NORMALIZED ATTENTION WITHOUT PROBABILITY CAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the popularity of attention based architectures like Transformers, the geometrical implications of $\mathrm{softmax}$-attention remain largely unexplored. In this work we highlight the limitations of constraining attention weights to the probability simplex and the resulting convex hull of value vectors. We show that Transformers are biased towards local information at initialization and sensitive to hyperparameters, contrast attention to max- and sum-pooling and show the performance implications of different architectures with respect to biases in the data. Finally, we propose to replace the $\mathrm{softmax}$ in self-attention with normalization, resulting in a generally applicable architecture that is robust to hyperparameters and biases in the data. We support our insights with empirical results from more than 30,000 trained models. Implementations are in the supplementary material.

1 INTRODUCTION

The concept of neural attention (Graves, 2013; Bahdanau et al., 2015) has sparked a number of architectural breakthroughs. The Transformer architecture (Vaswani et al., 2017) successfully deploys multi-headed self-attention in several consecutive layers for natural language processing (NLP) – a solution that has become increasingly popular (Vaswani et al., 2017; Radford et al., 2018; 2019; Brown et al., 2020; Devlin et al., 2019; Yang et al., 2019; Raffel et al., 2019; Liu et al., 2019). Apart from NLP, self-attention has shown success in applications ranging from image classification (Parmar et al., 2019; Dosovitskiy et al., 2021) to generative adversarial networks (Zhang et al., 2019) to reinforcement learning (Bram et al., 2019; Parisotto et al., 2019; Loynd et al., 2020). The attention architecture choice is thereby often based on one, if not both, of the following arguments: (1) Attention helps with the information flow by providing more direct, dynamic links between inputs and outputs. (2) Attention is directly interpretable as one can investigate the percentages to which different inputs are "attended" to. However, this second argument has been challenged recently, as several works show that attention weights do not directly correlate with predictions (Jain & Wallace, 2019; Wiegreffe & Pinter, 2019; Pruthi et al., 2019; Brunner et al., 2020; Pascual et al., 2020). With interpretability already in dispute, we focus on the other argument. Specifically, we question whether attention is the best way to route information and argue that apart from the proven success, $\mathrm{softmax}$ attention might not always be the best option.

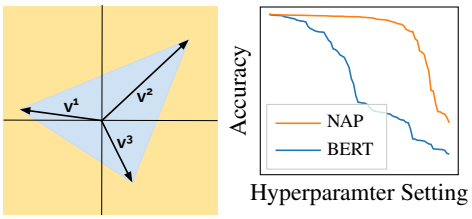

Figure 1: Softmax attention outputs can only lie within the convex hull spanned by the value vectors $\boldsymbol{v}^i$ (blue region). Removing this constraint with our normalized attention pooling (NAP) leads to an increased robustness to hyperparameters (See Section 5.1.1.)

In this work, we investigate the theoretical implications of constraining the attention weights to the probability simplex, and propose an unconstrained alternative based on normalization. We show that the popular Transformer architecture has an innate bias towards local information at initialization and showcase implications of this architectural bias on biases in the data. We investigate different architectures on an abstract level in numerous experiments and demonstrate the advantage of unconstrained attention. In particular, we improve robustness to hyperparameters and show the general applicability of attention based architectures as compared to sum and max pooling.

2 BACKGROUND AND RELATED WORK

Many data processing tasks can be addressed by representing the input as a set or sequence of discrete tokens, e.g., words in a sentence, frames in a video or points in a point cloud. As a general formulation, we represent each input token by a vector $\boldsymbol{x}^i \in \mathbb{R}^d$ for $i \in \{1, \ldots, N\}$, where N is the sequence length and d is the dimensionality of each token. For ease of notation we use the word "sequence" throughout, but note that all architectures discussed are also applicable to unordered sequences, i.e., sets of tokens. Multi-headed dot-product self-attention is a fundamental building block of the Transformer architecture (Vaswani et al., 2017). It allows for information exchange between different tokens of the input sequence. More formally, for each attention head m, the input vectors $\boldsymbol{x}^i$ are projected by an affine transformation to a query $\boldsymbol{q}_m^i$, key $\boldsymbol{k}_m^i$ and value vector $\boldsymbol{v}_m^i$. The dimensionality of these vectors is chosen as $d_h = \frac{d}{M}$, where M is the number of attention heads. The query and key vectors are used for a pairwise dot product, scaled by the square root of the head dimension d_h, to form the attention logits $l_m^{i,j}$ and attention vectors $\boldsymbol{a}_m^i$ as

$$l_m^{i,j} = \frac{< \boldsymbol{q}_m^i, \boldsymbol{k}_m^j >}{\sqrt{d_h}} \qquad \boldsymbol{a}_m^i = \mathrm{softmax}([l_m^{i,1}, \ldots, l_m^{i,N}])$$

where $\mathrm{softmax}$ refers to the normalized exponential function $\mathrm{softmax}(\boldsymbol{x})^j = \frac{\exp(x^j)}{\sum_k \exp(x^k)}$ commonly used to project vectors to the probability simplex $\mathcal{S}_P = \{\boldsymbol{a}_m^i | a_m^{i,j} \geq 0 \, \forall j \text{ and } \sum_j a_m^{i,j} = 1\}$. The output $\boldsymbol{o}_m^i$ of each attention head m is then given by a weighted sum of all value vectors $\boldsymbol{o}_m^i = \sum_j a_m^{i,j} \cdot \boldsymbol{v}_m^j$. These attention head outputs are concatenated and mixed trough an affine transformation to form the attention output in the Transformer architecture (Vaswani et al., 2017).

In this work, we investigate whether constraining the attention vectors $\boldsymbol{a}_m^i$ into the probability simplex through the $\mathrm{softmax}$ function is sensible from a geometric perspective. We contrast the multi-head self-attention architecture to attention-inspired architectures without $\mathrm{softmax}$ (discussed in Section 4) as well as simpler aggregation methods commonly used. Specifically, while Yun et al. (2020) show that Transformers are universal sequence-to-sequence function approximators, we question the practical necessity of an attention architecture, when sum pooling (Zaheer et al., 2017) already provides general function approximation capabilities (Zaheer et al., 2017; Xu et al., 2019; Segol & Lipman, 2020). Further, we compare to max pooling, a common aggregator choice that has shown good empirical success (Nagi et al., 2011; Zaheer et al., 2017; Veličković et al., 2020). Several works have proposed architectural changes to the Transformer (Yun et al., 2020; Wu et al., 2019; Lan et al., 2020; Dehghani et al., 2019; Fan et al., 2020; Wu* et al., 2020; Press et al., 2019; Bachlechner et al., 2020), however, these do not alter the multi-head self-attention. Another direction of research has focused on improving the computational efficiency of Transformers (Wang et al., 2020; Lee et al., 2019; Katharopoulos et al., 2020; Zaheer et al., 2020; Choromanski et al., 2021). However, these either retain the softmax (Lee et al., 2019; Wang et al., 2020; Zaheer et al., 2020) or try to stay close to it (Katharopoulos et al., 2020; Choromanski et al., 2021). We refer to Tay et al. (2020) for a more extensive overview on works in this direction. Our focus is different: we ask whether the softmax is necessary at all. The Synthesizer (Tay et al., 2021) and MLP-Mixer (Tolstikhin et al., 2021) show that an aggregation with learned aggregation weights can already lead to good results. However, these aggregation schemes make the models dependent on the input sequence length – a trait we explicitly seek to avoid. Closest related to our work is the work of Tay et al. (2019), which introduces an attention mechanism that allows for negative attention values. While their intuition inspired architecture shows improved performance in NLP tasks, we provide insights into the theoretical limitations and data-bias implications of $\mathrm{softmax}$-self-attention.

