initialized model are summarized below:

```
'0.ln_1.weight',
'0.ln_1.bias',
'0.attn.c_attn.weight',
'0.attn.c_attn.bias',
'0.attn.c_proj.weight',
'0.attn.c_proj.bias',
'0.ln_2.weight',
'0.ln_2.bias',
'0.mlp.c_fc.weight',
'0.mlp.c_fc.bias',
'0.mlp.c_proj.weight',
'0.mlp.c_proj.bias',
'1.ln_1.weight',
'1.ln_1.bias',
'1.attn.c_attn.weight',
'1.attn.c_attn.bias',
'1.attn.c_proj.weight',
'1.attn.c_proj.bias',
'1.ln_2.weight',
'1.ln_2.bias',
'1.mlp.c_fc.weight',
'1.mlp.c_fc.bias',
'1.mlp.c_proj.weight',
'1.mlp.c_proj.bias',
```

```
'2.ln_1.weight',
'2.ln_1.bias',
'2.attn.c_attn.weight',
'2.attn.c_attn.bias',
'2.attn.c_proj.weight',
'2.attn.c_proj.bias',
'2.ln_2.weight',
'2.ln_2.bias',
'2.mlp.c_fc.weight',
'2.mlp.c_fc.bias',
'2.mlp.c_proj.weight',
'2.mlp.c_proj.bias',
'3.ln_1.weight',
'3.ln_1.bias',
'3.attn.c_attn.weight',
'3.attn.c_attn.bias',
'3.attn.c_proj.weight',
'3.attn.c_proj.bias',
'3.ln_2.weight',
'3.ln_2.bias',
'3.mlp.c_fc.weight',
'3.mlp.c_fc.bias',
'3.mlp.c_proj.weight',
'3.mlp.c_proj.bias',
'4.ln_1.weight',
'4.ln_1.bias',
'4.attn.c_attn.weight',
'4.attn.c_attn.bias',
'4.attn.c_proj.weight',
'4.attn.c_proj.bias',
'4.ln_2.weight',
'4.ln_2.bias',
'4.mlp.c_fc.weight',
'4.mlp.c_fc.bias',
'4.mlp.c_proj.weight',
'4.mlp.c_proj.bias',
'5.ln_1.weight',
'5.ln_1.bias',
'5.attn.c_attn.weight',
'5.attn.c_attn.bias',
'5.attn.c_proj.weight',
'5.attn.c_proj.bias',
'5.ln_2.weight',
'5.ln_2.bias',
'5.mlp.c_fc.weight',
'5.mlp.c_fc.bias',
'5.mlp.c_proj.weight',
'5.mlp.c_proj.bias',
'6.ln_1.weight',
'6.ln_1.bias',
'6.attn.c_attn.weight',
'6.attn.c_attn.bias',
'6.attn.c_proj.weight',
'6.attn.c_proj.bias',
'6.ln_2.weight',
'6.ln_2.bias',
'6.mlp.c_fc.weight',
'6.mlp.c_fc.bias',
'6.mlp.c_proj.weight',
'6.mlp.c_proj.bias',
'7.ln_1.weight',
'7.ln_1.bias',
'7.attn.c_attn.weight',
'7.attn.c_attn.bias',
'7.attn.c_proj.weight',
```

```
'7.attn.c_proj.bias',
'7.ln_2.weight',
'7.ln_2.bias',
'7.mlp.c_fc.weight',
'7.mlp.c_fc.bias',
'7.mlp.c_proj.weight',
'7.mlp.c_proj.bias',
'8.ln_1.weight',
'8.ln_1.bias',
'8.attn.c_attn.weight',
'8.attn.c_attn.bias',
'8.attn.c_proj.weight',
'8.attn.c_proj.bias',
'8.ln_2.weight',
'8.ln_2.bias',
'8.mlp.c_fc.weight',
'8.mlp.c_fc.bias',
'8.mlp.c_proj.weight',
'8.mlp.c_proj.bias',
'9.ln_1.weight',
'9.ln_1.bias',
'9.attn.c_attn.weight',
'9.attn.c_attn.bias',
'9.attn.c_proj.weight',
'9.attn.c_proj.bias',
'9.ln_2.weight',
'9.ln_2.bias',
'9.mlp.c_fc.weight',
'9.mlp.c_fc.bias',
'9.mlp.c_proj.weight',
'9.mlp.c_proj.bias',
'10.ln_1.weight',
'10.ln_1.bias',
'10.attn.c_attn.weight',
'10.attn.c_attn.bias',
'10.attn.c_proj.weight',
'10.attn.c_proj.bias',
'10.ln_2.weight',
'10.ln_2.bias',
'10.mlp.c_fc.weight',
'10.mlp.c_fc.bias',
'10.mlp.c_proj.weight',
'10.mlp.c_proj.bias',
'11.ln_1.weight',
'11.ln_1.bias',
'11.attn.c_attn.weight',
'11.attn.c_attn.bias',
'11.attn.c_proj.weight',
'11.attn.c_proj.bias',
'11.ln_2.weight',
'11.ln_2.bias',
'11.mlp.c_fc.weight',
'11.mlp.c_fc.bias',
'11.mlp.c_proj.weight',
'11.mlp.c_proj.bias'
```

Listing 3: Label list of ticks.

For iGPT, the number of transformer block is twice (from 0-11 to 0-23) but the configuration is the same.

G Gradient Analysis

G.1 Details of Experiments

We randomly sample 100 samples for gradient norm in Fig. 6 and 50 samples for gradient confusion in Fig. 5 and compute the gradient of the loss on each of these samples for all parameters of the models fine-tuned after 1 epoch. The bar plot of Fig. 5 is the minimum of the $50 \times 50 = 2500$ cosine similarities. Each point of Fig. 6 corresponds to the gradient norm of each data sample in 100 gradient norms. The top of the box is the 1st quartile and the bottom of the box is the 3rd quartile of the points. The whiskers extend from the box by 1.5 times the inter-quartile range. For gradient clipping, we use a method in Pytorch [67] and set the maximum norm to be 0.25, following the previous work [9].

The process of creating Fig. 7 is as follows. We first compute gradient norms per parameter set and examine the bar plot that we will explain in Appendix G.2. Noticing that two peaks exist in the figure, we check the label of the parameter sets and find that they are `0.ln_1.weight` and `0.ln_1.bias`. Thus, we lump the remaining as `others` and only show the ratio in the main body of the paper to tell readers the main finding from the observation in an easy-to-understand manner. The parameter sets considered to compute the parameter set-wise gradient norms are the same as those in Appendix F.2.

G.2 Gradient Norm for Each Parameter

For visibility, we show only the ratio in Section 5.4 as Fig. 7. The full results for gradient norms of each parameter are shown in Fig. 18. The labels of the ticks on the x-axis are the same as that in Appendix F.2. We immediately notice that the first two parameters (`0.ln_1.weight` and `0.ln_1.bias`) are much larger than the remains.

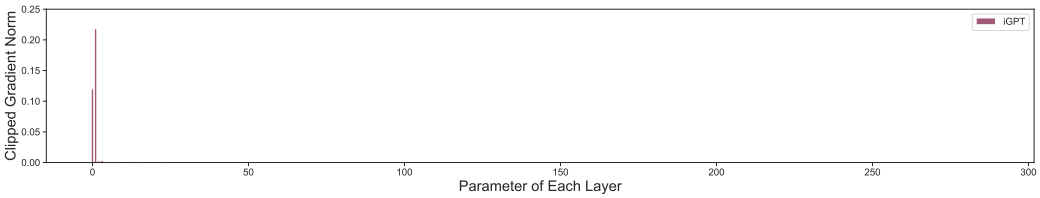


Figure 18: Gradient norm of iGPT's each parameter at epoch 1.

G.3 Analysis of the Effect of Gradient Clipping

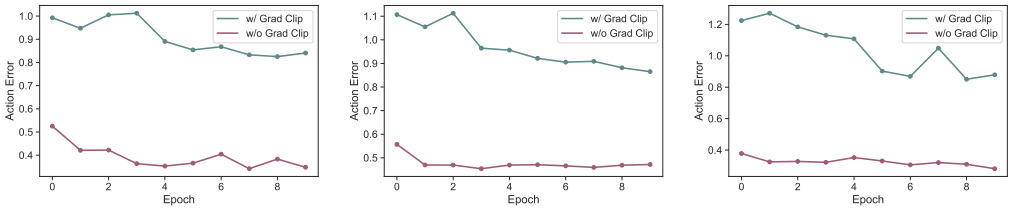
In Section 5.4, we noted that large gradient and gradient clipping could be a cause of the bad performance of iGPT. To further explore this point, we conducted a simple experiment.

The experiments of the previous study [9] and our experiments both used a Pytorch function called `torch.nn.utils.clip_grad_norm_` [9] for gradient clipping. This function divides each gradient by the total norm of all gradients and multiplies the clipping constant (gradient norm clipping). However, as we pointed out in Section 5.4, the gradient norm values are dominated by only a few parameters (the layer normalization layer in the first block). This large norm affects the normalization of all gradients, decreasing the informational value of most gradients. Therefore, we hypothesized that one of the possible reasons why iGPT is difficult to learn can be the use of `torch.nn.utils.clip_grad_norm_function`.