3 IMPLICATIONS OF SOFTMAX ATTENTION

To start our discussion, we highlight an observation that follows directly from attention vectors $\boldsymbol{a}^i$ being constrained to the probability simplex $\mathcal{S}_P$:

Outputs $\boldsymbol{o}_m^i$ are convex combinations of the vectors $\boldsymbol{v}_m^i$: This in itself has interesting implications. First and foremost, we note that a convex combination of vectors $\boldsymbol{v}_m^i$ cannot yield any vector outside the convex hull spanned by the value vectors $\boldsymbol{v}_m^i$. An illustration of this output "cage" is given in Figure 1 (left). We conjecture that this constraint limits the hypothesis space of the optimization, making it more difficult for gradient descent to find a good solution. This conjecture is

supported by our experimental results showing an increased robustness to hyperparameter choices when the constraint is removed. Further, we note the following from a theoretical perspective:

No convex combination can in itself represent XOR: A formal proof is given in Appendix A. Note that this implication highlights an inability to represent non-linearity. While the binary exclusive OR can be represented in architectures with multiple heads and layers, the insight further underlines our argument: An aggregation with weights constrained to the probability simplex is restrictive. The convex hull view point also leads to an additional insight that, to the best of our knowledge, has not been discussed in the literature so far:

Transformers are not aggregation size independent: Specifically, Transformers focus on local information at initialization; and this focus depends on the sequence length N. To see this, consider the embeddings $\boldsymbol{e}^i$ after the first residual connection:

$$\boldsymbol{e}^i = \boldsymbol{x}^i + \boldsymbol{W}\left[\sum_j a_1^{i,j} \cdot \boldsymbol{v}_1^j \middle| \ldots \middle| \sum_j a_M^{i,j} \cdot \boldsymbol{v}_M^j\right] + \boldsymbol{b}$$

where $[\cdot|\cdot]$ denotes concatenation and $\boldsymbol{W}$ and $\boldsymbol{b}$ represent the parameters of the affine transformation that mixes the M attention head outputs to form the attention layer output. Our aim is to show how much the embedding $\boldsymbol{e}^i$ is influenced by the local information $\boldsymbol{x}^i$ relative to the context information $\{\boldsymbol{x}^j | j \neq i\}$. We first note that the contribution of context information depends on the initialization of $\boldsymbol{W}$, where a typical initialization in language models yields a contractive projection.[1] This favors the residual connection, i.e., local information. However, even if we consider $\boldsymbol{W}$ as scale preserving, we note that the magnitudes of the attention head outputs $\boldsymbol{o}_m$ are upper bounded by the magnitudes of the value vectors $\boldsymbol{v}_m$ as a result of the convex hull. Moreover, attention logits are normally close to 0 at initialization (to have the softmax in the unsaturated region). This yields attention to be close to mean aggregation as $\boldsymbol{o}_m^i \approx \bar{\boldsymbol{v}}_m = \frac{1}{N}\sum_j \boldsymbol{v}_m^j$. Considering $\boldsymbol{v}_m^j$ to be distributed with zero-mean and variance $\sigma_{\boldsymbol{v}}^2$ at initialization, we note that the expected magnitude and standard deviation of $\bar{\boldsymbol{v}}_m$ scales proportional to $\frac{\sigma_{\boldsymbol{v}}}{\sqrt{N}}$. This means that the fraction of context information in $\boldsymbol{e}^i$ is dependent on the sequence length N and, counterintuitively, is smaller for longer sequences. Specifically, with the constant contribution of the residual connection, Transformers focus more on local information; and more so in longer sequences than in shorter sequences. For reference, we visualize the dependence of $\boldsymbol{o}_m$ on N at initialization for different aggregators in Figure 2. We note that while an architectural bias towards local information might be beneficial in some applications, the implicit dependence on aggregation size N is questionable. Finally, we provide additional insights to a fact that is often overlooked:

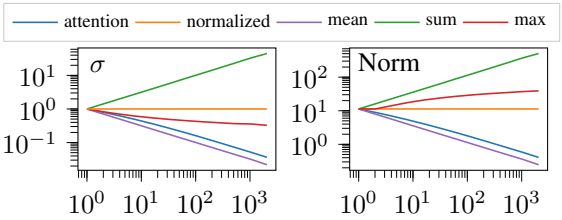

Figure 2: The standard deviation (σ) and norm of a pooling output depends on the sequence length N (x-axis) if the output is not normalized. Softmax attention outputs (blue) scale similar to mean pooling at initialization, i.e., Transformers focus more on local information in longer sequences. Experiment details are given in Appendix C.

A softmax can easily saturate, which leads to vanishing gradients: To show this, we look at the term $g = \frac{\delta a_m^{i,j}}{\delta l_m^{i,j}}$ which gives a gradient factor on the back-propagation path through the softmax. Specifically, we note that $g \in (0, 0.25]$ and that the range of $l_m^{i,j}$ for which $g > 0.01$ is less than 8.5. This means that any gradient update that changes $l_m^{i,j}$ by a constant of $c > 8.5$, e.g. by updating the bias terms, will most certainly lead to vanishing gradients henceforth. This is particularly worrisome in the Transformer architecture, where gradients correlate with the input due to the multiplicative interactions in the attention. This means that a single large input can result in a parameter update that yields vanishing gradients on all following data points. We provide a more in depth discussion and proofs for all claims made here in Appendix B.

[1] As an example, BERT (Devlin et al., 2019) initializes $\boldsymbol{W}$ with parameters drawn from a truncated normal distribution with standard deviation set to 0.02. This would only be roughly scale preserving for $d = 2500$.

4 NORMALIZED ATTENTION POOLING

Given the implications of $\mathrm{softmax}$ attention, one might seek alternatives. To investigate different architectural choices, we contrast the following architectures in an abstract empirical study:

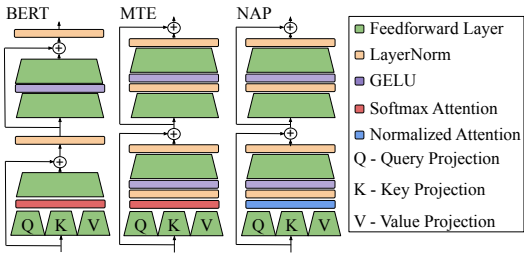

Figure 3: Difference in 1 Transformer layer.

Transformer Encoder (BERT): As a starting point, we replicate the encoder architecture presented by Vaswani et al. (2017) as described in the code release of Devlin et al. (2019).[2] This architecture has been used extensively (Radford et al., 2018; Yang et al., 2019; Raffel et al., 2019; Liu et al., 2019; Clark et al., 2020). Each Transformer-layer consists of two sub-modules: a multi-head self-attention "layer" and a feed forward network. Both modules are encompassed by residual connections. The multi-head self-attention "layer" consists of a projection to queries, keys and values, the attention mechanism as well as a mixing layer as described in Section 2. The feed forward network consists of two layers with a GELU (Hendrycks & Gimpel, 2016) non-linearity. Layer normalization (Ba et al., 2016) is applied *between* incoming and outgoing residual connections. Note that this gives a crucial distinction of this architecture: Embeddings are normalized *after* they are summed with the residual connection. This yields the implicit dependence on sequence length as discussed in Section 3. We train this architecture with learning rate warm-up and gradient norm clipping (cf. Devlin et al. (2019)).

Modified Transformer Encoder (MTE): To overcome the implicit dependence on sequence length, we modify the architecture by moving the layer normalizations and adding additional normalizations as shown in Figure 3. Note that this is different from the recently studied PreNorm (Parisotto et al., 2019; Nguyen & Salazar, 2019; Liu et al., 2020) that places the normalization before the attention mechanism. To further remove cofounding factors, we remove learning rate warm-up and gradient clipping, but keep a linearly decreasing learning rate schedule, taking Li et al. (2020) as reference. We provide an ablation of all modifications in Appendix D.1. All following architectures apply the same modifications. The resulting *MTE* architecture still projects attention weights to the probability simplex in the multi-head attention and is thereby limited to convex combinations of value vectors.

Normalized Attention Pooling (NAP): Given the success of online normalization during training - be it through batch- (Ioffe & Szegedy, 2015), layer- (Ba et al., 2016) , group- (Wu & He, 2020), instance- (Ulyanov et al., 2016) or weight-normalization (Salimans & Kingma, 2016) - our main proposal is to simply replace the softmax with a normalization:

$$\boldsymbol{a}_m^i = \mathrm{normalize}([l_m^{i,1}, \ldots, l_m^{i,N}]) \qquad \text{with } \mathrm{normalize}(\boldsymbol{x})^j = g \cdot \frac{x^j - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}} + b$$

where $\mu_{\boldsymbol{x}} = \frac{1}{N}\sum_j x^j$ and $\sigma_{\boldsymbol{x}} = \sqrt{\frac{1}{N}\sum_j (x^j - \mu_{\boldsymbol{x}})^2}$ are the mean and standard deviation of the corresponding input vector $\boldsymbol{x}$, in our case the logit vector calculated through key-query dot products. Similar to layer normalization (Ba et al., 2016), we introduce gain and bias parameters g and b initialized to 1 and 0, respectively. However, while Ba et al. (2016) introduce gain and bias vectors, we only introduce scalar parameters and broadcast these over the sequence, as we want the architecture to be independent of the sequence length N. Note that while no convex combination can represent the logical XOR, a normalized weighting can – we give a proof in Appendix A.

No Online Logit Normalization (NON): To investigate whether a dynamic normalization of the attention logits is necessary, we also train a model where we use the logits $l_m^{i,j}$ directly as attention weights, i.e., $\boldsymbol{o}_m^i = \mathrm{GELU}(\frac{1}{\sqrt{N}}\sum_j l_m^{i,j} \cdot \boldsymbol{v}_m^j)$. The factor $\frac{1}{\sqrt{N}}$ is introduced such that the model has in expectation a constant contribution of context at initialization.

Simple Summation of Embeddings (sum): From a theoretical perspective, summation is sufficient for general function approximation (Zaheer et al., 2017; Xu et al., 2019; Segol & Lipman, 2020). Therefore, we investigate to simply replace attention with a sum-reduce-broadcast operation.

[2] `https://github.com/google-research/bert`

```
// Case distinction task data generator
inputs ← random integer sequence
if 64 in inputs              // argmin case
   label ← argmin(inputs)
else if 50 in inputs         // first case
   label ← 0
else                         // argmax case
   label ← argmax(inputs)
return (inputs, label)
```

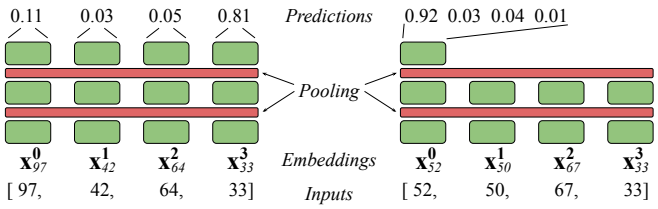

Figure 4: **Left:** Pseudo code of the data generation. The case distinction points 64 and 50 are chosen arbitrarily. **Right:** Task setup for outputs across all tokens (cf. Section 5.1) and outputs from the first token (cf. Section 5.2). Green boxes represent the trainable network layers (shared across tokens) while red boxes represent the pooling across tokens. The targets of the displayed examples would be [0, 0, 0, 1] and [1, 0, 0, 0], respectively.

Max Pooling over Sequence Dimension (max): Similar to sum pooling, we can replace the attention sub-module with a simple max-reduce-broadcast operation over the sequence dimension. Note that max pooling is a powerful operation that yields a direct link to up to d different tokens.

We provide a schematic figure of all architectures in Appendix D. If not varied in a corresponding experiment, we default architecture hyperparameters to $L = 2$ Transformer-layers (consisting of an attention sub-module and feed forward sub-module each), $M = 4$ heads to calculate the logits (if applicable), $d = 128$ as model dimension and train on a total of 3200 batches of 32 example sequences each, using the Adam optimizer (Kingma & Ba, 2014). The hidden dimension of the feed forward sub-modules is $4 \cdot d$ for the models *BERT*, *MTE*, *NAP* and *NON*. For the models *sum* and *max* we increase the feed forward hidden dimension to approximately match the parameter count.

5 Experiments and Results

Our goal with this work is to provide an insight into the variety of performance implications that the architecture choices entail. We aim to provide these insights independent of any particular application, as these architectures can be applied to a variety of tasks – from NLP (Vaswani et al., 2017; Devlin et al., 2019) to graph neural networks (Velickovic et al., 2018; Yun et al., 2019; Xu et al., 2019; Veličković et al., 2020) to reinforcement learning agents (Parisotto et al., 2019; Fang et al., 2019; Loynd et al., 2020). We therefore focus on carefully crafted synthetic tasks that (1) are general enough in that we can expect the insights to generalize to a large set of downstream tasks and (2) let us modify key aspects that are hidden in real world data sets, such as a bias towards a certain sub-task. The focus on synthetic tasks also allows us to get a better grasp on the learning dynamics – the focus of this work – as we can train thousands of models in diverse hyperparameter combinations. To limit the influence of confounding hyperparameters, we generate new data points for every batch. This allows us to omit regularizations and their confounding factors. See Appendix D.2 for an in depth discussion of this setup as well as regularized ablations.

5.1 Argmin-First-Argmax Case Distinction Task

As a first task, we train the networks to pin-point a specific, input dependent token. Note that the ability to pin-point a specific token is an abstract task relevant to NLP (e.g., question answering or co-reference resolution), graph neural networks (e.g., finding the next hop in a shortest path) as well as reinforcement learning (e.g., action credit assignment). We consider an input pipeline where tokens from a fixed integer-vocabulary are translated to a randomly initialized embedding. To the embedded tokens, a (also randomly initialized) positional embedding is added to provide position-relative information. The sequence of tokens is then processed by several architecture dependent Transformer-layers (cf. Section 4). Finally, each contextualized embedding is projected to a single output. A softmax-crossentropy loss is applied over the sequence dimension. See Figure 4 (middle) for a visualization. To make the task input dependent, we generate the data as given in the pseudo code in Figure 4 (left). Note that the `argmin` and `argmax` make this task quite challenging from a learning perspective as the networks start from random embeddings which do not provide any ordering information. Which embeddings correspond to bigger integers and which to smaller

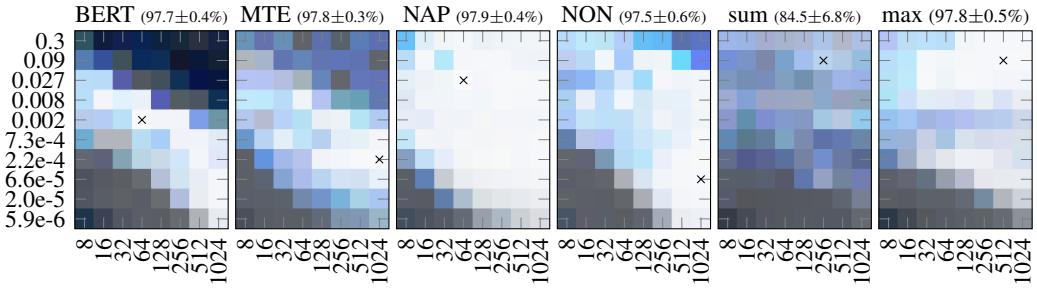

Figure 5: Learning rate (y-axis) vs. model dimension d (x-axis) on the argmin-first-argmax case distinction task (with output across all tokens). The pixels' R (red), G (green) and B (blue) values correspond to min-, mean- and max-accuracy, respectively, of the corresponding hyperparameter combination – see main text for details. The plot shows the validation accuracy when validating on sequences of length $N = 64$. Crosses indicate the combination for best validation accuracy, which we report with standard deviation behind the model name.

integers has to be inferred during training. Further, the case distinction in this task lets us tweak the data bias towards each sub-task. Specifically, we consider a vocabulary size of $S = 100$ integers (0-99) and uniformly random sampled sequences of $N = 128$ tokens in length. This leads to a bias as $p_{argmin} = 1 - (1 - \frac{1}{S})^N \approx 72.4\%$ of data points require the network to pin-point the minimum in the input sequence, $p_{first} \approx 20.1\%$ require to pin-point the first token of the sequence and $p_{argmax} \approx 7.5\%$ require to pin-point the maximum in the input.

5.1.1 Varying Model Dimension d

As a first investigation, we are interested in how varying the model dimension d influences the architectures ability to learn the given task. For this, we train each of the architectures for each of the model dimensions $d \in \{8, 16, 32, 64, 128, 256, 512, 1024\}$ using 10 different learning rates and 5 random seeds for each hyperparameter combination. As we want to base our insights on as many results as possible, we derive a novel, human friendly visualization of results. Figure 5 shows the first results as follows: The outcome of each hyperparameter combination is reported as an RGB pixel in the plot, where the R (red) value corresponds to the accuracy of the worst performing random seed, the G (green) value corresponds to the average over the random seeds and the B (blue) value corresponds to the best performing random seed. For each value (R, G and B), the max over the course of training is taken. This assignment roughly translates as follows: The brighter, the better - brighter pixels correspond to higher min-, mean- and max-accuracy. Blue/turquoise pixels highlight a large performance variation across random seeds and black/grey pixels correspond to hyperparameter combinations where none of the random seeds could solve the task. Given that all architectures are applicable to sequences of any length, we investigate how the architectures generalize to sequences of different length. Specifically, we validated each of the models trained on sequences of length $N = 128$ after every 100 batches on 32 batches with sequences of half the length ($N = 64$). The condensed results in Figure 5 give rise to the following observations: (1) Most models have some hyper-parameter combinations that learn the task well (white pixels). (2) The optimal learning rate depends on the model size, especially in the *BERT* architecture. This has profound implications for hyperparameter optimization: Tuning hyperparameters independent of each other might lead to sub-optimal results. (3) Models with probability simplex limitations (*BERT* and *MTE*) work for a smaller range of hyperparameters. (4) Our *NAP* architecture seems to be the most robust to this generalization. We provide case learning curves and additional results in Appendices E and F. Specifically, to be sure, we repeat this experiment with $L = 4$ and $L = 6$ Transformer layers, but the results stay the same (see Appendix F.1).