Thus we trained the image-pre-trained model without using gradient clipping. The basic experimental setting is the same as that of Section 5.4 except that we trained the model for 10 epochs, instead of 40 epochs. The results are shown in Figs 19 and 20 for action error and mean return, respectively. The x-axis is the epoch. These figures show that eliminating gradient clipping does not immediately solve the catastrophic performance at least up to 10 epochs. However, we did find that action error is much smaller for no gradient clipping condition, indicating that the learning process of it seems to be

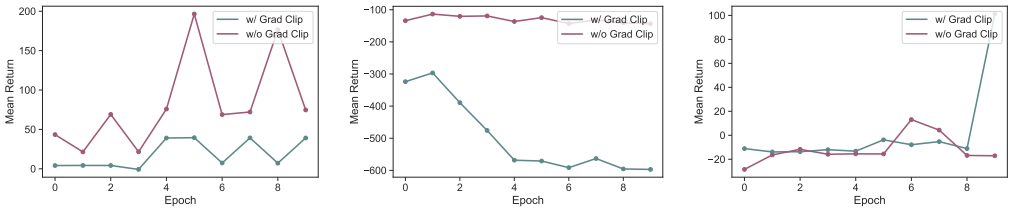
[9] Documentation of `torch.nn.utils.clip_grad_norm_` :
https://pytorch.org/docs/stable/generated/torch.nn.utils.clip_grad_norm_.html

more stable and efficient than the clipping was applied. We also found that the return means seem to improve, albeit very slightly. Thus, we can conclude that gradient clipping is a cause of the bad performance of iGPT, though it does not seem to be critical. Exploring the critical factor that makes iGPT performance catastrophic is left a future work.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 19: Action error: with gradient clipping v.s. without gradient clipping



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

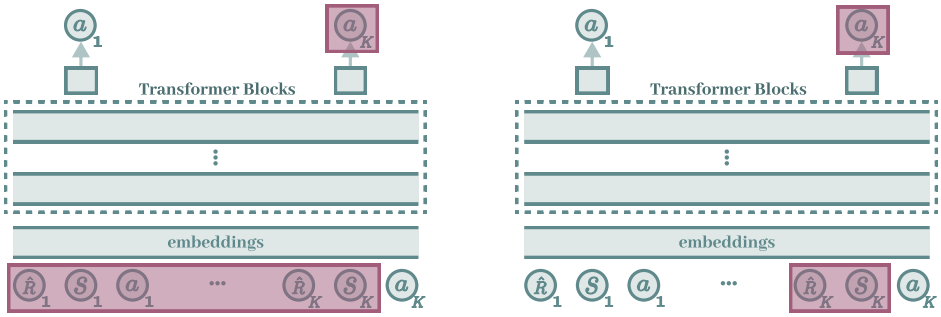
Figure 20: Mean return: with gradient clipping v.s. without gradient clipping

H Fine-Tuning with No Context Information

H.1 Details of Experiments

The configuration of the training is the same as that described in Appendix B except that the context is 1 (no context). We evaluate the model per epoch and the normalized score in Table 4 is the result of the best checkpoint in the 40 checkpoints. The mean return reported in Fig. 8 and Table 4 is the average of the returns over the trajectory. Table 4 is the result of one random seed (seed = 666). The mean and standard deviation of the normalized score for the two seeds (seed = 666 & 42) and the mean return of another seed (seed = 42) are reported in Appendix J.5.

In this paper, we mean *context* by the number of accessible time steps for a model to predict, following the previous study [35]. If the context length is K, the model can use inputs in past K steps to predict the action at the current step, while the context length is 1 (no context), the model has to predict only from the current time step.



(a) Context length is K **(b)** Context length is 1

Figure 21: Descriptive diagram to explain what we mean by *context*.

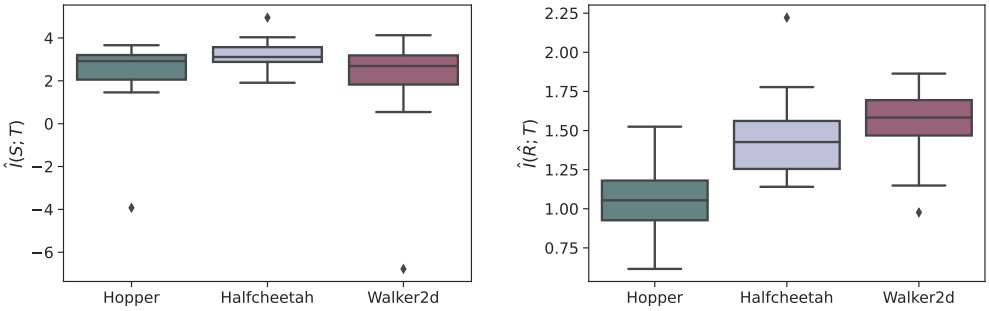
H.2 Analysis of Why Randomly Initialized Model Fails for Hopper with No Context

In Section 5.5, we observe that the performance of randomly initialized models for the Hopper environment is particularly worse than that for other environments (Table 4). In this Section, we further explore a possible cause of this observation.

We speculate that this may be because *how much of the range of context that needs to be looked at* changes depending on the data set. For example, prior studies on decision transformer reported that the effect of context varied depending on the task [35]. Improvement by context means that the context is important information for solving the task. Thus, it could be possible that Hopper is the dataset that requires more information from context than the other two data sets.

As a test of the hypothesis, we randomly sampled a batch sample and calculated the mutual information between action and state or return-to-go at the same time step and compare them between different environments; note that this is the mutual information between the data, not between the data and representation. The higher the value of the mutual information, the higher the mutual dependence between state or return-to-go and action at the same time step. Hence, higher mutual information suggests that the model could predict action better only from the information at the current time step.

In particular, we sample 100 samples of context length $K = 20$. Then, for all time step t, we compute the estimated mutual information $\hat{I}(\boldsymbol{s}_t; \boldsymbol{a}_t)$ between state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$ and that $\hat{I}(\hat{R}_t; \boldsymbol{a}_t)$ between return-to-go $\hat{R}_t$ and action. Other setup for mutual information estimation is the same as that of Section 5.2. The result is shown in Fig 22, where (a) is the box plot for the state and action and (b) is that for the return-to-go and action. The top of the box is the 1st quartile and the bottom of the box is the 3rd quartile of the points. The whiskers extend from the box by 1.5 times the inter-quartile range. Each point corresponds to different time step in the context.



(a) $\hat{I}(\boldsymbol{s}_t; \boldsymbol{a}_t)$ **(b)** $\hat{I}(\hat{R}_t; \boldsymbol{a}_t)$

Figure 22: Mutual information between data at the same time step in trajectories

We observe that the estimated mutual information between return-to-go and action is smaller on average for Hopper than for other environments (Fig. 22 (b)), though that between action and state does not differ that much (Fig. 22 (a)). Prior research has indicated that return-to-go information seems to be important for prediction [9]. Hence, we can say that models have to use more information from the other steps in the context to solve the Hopper task than models do for other environments. This result supports our hypothesis above.

I More In-Depth Analysis of Context Dependence

I.1 Replacement by the Pre-Trained Block

I.1.1 Details of Experiments

The training configuration is the same as that of Section 5.5. The smoothing factor of the exponential moving average for Fig. 9 is 0.8. The action error is the mean square loss between the true action $\boldsymbol{a}_t$ and the predicted action $\hat{\boldsymbol{a}}_t$. Considering that the goal of this experiment is to highlight the effect of pre-training, we train the model for 10 epochs for Hopper and 5 epochs for HalfCheetah and

Walker2D because Fig. 8 shows that the difference between the randomly initialized and pre-trained model is evident during these epochs.

I.2 Attention Distance Analysis

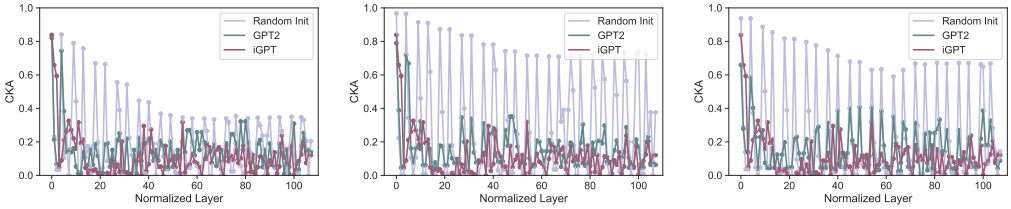
I.2.1 Details of Experiments

In the main body of this paper, we mean *attention distance* by *average/mean attention distance* in the previous work [59, 43]. Because we use only one attention head in this study, there is a single point per sample in Fig. 10. We randomly sample 100 trajectory samples and compute attention for these trajectories. When calculating attention distance, we allow the model to access the context of $K = 20$ length and compute the attention distance up to the length. Each point of the box plot (Fig. 10) corresponds to the attention distance of each sample. The configuration for the box plot is the same as that in Appendix G.

J Results for Other Conditions

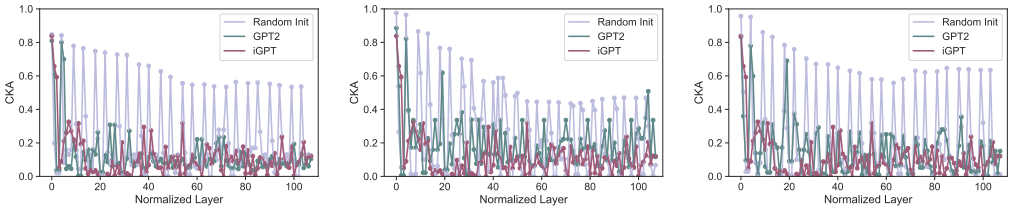
J.1 Activation Similarity

J.1.1 CKA Between Pre and Post-Fine-Tuning



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

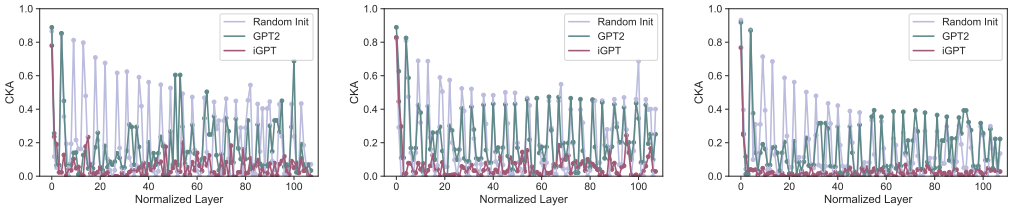
Figure 23: CKA similarity of each layer between pre and post-fine-tuning (Action).



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

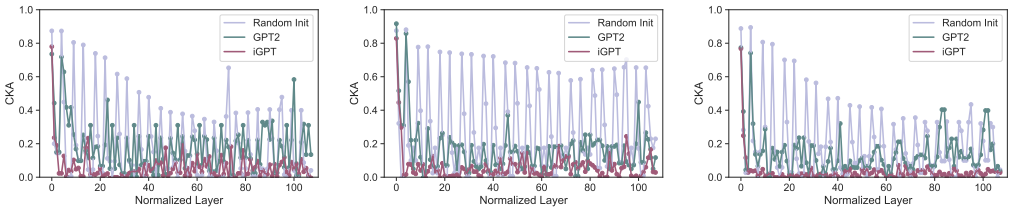
Figure 24: CKA similarity of each layer between pre and post-fine-tuning (Return-to-go).

J.1.2 CKA Between Pre and Post-Fine-Tuning (Seed = 42)



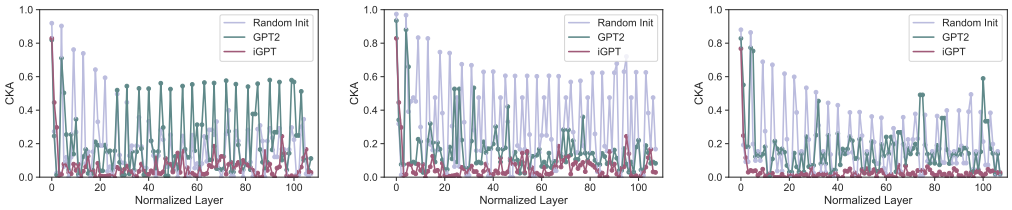
(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 25: CKA similarity of each layer between pre and post-fine-tuning (State, Seed = 42).



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

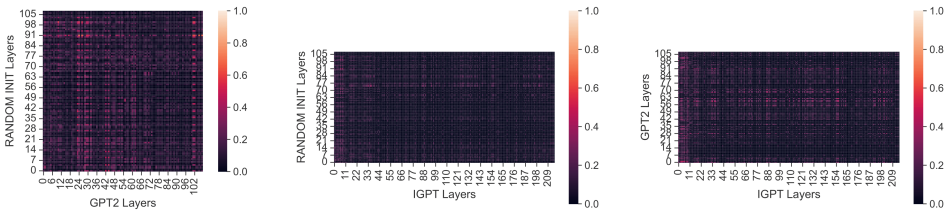
Figure 26: CKA similarity of each layer between pre and post-fine-tuning (Action, Seed = 42).



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

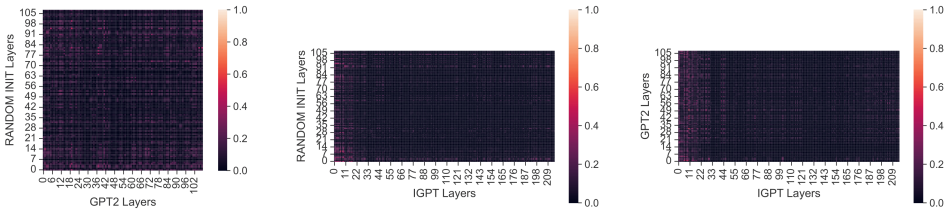
Figure 27: CKA similarity of each layer between pre and post-fine-tuning (Return-to-go, Seed = 42).

J.1.3 CKA Between Different Models



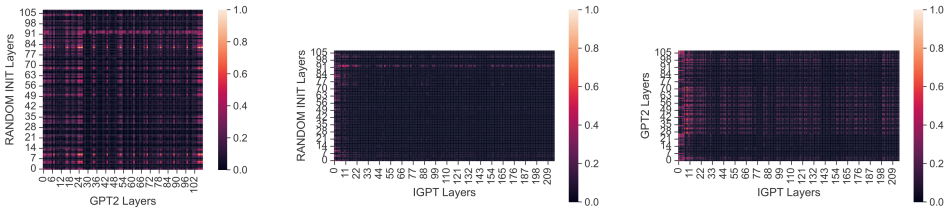
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 28: CKA between different models (Hopper & Return-to-go).



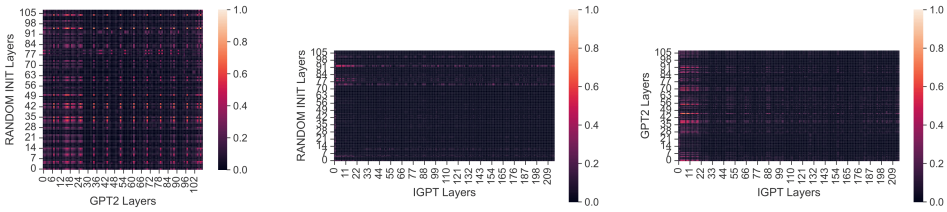
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 29: CKA between different models (Hopper & Action).



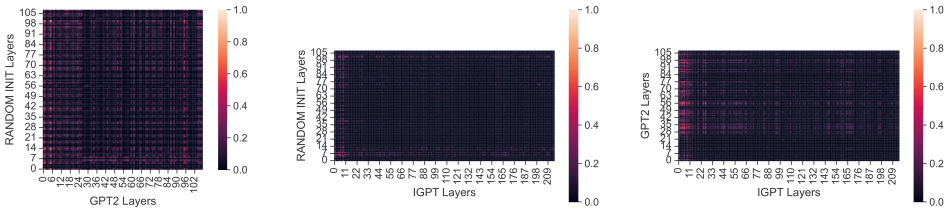
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 30: CKA between different models (HalfCheetah & State).



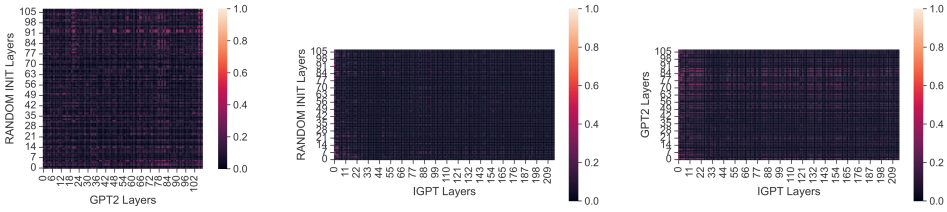
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 31: CKA between different models (HalfCheetah & Return-to-go).



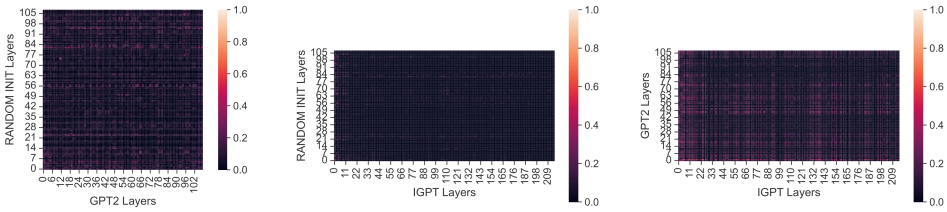
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 32: CKA between different models (HalfCheetah & Action).



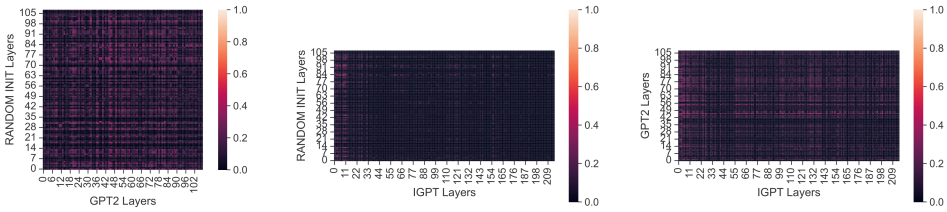
(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

Figure 33: CKA between different models (Walker2D & State).



(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

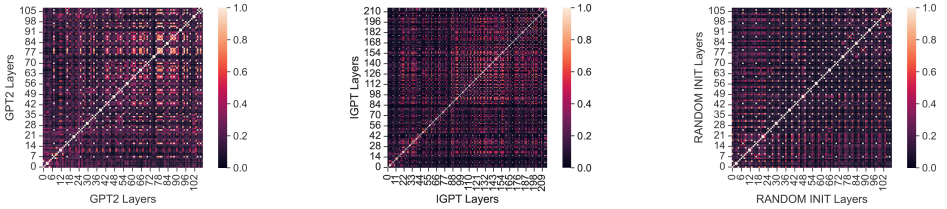
Figure 34: CKA between different models (Walker2D & Return-to-go).



(a) Random Init. vs GPT2 **(b)** Random Init. vs iGPT **(c)** GPT2 vs iGPT

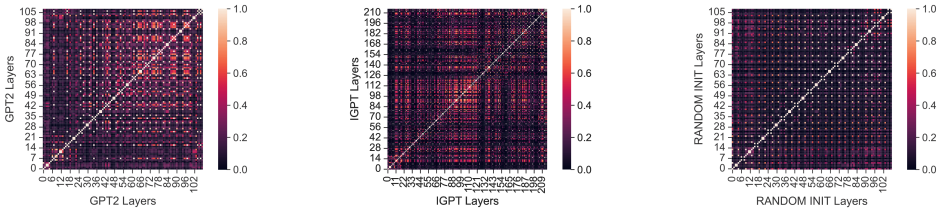
Figure 35: CKA between different models (Walker2D & Action).