5.1.2 Case Accuracy under Varying Data Biases

As a next experiment we reset the model dimension to $d = 128$ and investigate the models under varying data biases. We do so in two distinct ways: First, we vary the sequence length $N \in \{4, 8, 16, 32, 64, 128, 256, 512\}$. Note that this implicitly varies the biases p_{argmin}, p_{first}

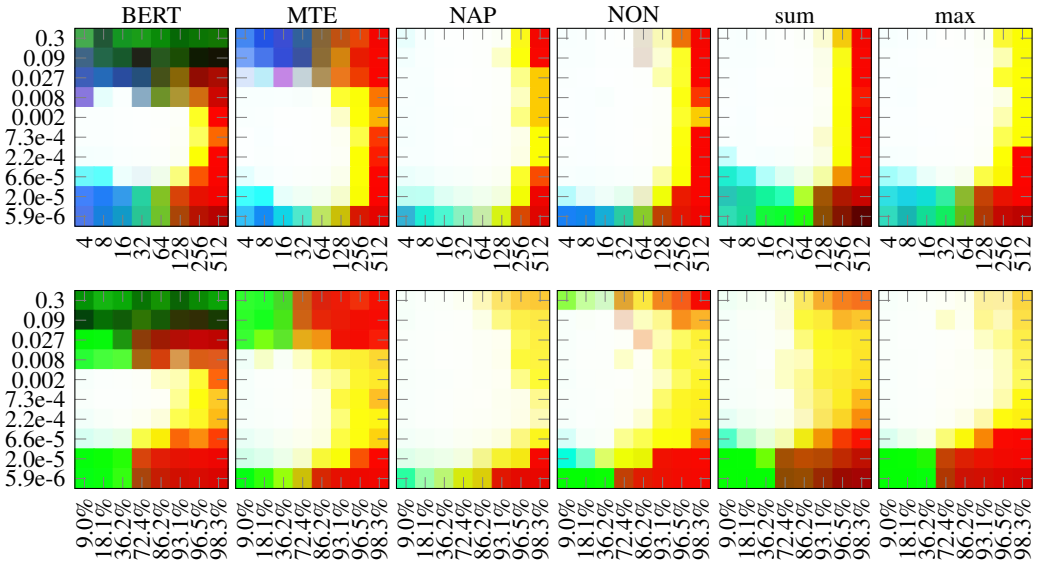

Figure 6: Biased data results on the case distinction task (with output across all tokens). RGB pixel values correspond to *argmin*-, *first*- and *argmax*-mean-case-accuracies, respectively. **Top row:** Learning rate (y-axis) vs. sequence length N (x-axis). **Bottom row:** Learning rate (y-axis) vs. percentage of *argmin*-case in the data (x-axis) with fixed $N = 128$.

and p_{argmax} in the data. Second, we fix the sequence length at $N = 128$ and vary p_{argmin} by explicitly adding or removing corresponding data points. We report the case specific accuracies in Figure 6 as follows: After every 100 batches, we validate the models on 1000 examples per case. Reported is the best accurracy over the course of training in form of pixel value with R (red) corresponding to the *argmin*-case accuracy, G (green) corresponding to the *first*-case accuracy and B (blue) corresponding to the *argmax*-case accuracy. As a consequence, white pixels correspond to all cases learned and yellow pixels correspond to the *argmin*- and *first*-case learned. We make the following observations: (1) If the learning rate is too low, models tend to focus on the majority case (indicated in a shift from blue to red as the bias shifts from the *argmax*- to the *argmin*-case with increasing sequence length N and the shift from green to red for fixed N). (2) If the learning rate is too high, the *BERT* architecture tends to focus on the *first*-case. We believe this is due to the architectural bias towards local information as discussed in Section 3. Note that the *first*-case can be solved by relying on the local positional embedding. (3) Only the *NAP* and *max* architecture manage to learn all three cases from highly biased data (see $N = 256$ and $p_{argmin} = 96.5\%$). This shows that the NAP architecture is not only robust to hyperparameters but also robust to biases in the data. In Appendix F.2 and F.3 we provide further experiments investigating different batch sizes and initialization scales. In Appendix F.4 we investigate the local vs. global aggregation focus further. The results are complementary.

5.2 First Token Output

The task so far requires the architectures to learn an information flow between tokens to distinguish the case and decide per token, whether it is the token that is looked for or not. Now we investigate, whether all this information can also be aggregated into a single token. We therefore modify the architecture output slightly in that we only take the contextualized embedding of the first token and project from it to a vector of size N (see example on the right in Figure 4). Note that this task set-up is harder and can highlight bottlenecks in the information flow across tokens. We fix the sequence lenght to $N = 128$ and again vary the model dimension d. We report the case specific mean accuracies in Figure 7. Min-, mean- and max-overall-accuracies are given in Appendix F.5. We observe: (1) All architectures learn for (almost) all combinations the now close to trivial *first*-case, even though it is not the majority case. (2) The *sum pooling* architecture does not learn any of the other cases. (3) Only *NAP* and *max* learn all three cases in some hyperparameter combinations.

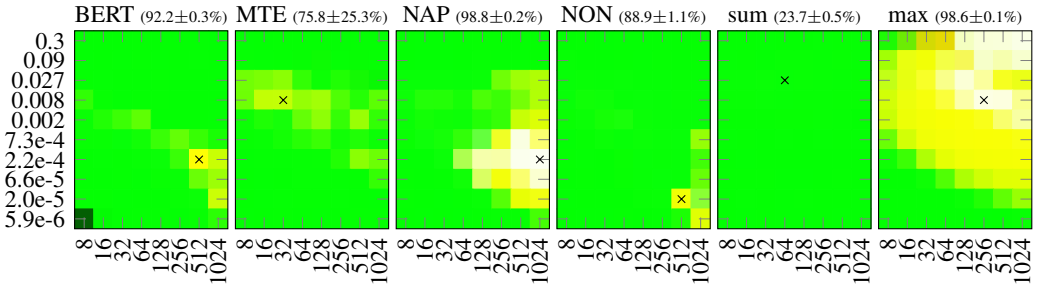

Figure 7: Learning rate (y-axis) vs. model dimension d (x-axis) on the case distinction task with output from the first token. RGB pixel values correspond to the case accuracies. Crosses indicate the best accuracy, reported behind the model name.

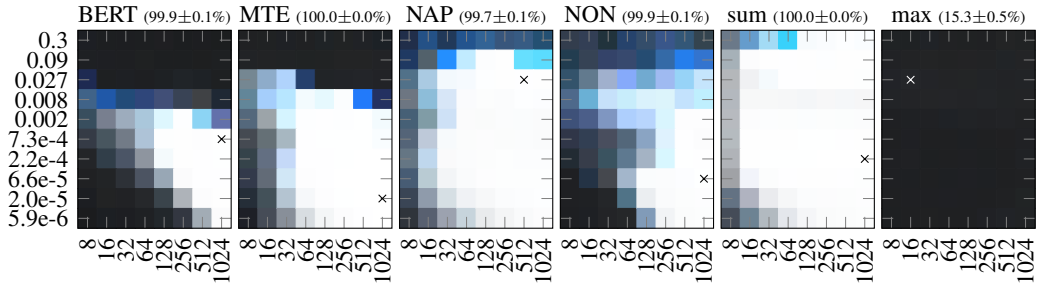

Figure 8: Learning rate (y-axis) vs. model dimension d (x-axis) on the mode finding task. RGB pixel values correspond to min, mean and max accuracy. Crosses indicate the combination for best accuracy, reported behind the model name.

The worse performance of *NON* highlights the advantage of online normalization of the logits. While the softmax provides some form of online normalization, we hypothesize that the worse performance of *MTE* and *BERT* in this task stems from an information bottleneck induced by the probability simplex limitations. To test this hypothesis further, we vary the number of attention heads M. The results are shown in Figure 10. We observe that increasing the number of heads helps the *MTE* and *BERT* architecture, supporting our hypothesis. Note however, that *MTE* and *BERT* are still outperformed significantly by *NAP*. In Appendix F.6 we provide a further experiment, varying the depth up to $L = 64$. Results are complementary.

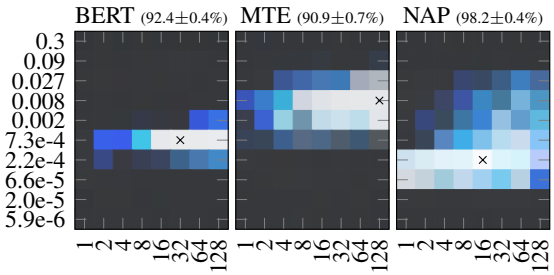

Figure 10: Learning rate (y-axis) vs. attention heads M (x-axis) on the case distinction task (first token output). RGB pixel values correspond to min, mean and max accuracy. Crosses indicate the combination for best accuracy, reported behind the model name.

5.3 Mode Finding Task

Given the results so far, one could conclude that *max* is the best choice due to its simplicity. Note however, that *max* has an architectural prior that is in line with the underlying task of finding the maximum or minimum of the sequence. To study the effect of architectural priors, we experiment on an additional task: Finding the mode/most common integer in the input sequence. Also this task has ties to NLP (e.g., sentiment analysis), graph neural networks (e.g., consensus/agreement) and reinforcement learning (e.g., count based exploration). Here we remove the positional embeddings, as this task can also be done on sets, and project from the contextualized embedding of the first token to a vector of dimension S (the vocabulary size) over which we apply the softmax-cross-

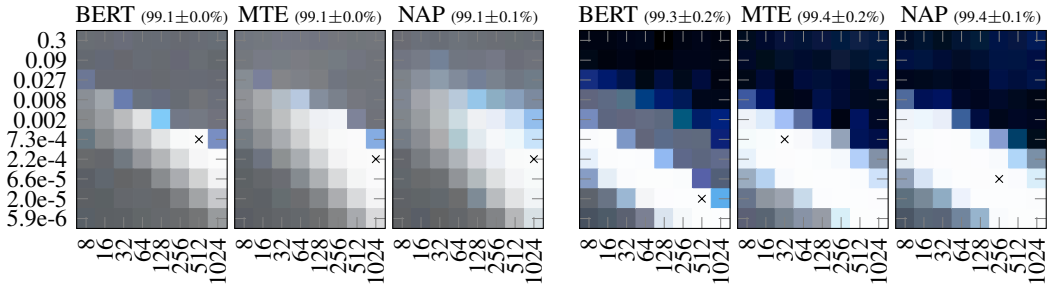

Figure 9: Learning rate (y-axis) vs. model dimension d (x-axis). RGB pixel values correspond to min, mean and max validation performance. Crosses indicate the best combination (validation), from which we report the test performance. For *NON*, *sum* and *max* see Appendices H and I. **Left:** Protein-protein-interaction task. Shown is the node classification F1-score. **Right:** Altered working memory graph agent in the Baby-AI level 3 reinforcement learning task. Shown is the success rate.

entropy loss. We keep $N = 128$ but reduce S to 10 to have meaningful modes. Ties are broken by taking the smallest integer of the ones with maximal occurrence. Results of varying the model dimension d are reported in Figure 8. We observe: (1) *sum pooling* works well on this task, as it has a suitable architecutral prior. (2) *max pooling* cannot learn the task, not even with a model dimension $d = 1024 = 8 \cdot N$. In Appendix G.1 we provide complementary results varying the vocabulary size. We refer an interested reader to Xu et al. (2020) for more on architecture-task alignment.

5.4 Generalization to Graph Neural Networks and Reinforcement Learning

NLP experiments normally require large amounts of data and large models – a setting which would make an investigation of the training dynamics under various hyperparameters prohibitively expensive. We therefore focus in our generalization study on the two other domains where attention based architectures have recently gained traction. Specifically, to see whether the the learning dynamics seen so far generalize, we perform two additional experiments: One investigating the architectures in a graph neural network (GNN) task and one investigating them in a reinforcement learning (RL) setup. Specifically, we train a transformer based GNN in the Protein-Protein-Interaction (PPI) graph node classification task (Zitnik & Leskovec, 2017) and investigate the different architectures as alterations on the working memory graph architecture for RL (Loynd et al., 2020), training on level 3 of the Baby-AI environment (Chevalier-Boisvert et al., 2019). The validation results under varying model dimension d are given in Figure 9. The results replicate several observations made so far. Specifically: (1) All attention based architectures can learn the task for some hyperparameter combinations. (2) There is a correlation between optimal learning rate and model dimension. (3) *BERT* is hyperparameter sensitive in comparison. See Appendices H and I for experiment details.

6 Conclusion

Taking all observations together, we come to the following conclusions: Many recent works apply some sort of self-attention mechanism involving a $\mathrm{softmax}$ that projects the attention weights to the probability simplex. In this work we question the $\mathrm{softmax}$ in dot-product self-attention modules. Our theoretical investigation shows that $\mathrm{softmax}$-attention outputs are constrained to the convex hull spanned by the value vectors. In our experiments we show that this can lead to an unwanted hyperparameter sensibility. We show that simpler architectures like max- and sum-pooling perform well when their architectural prior aligns with the underlying task. These architectures however fail in cases where the architectural prior is not suitable. As a solution, we propose to replace the $\mathrm{softmax}$ in attention through normalization. Our resulting normalized attention pooling (*NAP*) architecture is the only architecture of the 6 investigated that is robust in all tasks and setups, even if the data is heavily skewed towards easier sub-tasks. We hope that our work provides a stepping stone to examine architectures under varying biases in the data. Further, we see a lot of potential for future work to investigate and maybe exploit the correlated effects of hyperparameters.

REFERENCES

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

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Timo Bram, Gino Brunner, Oliver Richter, and Roger Wattenhofer. Attentive multi-task deep reinforcement learning. 07 2019.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Gino Brunner, Yang Liu, Damian Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. On identifiability in transformers. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJg1f6EFDB.

Satrajit Chatterjee. Coherent gradients: An approach to understanding generalization in gradient descent-based optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryeFY0EFwS.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJeXCo0cYX.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1xMH1BtvB.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

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. URL https://openreview.net/forum?id=YicbFdNTTy.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Kuan Fang, Alexander Toshev, Fei-Fei Li, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 538–547. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00063. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Fang_Scene_Memory_Transformer_for_Embodied_Agents_in_Long-Horizon_Tasks_CVPR_2019_paper.html.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Sarthak Jain and Byron C. Wallace. Attention is not explanation. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 3543–3556. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1357. URL https://doi.org/10.18653/v1/n19-1357.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 2019. URL http://proceedings.mlr.press/v97/lee19d.html.

Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HyxLRTVKPH.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Ricky Loynd, Roland Fernandez, Asli Celikyilmaz, Adith Swaminathan, and Matthew Hausknecht. Working memory graphs. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6404–6414. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/loynd20a.html.

Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 342–347. IEEE, 2011.

Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. *arXiv preprint arXiv:1910.06764*, 2019.

Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 68–80, 2019.

Damian Pascual, Gino Brunner, and Roger Wattenhofer. Telling bert's full story: from local attention to global aggregation. *arXiv preprint arXiv:2004.05916*, 2020.

Ofir Press, Noah A Smith, and Omer Levy. Improving transformer models by reordering their sublayers. *arXiv preprint arXiv:1911.03864*, 2019.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C Lipton. Learning to deceive with attention-based explanations. *arXiv preprint arXiv:1909.07913*, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 901, 2016.

Nimrod Segol and Yaron Lipman. On universal equivariant set networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxTwkrKDB.

Yi Tay, Anh Tuan Luu, Aston Zhang, Shuohang Wang, and Siu Cheung Hui. Compositional de-attention networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 6135–6145. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/16fc18d787294ad5171100e33d05d4e2-Paper.pdf.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *CoRR*, abs/2009.06732, 2020. URL https://arxiv.org/abs/2009.06732.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models, 2021.

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *CoRR*, abs/2105.01601, 2021. URL https://arxiv.org/abs/2105.01601.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768, 2020. URL https://arxiv.org/abs/2006.04768.

Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *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, Hong Kong, China, November 3-7, 2019*, pp. 11–20. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1002. URL https://doi.org/10.18653/v1/D19-1002.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkVhlh09tX.

Yuxin Wu and Kaiming He. Group normalization. *Int. J. Comput. Vis.*, 128(3):742–755, 2020. doi: 10.1007/s11263-019-01198-w. URL https://doi.org/10.1007/s11263-019-01198-w.

Zhanghao Wu*, Zhijian Liu*, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByeMPlHKPH.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 5754–5764, 2019.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 11960–11970, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 3391–3401, 2017. URL http://papers.nips.cc/paper/6931-deep-sets.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Han Zhang, Ian J. Goodfellow, Dimitris N. Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7354–7363. PMLR, 2019. URL http://proceedings.mlr.press/v97/zhang19d.html.

Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinform.*, 33(14):i190–i198, 2017. doi: 10.1093/bioinformatics/btx252. URL https://doi.org/10.1093/bioinformatics/btx252.

A REPRESENTATION POWER WITH RESPECT TO XOR

Lemma 1. *No convex combination can represent the binary exclusive OR (XOR) function defined on binary inputs $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ by the indicator function as $XOR(x_1, x_2) = \mathbf{1}_{x_1 \neq x_2}$.*

Proof. Suppose there exist convex combination weights a_1 and a_2 with $a_1 + a_2 = 1$, such that $a_1 \cdot x_1 + a_2 \cdot x_2$ represents the XOR function. Plugging in $x_1 = x_2 = 1$ yields $a_1 \cdot x_1 + a_2 \cdot x_2 = a_1 + a_2 = 1$, which gives the contradiction. □

Lemma 2. *Given the two binary inputs $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$, there exists an affine mapping $f : \{0, 1\}^2 \to \mathbb{R}^2$, such that the normalized weighting given by*

$$\frac{f_1(x_1, x_2) - \mu_{f(x_1,x_2)}}{\sigma_{f(x_1,x_2)}} \cdot x_1 + \frac{f_2(x_1, x_2) - \mu_{f(x_1,x_2)}}{\sigma_{f(x_1,x_2)}} \cdot x_2$$

is equivalent to the logical exclusive OR given by the indicator function as $XOR(x_1, x_2) = \mathbf{1}_{x_1 \neq x_2}$.