J.1.4 CKA Between Different Layers in a Model



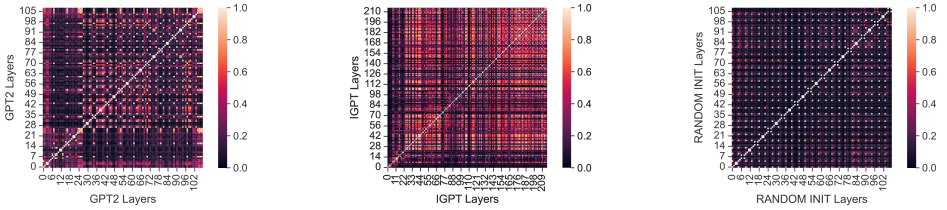
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 36: CKA of different layers in the same model (Hopper & Return-to-go).



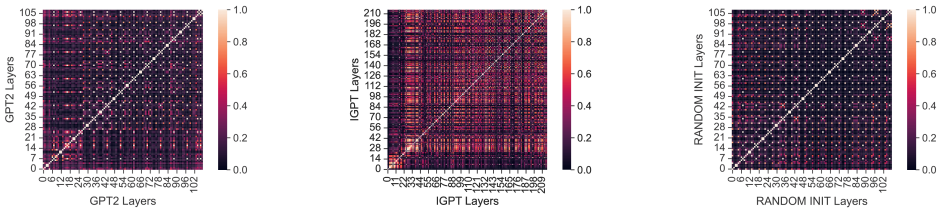
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 37: CKA of different layers in the same model (Hopper & Action).



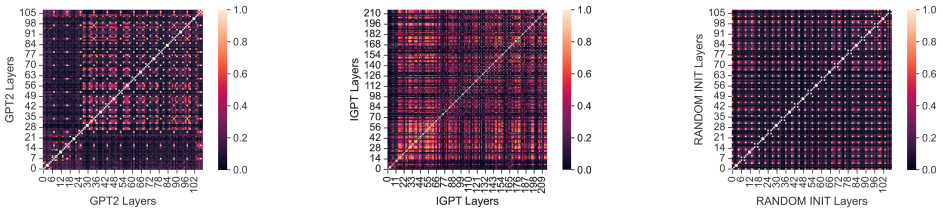
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 38: CKA of different layers in the same model (HalfCheetah & State).



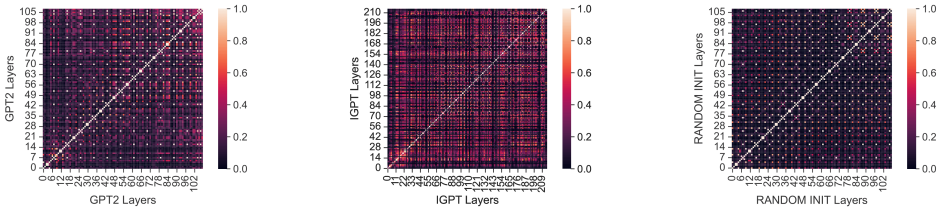
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 39: CKA of different layers in the same model (HalfCheetah & Return-to-go).



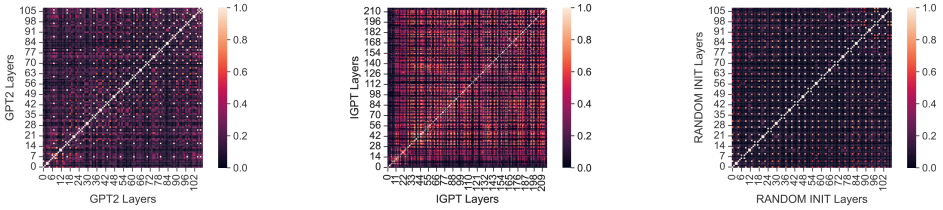
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 40: CKA of different layers in the same model (HalfCheetah & Action).



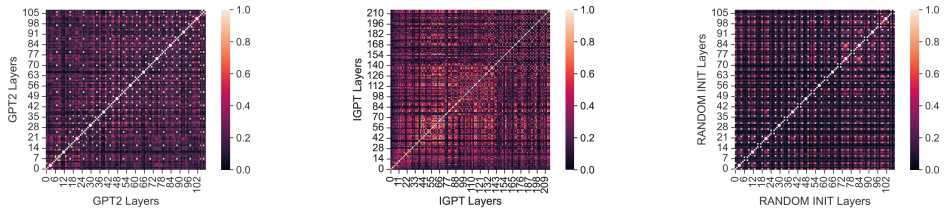
(a) GPT2 **(b)** iGPT **(c)** Random Initialization

Figure 41: CKA of different layers in the same model (Walker2D & State).



(a) GPT2 **(b)** iGPT **(c)** Random Initialization

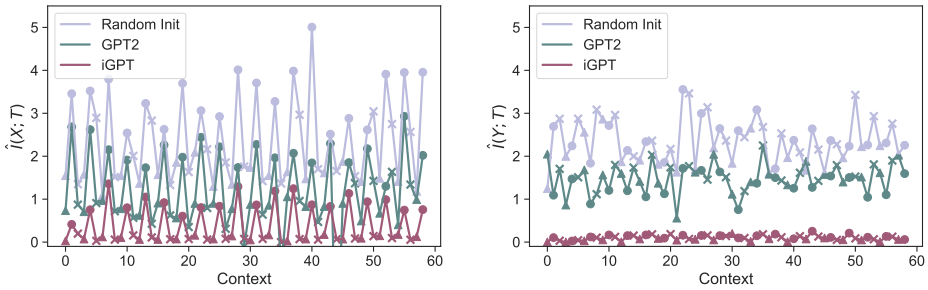
Figure 42: CKA of different layers in the same model (Walker2D & Return-to-go).



(a) GPT2 **(b)** iGPT **(c)** Random Initialization

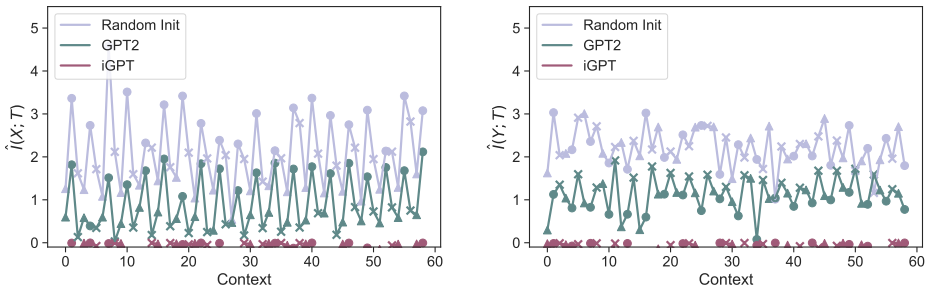
Figure 43: CKA of different layers in the same model (Walker2D & Action).

J.2 Mutual Information Between Hidden Representation and Data



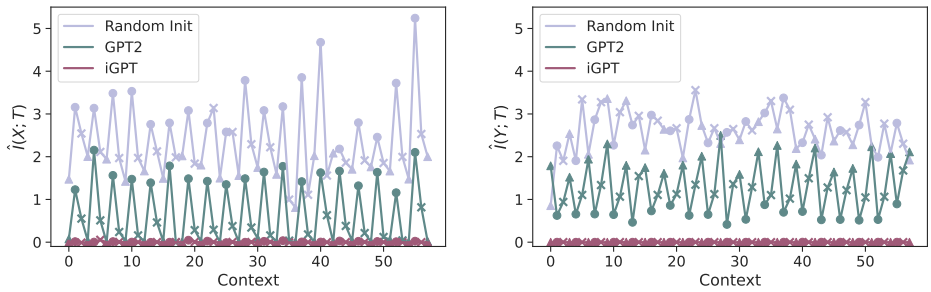
(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 44: Estimated mutual information between data and hidden representation (Shallow).



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 45: Estimated mutual information between data and hidden representation (Deep).



(a) $\hat{I}(X;T)$ **(b)** $\hat{I}(Y;T)$

Figure 46: Estimated mutual information between data and hidden representation (Middle, Seed = 42).

J.3 Parameter Similarity

J.3.1 Parameter Similarity (Other Environment)

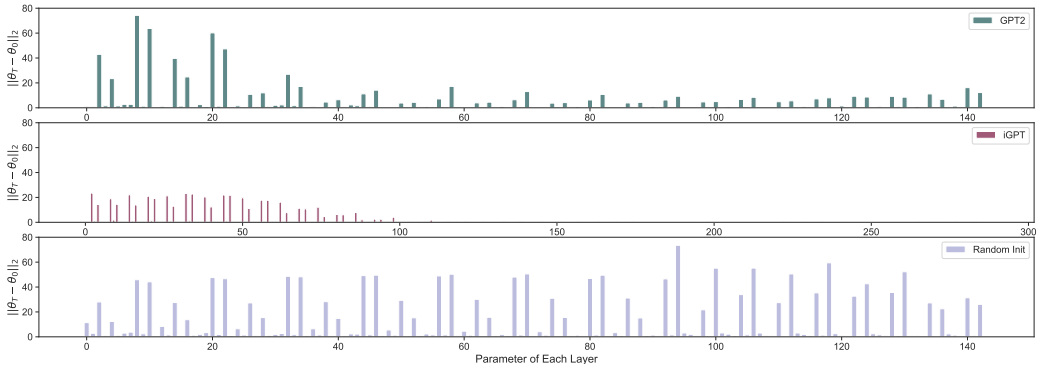


Figure 47: L2 distance of each parameter between pre post-fine-tuning (HalfCheetah).