Proof. For a vector $\boldsymbol{l} \in \mathbb{R}^2$, the standard deviation $\sigma_{\boldsymbol{l}}$ can be simplified to

$$\begin{aligned}
\sigma_{\boldsymbol{l}} &= \sqrt{\frac{1}{2} \sum_{i \in \{1,2\}} (l_i - \mu_{\boldsymbol{l}})^2} \\
&= \sqrt{\frac{1}{2} \left(\left(l_1 - \frac{l_1 + l_2}{2} \right)^2 + \left(l_2 - \frac{l_1 + l_2}{2} \right)^2 \right)} \\
&= \frac{1}{2} |l_1 - l_2|
\end{aligned}$$

and the normalization function reduces to

$$\begin{aligned}
normalize(\boldsymbol{l}) &= \left[\frac{l_1 - \mu_{\boldsymbol{l}}}{\sigma_{\boldsymbol{l}}}, \frac{l_2 - \mu_{\boldsymbol{l}}}{\sigma_{\boldsymbol{l}}} \right]^T \\
&= \left[\frac{l_1 - l_2}{|l_1 - l_2|}, \frac{l_2 - l_1}{|l_1 - l_2|} \right]^T \\
&= \begin{cases} [1, -1]^T & if \quad l_1 > l_2 \\ [-1, 1]^T & if \quad l_1 < l_2 \\ undef. & if \quad l_1 = l_2 \end{cases}
\end{aligned}$$

As an example, consider the affine mapping $f(\boldsymbol{x}) = \boldsymbol{l} = [3x_1 + 1, 2x_2]^T$, which for $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ results in the function

$$\frac{3x_1 + 1 - 2x_2}{|3x_1 + 1 - 2x_2|} \cdot x_1 + \frac{-3x_1 - 1 + 2x_2}{|3x_1 + 1 - 2x_2|} \cdot x_2 = \mathbf{1}_{x_1 \neq x_2}$$

□

We note that for a realization of such an affine mapping across tokens given the weight sharing constraints of the discussed architectures we would need x_1 and x_2 to be distinguishable for the mapping to keys and queries, e.g., through positional embeddings. This however does not invalidate our conclusion that normalized weighting is more expressive than softmax weighting, as we do not require the inputs that are weighted to be distinguishable.

B VANISHING GRADIENT ANALYSIS

For better readability we will use subscripts here to index sequence elements and omit the implicit reference to the head and vector dimension used in the main text.

Lemma 3. *The partial derivative $g = \frac{\delta \mathrm{softmax}(\boldsymbol{l})_i}{\delta l_i}$ is limited to the range $g \in (0, 0.25]$.*

Proof.

$$g = \frac{\delta \text{softmax}(\boldsymbol{l})_i}{\delta l_i} = \frac{\exp(l_i)\sum_j \exp(l_j) - \exp(2 \cdot l_i)}{\left(\sum_j \exp(l_j)\right)^2} = a_i \cdot (1 - a_i) \text{ with } a_i = \text{softmax}(\boldsymbol{l})_i$$

Maximizing this function we get that the maximum of $g = 0.25$ is achieved for $a_i = 0.5$. Noting that $a_i > 0$, the claim follows. □

To expand this further, we introduce the notion of α-bandwidth.

Definition 1. *We call the range Δx in which the absolute value of a gradient factor $\frac{\delta f(x)}{\delta x}$ exceeds a threshold α, i.e. $\left|\frac{\delta f(x)}{\delta x}\right| > \alpha$, its α-bandwidth $BW_\alpha\left(\frac{\delta f(x)}{\delta x}\right)$.*

Please refer to Figure 11 for a visualization of this definition. Intuitively, the α-bandwidth gives a measure for the range of operation, where at least an α-fraction of the gradient gets backpropagated.

It also provides a measure of how fast a function saturates: Given x with $\frac{\delta f(x)}{\delta x} > \alpha$, updating x to $x' = x \pm BW_\alpha\left(\frac{\delta f(x)}{\delta x}\right)$ puts x' into the saturated region where $\frac{\delta f(x')}{\delta x'} \leq \alpha$. A function with a small α-bandwidth can therefore more easily saturate than a function with a larger α-bandwidth.

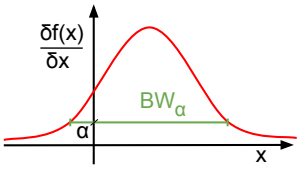

Figure 11: Visual definition

Lemma 4. *The α-bandwidth of $g = \frac{\delta \text{softmax}(\boldsymbol{l})_i}{\delta l_i}$ can be calculated as*

$$BW_\alpha(g) = \ln\left(1 + \frac{(2 - 4\alpha)\sqrt{1 - 4\alpha}}{4\alpha^2}\right)$$

Proof. Substituting $x = \exp(l_i)$ and $y = \sum_{j \neq i} \exp(l_j)$ we can write g as $g(x) = \frac{x}{x+y} \cdot \left(1 - \frac{x}{x+y}\right)$.

Solving $g(x) = \alpha$ yields $x = \frac{y}{2\alpha}\left(1 - 2\alpha \pm \sqrt{1 - 4\alpha}\right)$. Reversing the substitution of $l_i = \ln(x)$ and subtracting the larger from the smaller value, we get:

$$\begin{aligned} BW_\alpha(g) &= \ln\left(\frac{y}{2\alpha}\left(1 - 2\alpha + \sqrt{1 - 4\alpha}\right)\right) - \ln\left(\frac{y}{2\alpha}\left(1 - 2\alpha - \sqrt{1 - 4\alpha}\right)\right) \\ &= \ln\left(\frac{1 - 2\alpha + \sqrt{1 - 4\alpha}}{1 - 2\alpha - \sqrt{1 - 4\alpha}}\right) = \ln\left(1 + \frac{(2 - 4\alpha)\sqrt{1 - 4\alpha}}{4\alpha^2}\right) \end{aligned}$$

□

Corollary 1. *The range in which $g = \frac{\delta \text{softmax}(\boldsymbol{l})_i}{\delta l_i} > 0.01$ is smaller than 8.5.*

Proof. Calculating the α-bandwidth for $\alpha = 0.01$ we get $BW_{0.01}(g) \approx 8.477$. □

Note that $BW_\alpha(g)$ is independent of the length of the vector $\boldsymbol{l}$ as the remainder $y = \sum_{j \neq i} \exp(l_j)$ cancels. For the self-attention in the Transformer this translates to the fact that $BW_\alpha(g)$ is independent of the sequence length. This means that the Transformer architecture cannot adapt this range of operation according to the sequence length. Aggregated information from long sequences inevitably needs to be averaged or scaled to fall within this range.

We further note that the constant 0.01-bandwidth on its own is not the problem. In fact, both a sigmoid- and a tanh-non-linearity have a smaller 0.01-bandwidth. However, in the standard Transformer architecture, gradients are scaled dependent on the layer input. More specifically, dropping sequence indices to simplify notation, we have

Lemma 5. *For $\boldsymbol{k} = W_k\boldsymbol{x} + \boldsymbol{b}_k$, $\boldsymbol{q} = W_q\boldsymbol{x} + \boldsymbol{b}_q$, $W_q^T W_k + W_k^T W_q$ invertible and $||\boldsymbol{x}|| \geq c_1$ we can find constants $c_2 > 0$ and $c_3 > 0$ such that*

$$\left|\left|\frac{\delta l}{\delta \boldsymbol{x}}\right|\right| \geq c_2 \cdot ||\boldsymbol{x}|| - c_3$$

Proof. Writing out $l = \frac{<\boldsymbol{q},\boldsymbol{k}>}{\sqrt{d_h}}$ we get

$$l = \frac{1}{\sqrt{d_h}}(\boldsymbol{x}^T W_q^T W_k \boldsymbol{x} + (\boldsymbol{b}_k^T W_q + \boldsymbol{b}_q^T W_k)\boldsymbol{x} + \boldsymbol{b}_q^T \boldsymbol{b}_k)$$

Using $\frac{\delta \boldsymbol{x}^T A \boldsymbol{x}}{\delta \boldsymbol{x}} = \boldsymbol{x}^T(A + A^T)$ and substituting $\boldsymbol{g_b} = \boldsymbol{b}_k^T W_q + \boldsymbol{b}_q^T W_k$ we get

$$\frac{\delta l}{\delta \boldsymbol{x}} = \frac{1}{d_h}(\boldsymbol{x}^T(W_q^T W_k + W_k^T W_q) + \boldsymbol{g_b})$$