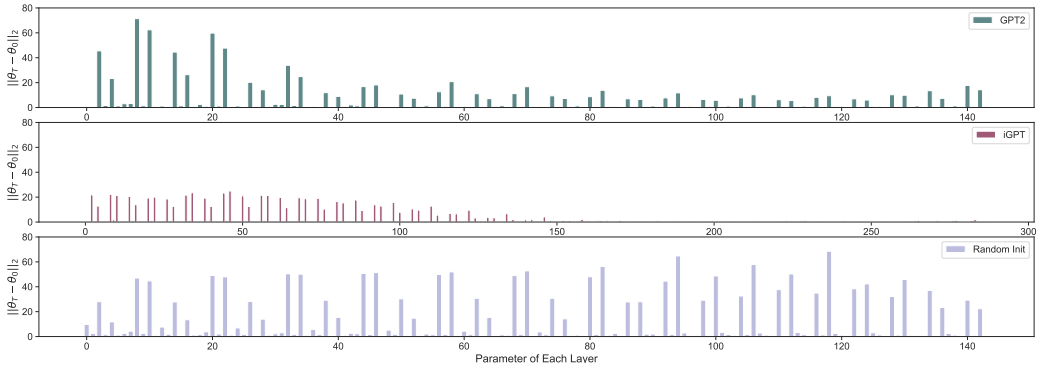


Figure 48: L2 distance of each parameter between pre post-fine-tuning (Walker2D).

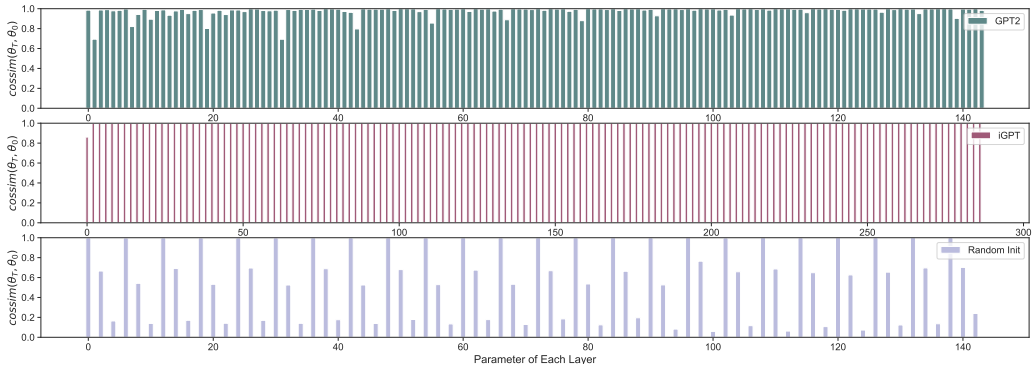


Figure 49: Cosine similarity of each parameter between pre post-fine-tuning (HalfCheetah).

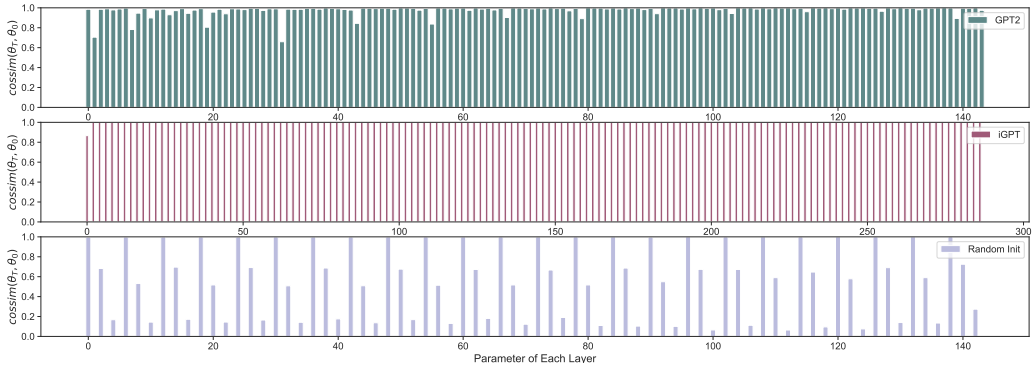


Figure 50: Cosine similarity of each parameter between pre post-fine-tuning (Walker2D).

J.3.2 Parameter Similarity (Seed = 42)

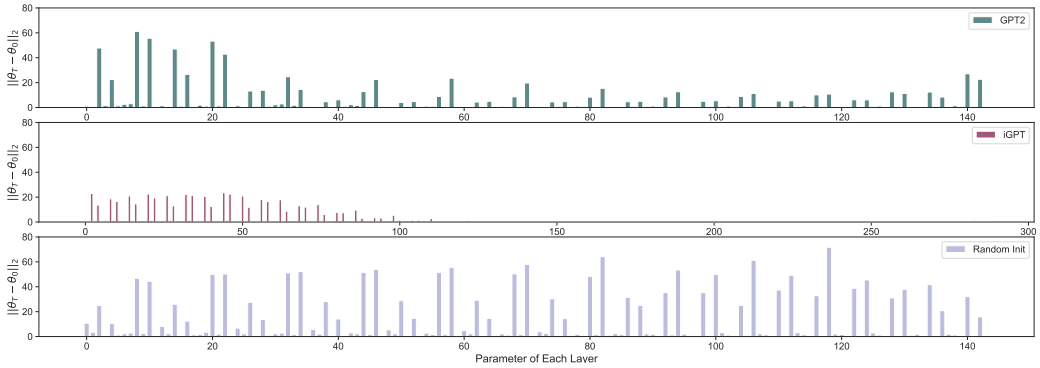


Figure 51: L2 distance of each parameter between pre post-fine-tuning (Hopper, Seed = 42).

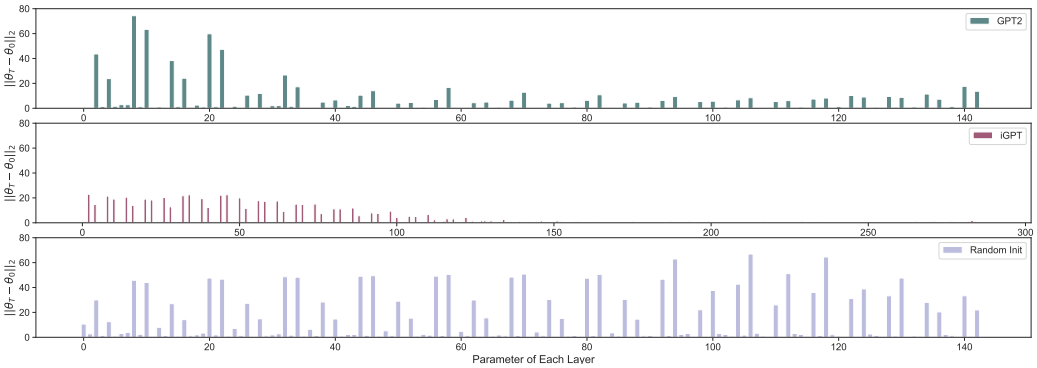


Figure 52: L2 distance of each parameter between pre post-fine-tuning (HalfCheetah, Seed = 42).

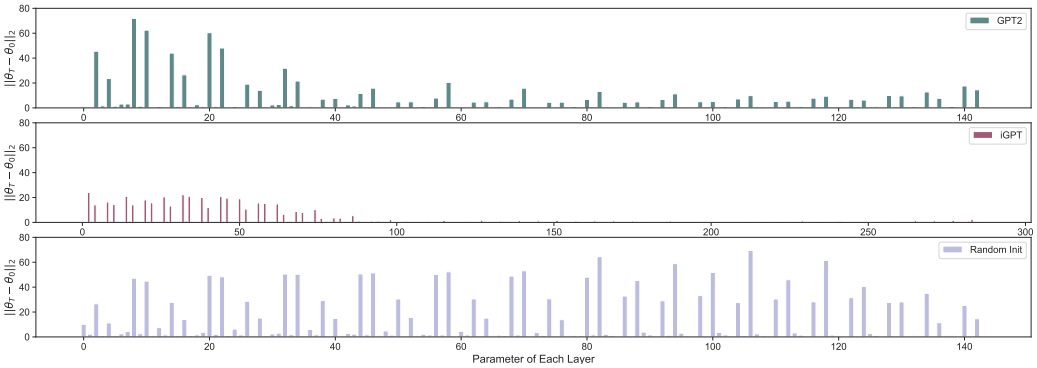


Figure 53: L2 distance of each parameter between pre post-fine-tuning (Walker2D, Seed = 42).

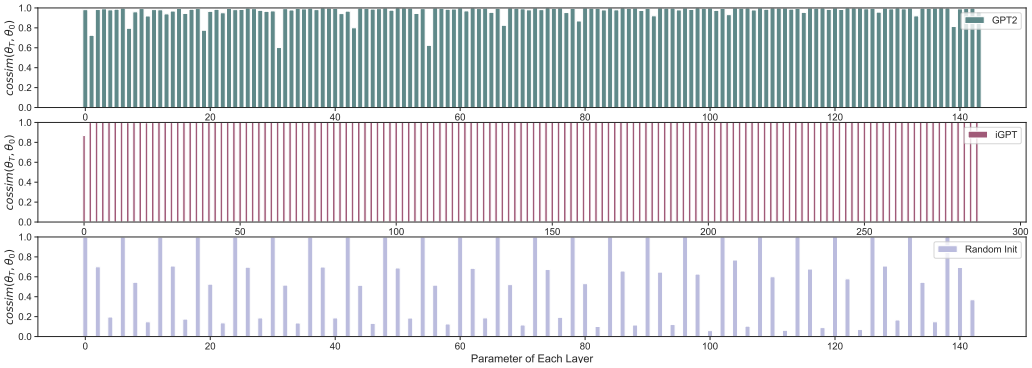


Figure 54: Cosine similarity of each parameter between pre post-fine-tuning (Hopper, Seed = 42).

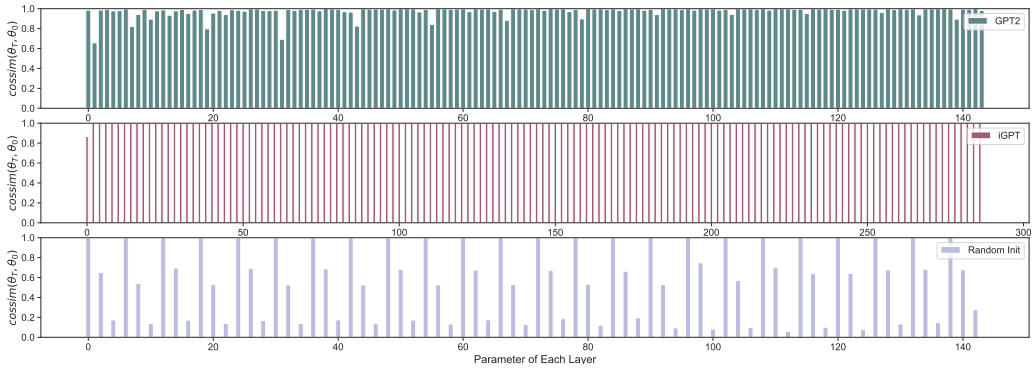


Figure 55: Cosine similarity of each parameter between pre post-fine-tuning (HalfCheetah, Seed = 42).

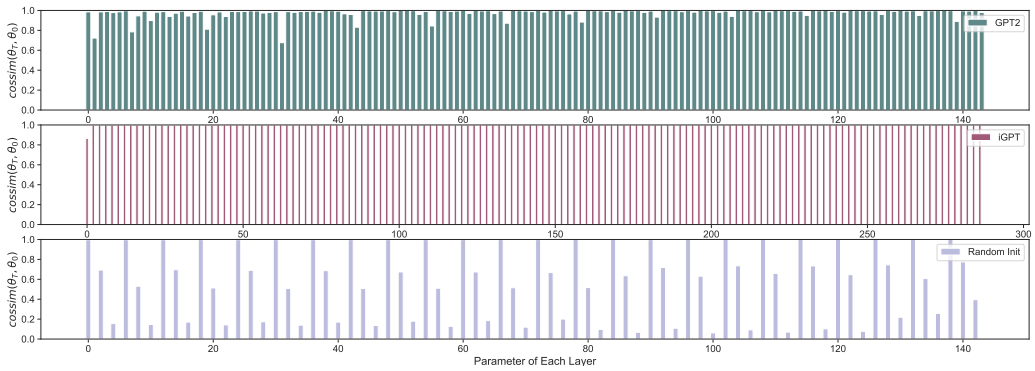
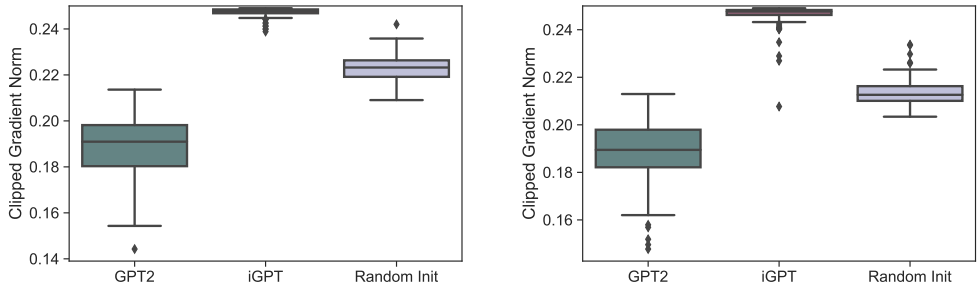


Figure 56: Cosine similarity of each parameter between pre post-fine-tuning (Walker2D, Seed = 42).

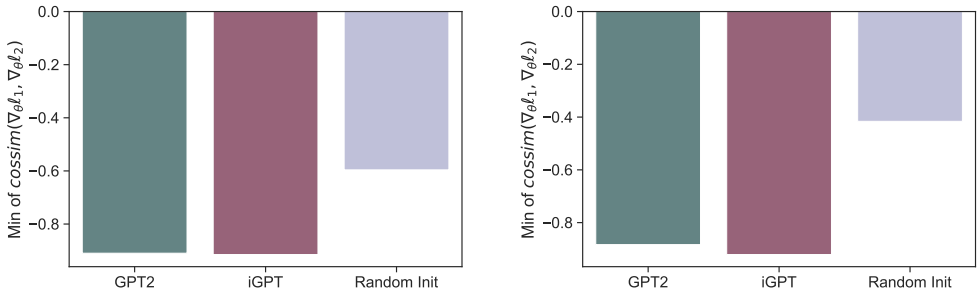
J.4 Gradient Analysis

J.4.1 Gradient Analysis (Other Environments)



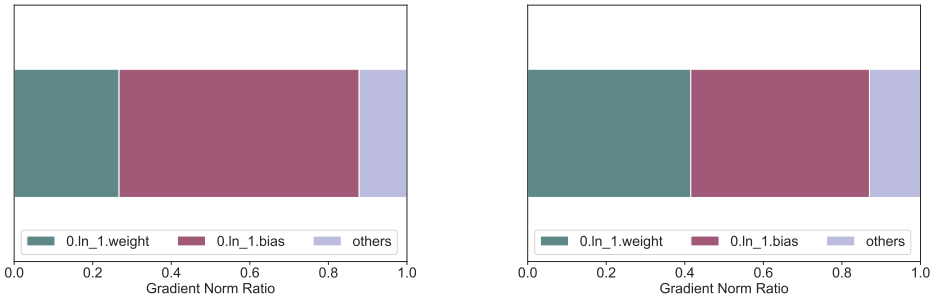
(a) HalfCheetah **(b)** Walker2D

Figure 57: Gradient norm.



(a) HalfCheetah **(b)** Walker2D

Figure 58: Minimum gradient cosine similarity.



(a) HalfCheetah **(b)** Walker2D

Figure 59: iGPT's gradient norm ratio.

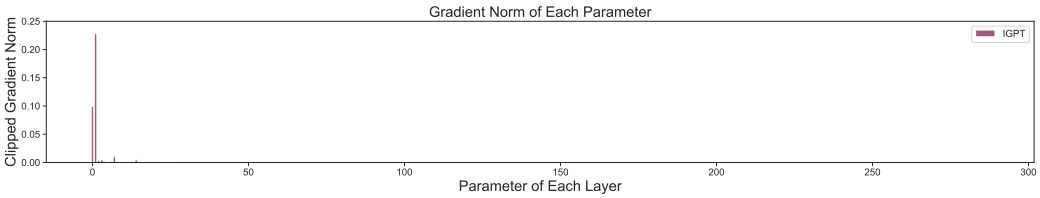


Figure 60: Gradient norm of iGPT's each parameter at epoch 1. (HalfCheetah)

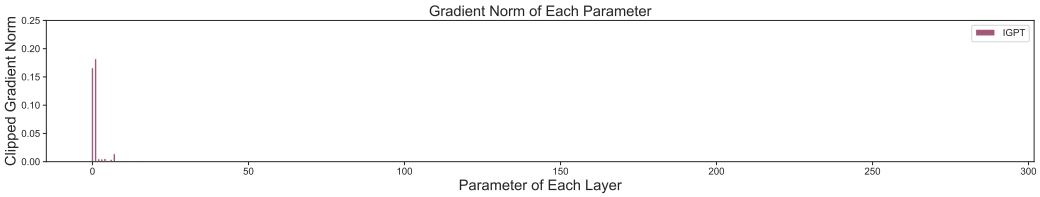
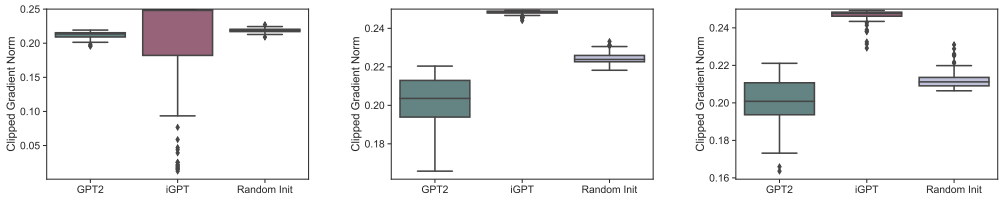


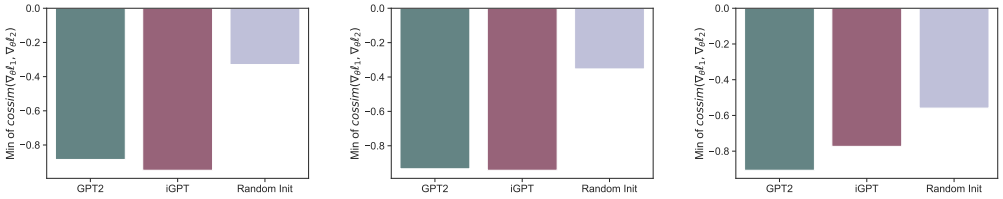
Figure 61: Gradient norm of iGPT's each parameter at epoch 1. (Walker2D)

J.4.2 Gradient Analysis (Seed = 42)



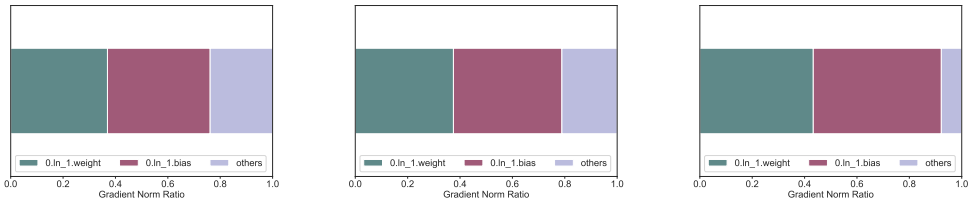
(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 62: Gradient norm (Seed = 42).