Setting $\boldsymbol{g_x} = \boldsymbol{x}^T(W_q^T W_k + W_k^T W_q)$ and solving for $\boldsymbol{x}$ we get $(W_q^T W_k + W_k^T W_q)^{-1}\boldsymbol{g_x}^T = \boldsymbol{x}$. With this, we can bound $||\boldsymbol{g_x}||$ with the operator norm on $(W_q^T W_k + W_k^T W_q)^{-1}$. Specifically, if we set c_2 as

$$c_2 = \frac{1}{\sqrt{d_h} \cdot ||(W_q^T W_k + W_k^T W_q)^{-1}||}$$

we get $\frac{1}{\sqrt{d_h}} \cdot ||\boldsymbol{g_x}|| \geq c_2 \cdot ||\boldsymbol{x}||$. We can then bound $\left|\left|\frac{\delta l}{\delta \boldsymbol{x}}\right|\right|$ as

$$\left|\left|\frac{\delta l}{\delta \boldsymbol{x}}\right|\right| = \frac{1}{\sqrt{d_h}}||\boldsymbol{g_x} + \boldsymbol{g_b}|| \geq \frac{1}{\sqrt{d_h}}\Big|||\boldsymbol{g_x}|| - ||\boldsymbol{g_b}||\Big|$$

Therefore, if we set $c_1 = \frac{1}{c_2 \cdot \sqrt{d_h}} \cdot ||\boldsymbol{g_b}||$ we get $||\boldsymbol{g_x}|| \geq c_2 \cdot \sqrt{d_h} \cdot ||\boldsymbol{x}|| \geq ||\boldsymbol{g_b}||$ and hence

$$\left|\left|\frac{\delta l}{\delta \boldsymbol{x}}\right|\right| \geq \frac{1}{\sqrt{d_h}}(||\boldsymbol{g_x}|| - ||\boldsymbol{g_b}||) \geq c_2 \cdot ||\boldsymbol{x}|| - c_3$$

for $c_3 = \frac{1}{\sqrt{d_h}} \cdot ||\boldsymbol{g_b}||$. □

Lemma 5 shows that the magnitude of the gradient factor scales with the magnitude of the input. Therefore, a single large input can yield a correspondingly large parameter update. This in turn can put the $\mathrm{softmax}$ easily into the saturated region, since the $\mathrm{softmax}$ only has a constant α-bandwidth. Once saturated, the $\mathrm{softmax}$ leads to vanishing gradients henceforth.

C Sequence Length Dependent Focus

For Figure 2 we sample 16'384 value, key and query vectors of dimension $d_h = 128$ per sequence length $N \in \{1, 2, 4, 8, 16, 32, 64, 128, 512, 1024, 2048\}$ from a normal Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_{d_h})$ - $\boldsymbol{I}_{d_h}$ being the d_h-dimensional identity matrix. We split the samples to form the sequences and calculate the corresponding output vectors $\boldsymbol{o}^i$ for $i \in \{1, \ldots, N\}$. Here, the softmax attention outputs are calculated as described in Section 2, while the mean-, sum- and max-outputs are calculated as mean-, sum- and max-reduce of the value vectors over the sequence dimension. For the normalized results we take the sum-output vectors and normalize them (over the d_h-dimensional vector dimension). Note that such a normalization can be applied to any of the aggregation methods to get qualitatively similar results. The plots in Figure 2 are generated by reporting the standard deviation over all output values and the mean norm of the output values, respectively.

Given the numerous successes of Transformers in natural language processing, we conjecture that a bias towards local information might be beneficial in language modeling. However, the implicit dependence on sequence length in a model that should be oblivious to different input sequence lengths is questionable. We leave an in depth investigation to future work.

D Architectures

We provide a schematic of 1 Transformer-layer of each architecture investigated in Figure 12. Our base architectures consist of 2 such layers followed by a projection to the output dependent on the task as described in the corresponding sections (cf. Section 5.1, 5.2 and 5.3).

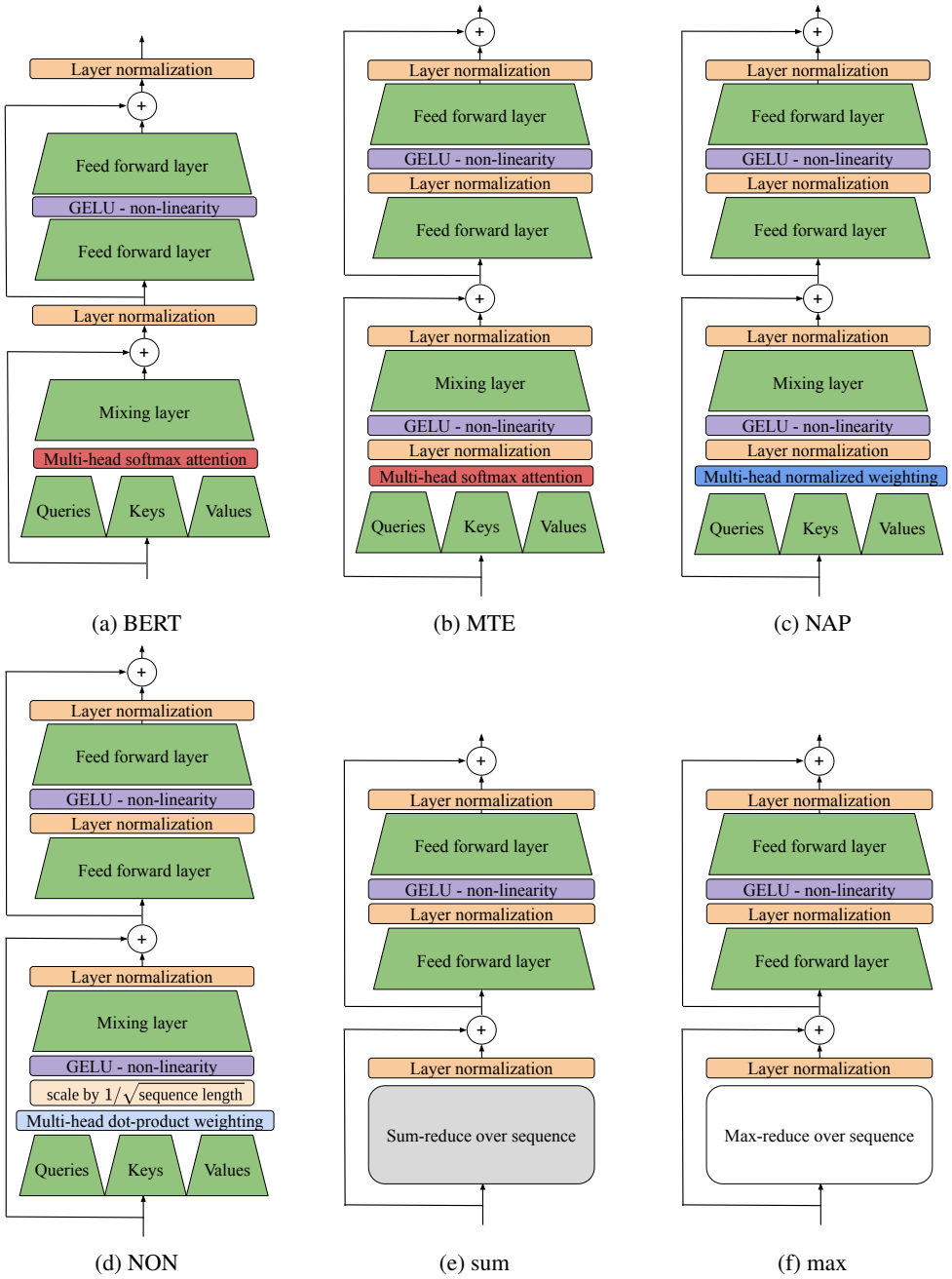

Figure 12: Schematics of 1 Transformer-layer block of the different architectures investigated. Green layers correspond to the main weight matrices that are trained. Note that displayed dimensions are not to scale - the hidden dimension of the feed forward layer is larger than the model dimension and the hidden layer size in the feed forward network of "max" and "sum" are adjusted to approximately match the parameter count of the other architectures.