(a) Hopper (Seed = 42) **(b)** HalfCheetah (Seed = 42) **(c)** Walker2D (Seed = 42)

Figure 63: Minimum gradient cosine similarity (seed = 42).



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 64: iGPT's gradient norm ratio. (Seed = 42)

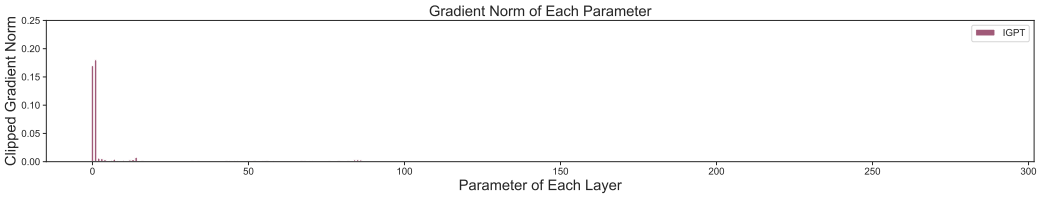


Figure 65: Gradient norm of iGPT's each parameter at epoch 1. (Hopper, Seed = 42)

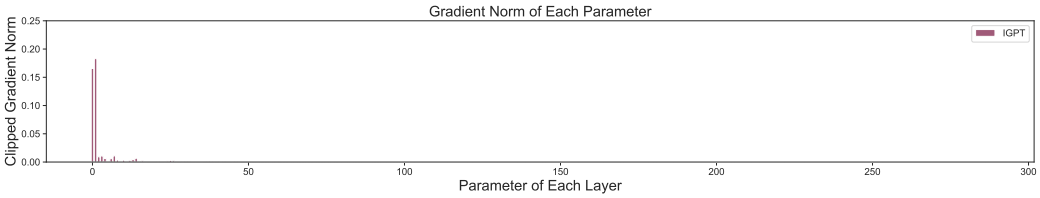


Figure 66: Gradient norm of iGPT's each parameter at epoch 1. (HalfCheetah, Seed = 42)

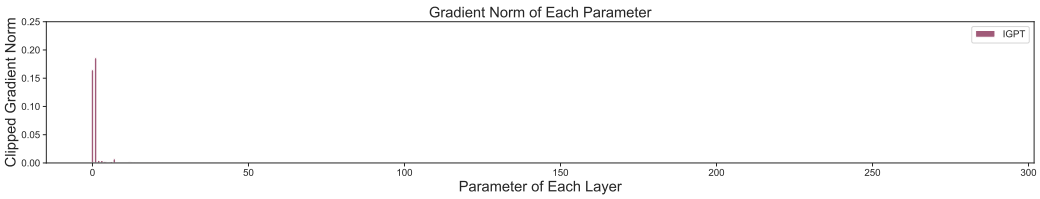
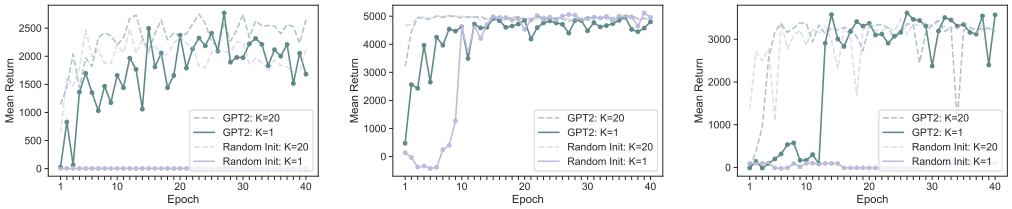


Figure 67: Gradient norm of iGPT's each parameter at epoch 1. (Walker2D, Seed = 42)

J.5 Fine-Tuning with No Context Information



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

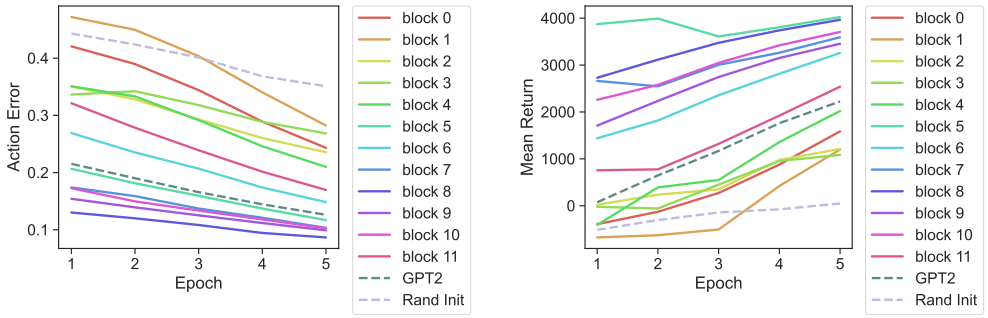
Figure 68: Mean return throughout fine-tuning when access to the context information is prohibited (seed = 42).

Table 4: Normalized return of $K = 1$ (seed = 666 & 42).

Dataset	Environment	GPT2	Random Init
Medium	Hopper	81.3 ± 2.5	-0.3 ± 0.1
	HalfCheetah	47.9 ± 0.1	49.2 ± 0.1
	Walker 2D	71.2 ± 2.2	35.6 ± 33.5

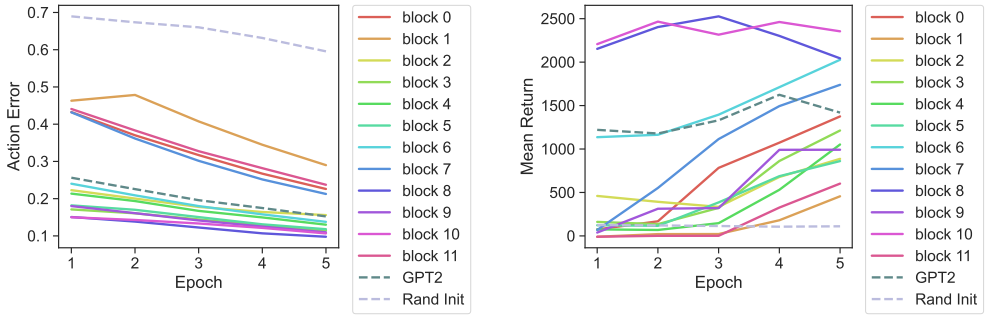
J.6 More In-Depth Analysis of Context Dependence

J.6.1 Replacement by the Pre-Trained Block



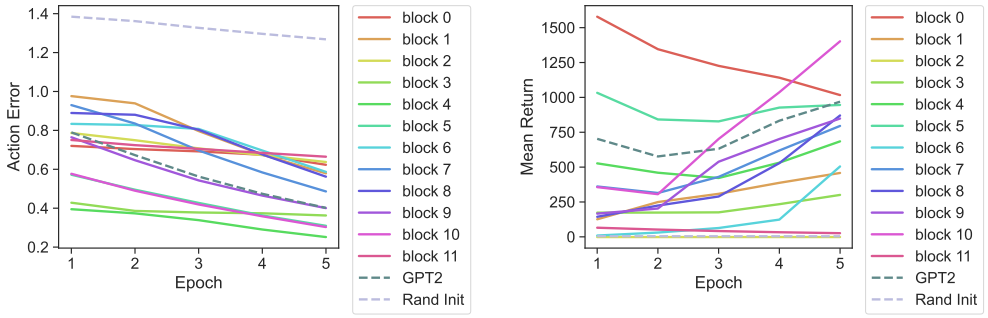
(a) Action error **(b)** Mean return

Figure 69: Learning curve when only a block is pre-trained (HalfCheetah, Seed = 666).



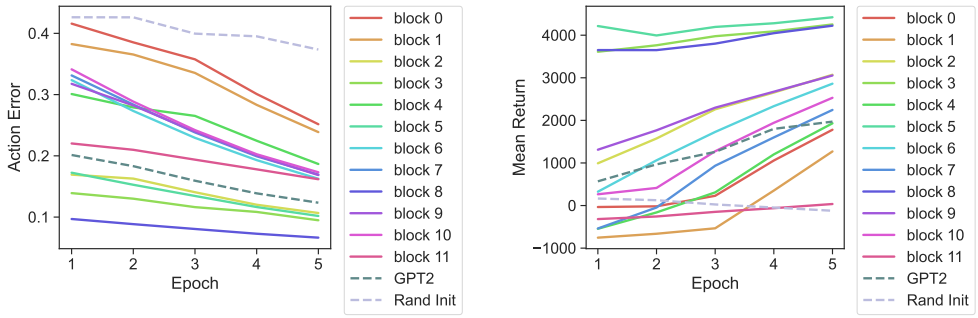
(a) Action error **(b)** Mean return

Figure 70: Learning curve when only a block is pre-trained (Walker2D, Seed = 666).



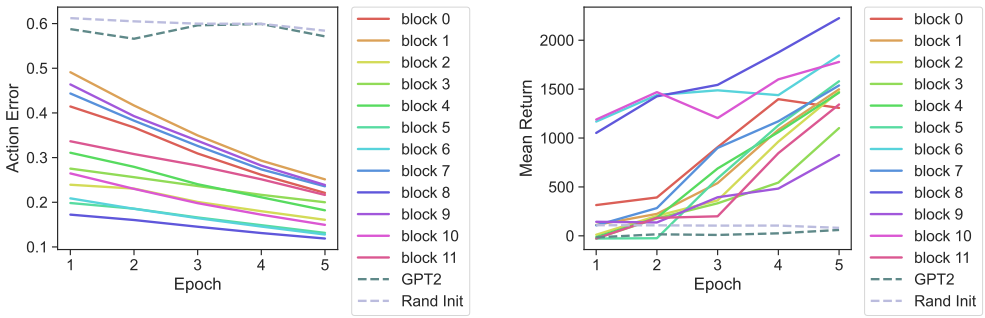
(a) Action error **(b)** Mean return

Figure 71: Learning curve when only a block is pre-trained (Hopper, Seed = 42).



(a) Action error **(b)** Mean return

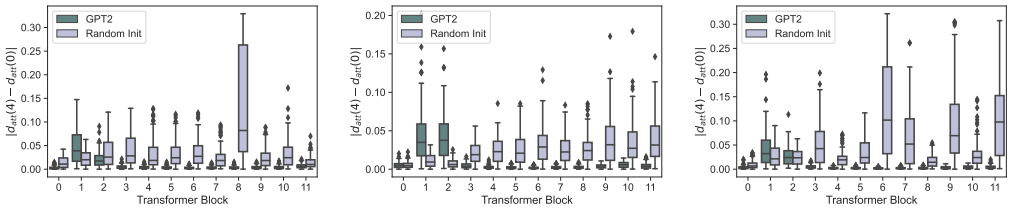
Figure 72: Learning curve when only a block is pre-trained (HalfCheetah, Seed = 42).



(a) Action error **(b)** Mean return

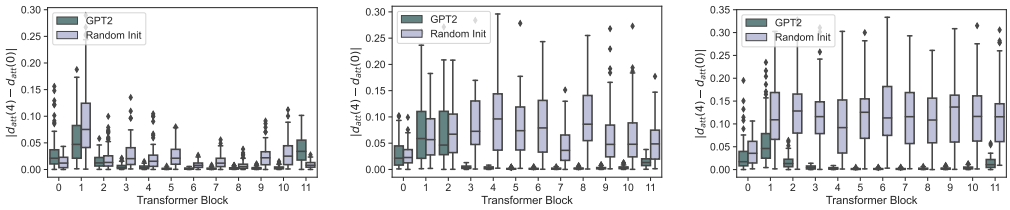
Figure 73: Learning curve when only a block is pre-trained (Walker2D, Seed = 42).

J.6.2 Attention Distance Analysis



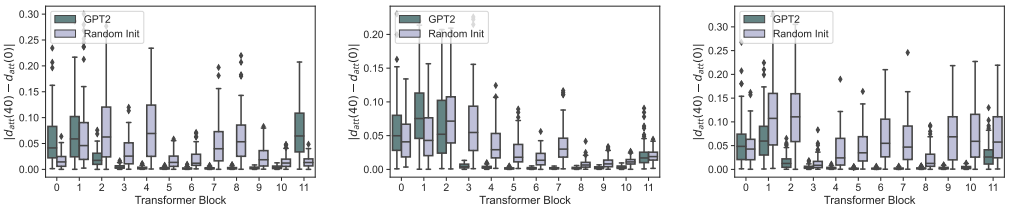
(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 74: Attention distance gap between epoch 0 and epoch 1.



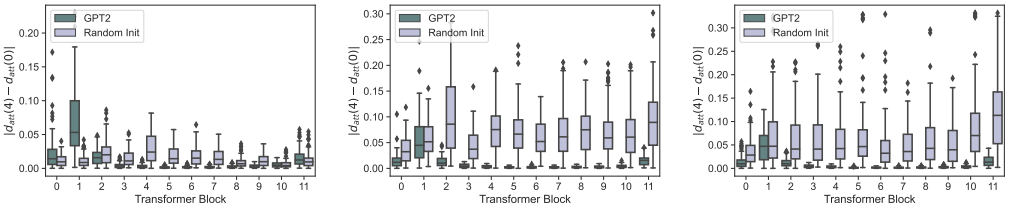
(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 75: Attention distance gap between epoch 0 and epoch 10.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 76: Attention distance gap between epoch 0 and epoch 40.



(a) Hopper **(b)** HalfCheetah **(c)** Walker2D

Figure 77: Attention distance gap between epoch 0 and epoch 4 (seed = 42).

K Details for Computation

The computational environment for our experiments is as follows:

- GPU: NVIDIA GeForce GTX TITAN X $\times$ 2GPU.
- CPU: Intel Xeon E5-2620 v3 2.4 GHz, 6 Cores $\times$ 2CPU.
- software: Ubuntu 20.04.4, GCC 9.4.0, CUDA 11.6.

The computational time of fine-tuning for 40 epochs is around from 24 to 72 hours for context length $K = 20$ and around from 7 to 48 hours for context length $K = 1$ per run.

L Licences of Assets Used for Our Experiments

L.1 Dataset

We use the offline RL dataset of Mujoco in D4RL [38]. The license of the dataset in D4RL is a Creative Commons Attribution 4.0 License (CC BY). Thus, we can use this dataset with no consent as long as we follow the term of use.

L.2 Code

The code we used does not require special consent from authors as long as we follow the terms of use. Their licenses are as follows:

- https://github.com/machelreid/can-wikipedia-help-offline-rl: MIT Licence.
- https://github.com/rail-berkeley/d4rl: Apache License 2.0.
- https://github.com/google-research/google-research/tree/master/representation_similarity: Apache License 2.0.
- https://github.com/gtegner/mine-pytorch: MIT License.

M Note on the Role of Each Sub-Sections in Section 5

Sections 5.1 and 5.2 show what may *not* be a cause of the good performance. In other words, these sections eliminate some seemingly possible causes of good performance of language-pre-trained models, respectively. These analyses further highlight the importance of attention distance for performance. Without the analysis of Section 5.1, we cannot exclude the possibility that good performance comes from re-using pre-trained representation *as well*, which is common in the uni-modal case as we explained in Section 5.1. Similarly, without the analysis of Section 5.2, the performance can benefit from fitting better into training data *as well*. In that case, it would be less clear whether utilizing some prior knowledge is solely important. Sections 5.4, 5.5, 5.6.1, and 5.6.2 discuss what may be a cause of the good performance, as summarized at the end of Section 5. Section 5.3 provides complementary findings for these sections. For example, the existence of some unchanged parameters implies the possibility that some information is re-used, while the relatively larger change in only shallower layers of pre-trained models partially supports that these unchanged parameters do not seem to contradict with changed representation observed in Section 5.1, as explained in Appendix N.

N Note on Relationship Between Parameter Change and Representation Change

In Section 5.1 we say that *representation* in some layers of the pre-trained model changes, while in Section 5.3 we say that *parameters* of them in some layers do not change that much, or vice versa for the randomly initialized model. These observations of Sections 5.1 and 5.3 are not necessarily contradictory because the change of parameters in layer ℓ does not correspond one-to-one with that in representation in the layer.

First, the representation of layer ℓ is affected by all parameters and input data prior to layer ℓ. So, even if the parameters of layer ℓ have not changed, if the parameters before layer ℓ have changed, the representation of layer ℓ may change. Second, since not all of the parameters of a neural network necessarily contribute to the output, the output of a layer may not change even if some parameters of that layer change. For example, if the input value to ReLU is negative, the output will remain 0 no matter how much the parameters involved change. So, for example, it is possible that even if the parameters of the ℓ-layer change, the representation of the ℓ-layer remains the same. Another example is when the values of two weights w_1^ℓ and w_2^ℓ in layer ℓ do not change the output of the vanilla feed-forward network because of its symmetry.

Another possible cause of this observation unique to the current analysis is that we use CKA to measure representational similarity. CKA is designed to be invariant to some transformations on the representation matrix [39]. Thus, different representation matrix is regarded as the same under these transformations even when parameters are changed.

O Note on Why Better Action Prediction Does Not Always Result in Better Return

In Section 5.2, we explained that randomly initialized models seem to predict action better, while in Table 1, they perform worse than GPT2. We will add a possible explanation of why better action prediction not necessarily comes to a better return.

The most likely cause of this observation is the current problem setup. As mentioned in Appendix B, the *medium* dataset is early stopped and thus does not necessarily converge to an optimal policy. This means that if the model accurately learns trajectory and predicts the next action, it does not always mean that it is the best action. Hence, we can see that a low action error does not necessarily mean a large mean return. However, We do not believe that the use of medium will hurt the validity of the analysis using this data since this dataset is not that pathological.

Another cause might be related to mutual information. The current analysis of mutual information shows that the representation of the hidden layer of the randomly initialized model holds more information as well of the input as well as the labels. Since the neural net representation is a vector representation if a single vector contains both types of information, these types of information

might have been mixed. This may make it difficult to properly utilize only the label information, making it harder to accurately predict the action. Another possibility is that the pre-trained model and the randomly initialized model differ in terms of which input token type (return-to-go, state, or action) information they encode. For example, in Fig. 2 (b), the language pre-trained model encodes information uniformly from the same token type: the hidden representation with high mutual information is the representation corresponding to the input from the triangles, i.e., the part corresponding to the return-to-go. On the other hand, in the randomly initialized models, there are variations (triangle, circle, and cross). It was noted in a previous study that attention weight was strong between the same token types [9]. Therefore, perhaps the pre-trained model might encode less total information, but the amount of information *effectively* utilized is not that different. Finally, as pointed out in the explanation about the limitation, the limitations of mutual information as a metric might be a cause.