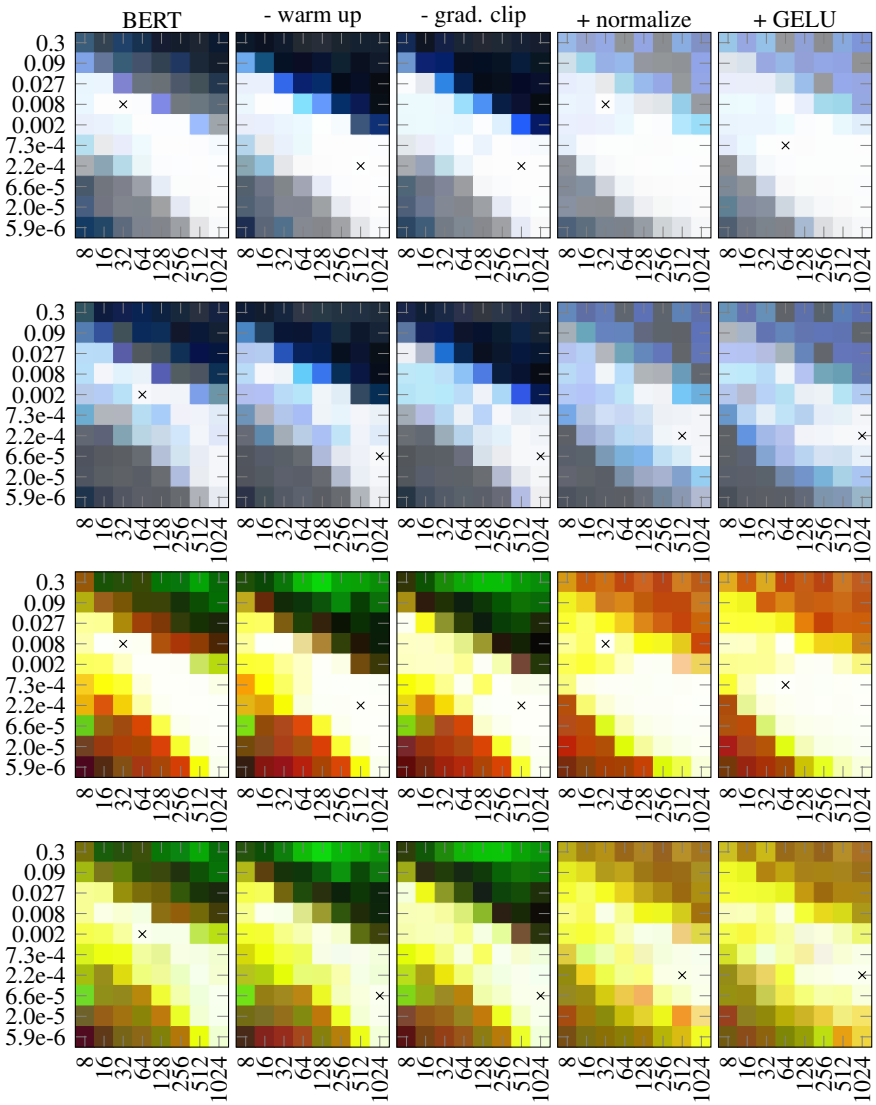

Figure 13: Learning rate (y-axis) vs. model dimension d (x-axis) on the argmin-first-argmax case distinction task (with output across all tokens) - architecture modification ablation study. In the first two rows, RGB pixel values correspond to min-, mean- and max-accuracy. In the last two rows, RGB pixel values correspondto *argmin*-, *first*- and *argmax*-mean-case-accuracies. **1. row:** Training accuracy (sequence length $N = 128$). **2. row:** Validation accuracy when validating on sequences of half the length ($N = 64$). **3. row:** Training case accuracy (sequence length $N = 128$). **4. row:** Validation case accuracy when validating on sequences of half the length ($N = 64$). Crosses indicate the combination for best mean accuracy, which are reported in Table 1.

D.1 Architecture Modification Ablations

An empirical ablation of the modifications that lead from the *BERT* architecture to the *MTE* architecture is given in Figure 13. The plots are generated as described in Sections 5.1.1 and 5.1.2.

The first column in Figure 13 corresponds to the original *BERT* architecture, trained with gradient norm clipping and learning rate warm up.

The second column (- warm up) corresponds to the same architecture, but trained without learning rate warm up. Here we see that too high learning rates learn even less without learning rate

Table 1: Ablation study accuracy values taken from the hyper-parameter combination that led to the best mean overall accuracy, indicated by a cross in Figure 13.

		BERT	- warm up	- grad. clip	+ normalize	+ GELU
Overall	min	99.3%	99.4%	99.3%	99.2%	99.3%
Training	mean	99.4%	99.5%	99.4%	99.3%	99.3%
Accuracy	max	99.5%	99.6%	99.6%	99.5%	99.5%
Overall	min	96.5%	96.9%	96.9%	96.3%	96.7%
Validation	mean	97.2%	97.6%	97.2%	96.8%	97.3%
Accuracy	max	98.2%	98.2%	98.2%	98.4%	98.2%
Mean Case	argmin	99.3%	99.6%	99.5%	99.6%	99.5%
Accuracy	first	100%	100%	100%	100%	100%
Training	argmax	96.9%	98.1%	97.5%	96.5%	98.0%
Mean Case	argmin	98.0%	98.0%	98.0%	98.1%	97.9%
Accuracy	first	100%	100%	100%	100%	100%
Validation	argmax	93.9%	93.8%	93.1%	91.2%	93.8%

warm up in the *BERT* architecture, hinting at a necessity for learning rate warm up for the original architecture.

The third column (- grad. clip) reports the results if we further remove gradient clipping from the training schedule. This does not seem to have a big impact in our setup.

Next, we report in the forth column (+ normalize) the results of moving the layer normalization before the residual addition and introducing an additional layer normalization right after the attention mechanism as well as on the hidden layer of the feed forward network. Note that this change removes the bias towards local information discussed in the end of Section 3. We see that this change leads to a profound shift in focus in regions where the learning rate is high: models with the original normalization focus on the (local) *first*-case, while models with our normalization focus on the (majority) *argmin*-case. This is in line with the insights stated in Section 5.1.2.

Finally, we report in the fifth column (+ GELU) the results of adding an additional GELU layer after the attention. These results correspond to the *MTE* architecture used throughout the paper.

Apart from the performance landscape changes just mentioned, the best hyper-parameter accuracies remain similar throughout all modifications, cf. Table 1.

D.2 Regularization Experiments

To limit the number of variables which are not accounted for in the experiments, we focus on the *infinite data but limited training time* regime. In this regime, every batch consists of new data points. We believe that this regime is of paramount interest in future research, as more devices create a constant stream of data and training is more limited by the available training time than the available data. This regime allows us to omit regularization in all architectures as over-fitting is not an issue. In fact, our supplementary experiments below as well as related work (Lan et al., 2020) show that regularization does not help in this regime. We leave a comparison of the architectures in the limited data regime to future work.

Here, we show empirical results supporting the intuition that $L2$ as well as *dropout* regularization does not help in our setup. For each of our tasks, we take our default hyper-parameters ($d = 128$, $L = 2$, $M = 4$, $N = 128$) and train 5 random seeds per learning rate for models with regularization, varying the dropout rate in $\{0.0625, 0.125, 0.25, 0.5\}$ and the $L2$ regularization weighting in $\{0.0001, 0.001, 0.01, 0.1\}$. Tables 2, 3 and 4 report the best mean accuracy achieved with the small number behind the accuracies indicating the regularization used, 1 referring to the smallest, 4 to the largest. We underline the results where regularization did lead to an improvement in mean accuracy. Note however that these improvements should be taken with a grain of salt, as (1) none of these improvements is significant considering the performance variation across random seeds and (2) the regularized values are likely to be overestimated, as the max is taken over 40 averages (4 regularization values times 10 learning rates) as compared to 10 averages (10 learning rates) in the unregulated case.

Table 2: Regularization results in the case distinction task with output taken across all tokens. The top three rows correspond to the best mean training accuracy, while the bottom three rows correspond to the best mean validation accuracy when validating on sequences of half the length.

	BERT	MTE	NAP	NON	sum	max
unregularized	99.3%	99.1%	99.3%	99.1%	98.9%	99.2%
with dropout	$98.1\%^{1}$	$97.3\%^{1}$	$97.8\%^{1}$	$97.5\%^{1}$	$97.3\%^{1}$	$98.2\%^{1}$
with $L2$-regularization	$99.3\%^{2}$	$99.2\%^{1}$	$99.2\%^{2}$	$99.2\%^{1}$	$98.9\%^{1}$	$99.4\%^{2}$
unregularized	95.5%	95.5%	97.0%	95.3%	75.0%	97.1%
with dropout	$94.4\%^{1}$	$94.6\%^{1}$	$96.8\%^{2}$	$96.0\%^{1}$	$83.1\%^{1}$	$96.3\%^{1}$
with $L2$-regularization	$97.2\%^{2}$	$93.6\%^{2}$	$97.1\%^{1}$	$96.1\%^{2}$	$67.7\%^{2}$	$97.2\%^{2}$

Table 3: Regularization results in the case distinction task with output from the first token. The top three rows correspond to the best mean training accuracy, while the bottom three rows correspond to the best mean validation accuracy when validating on sequences of half the length.

	BERT	MTE	NAP	NON	sum	max
unregularized	36.6%	66.5%	94.5%	23.2%	22.8%	97.8%
with dropout	$44.9\%^{1}$	$44.3\%^{1}$	$85.0\%^{1}$	$23.2\%^{1}$	$22.6\%^{1}$	$92.6\%^{1}$
with $L2$-regularization	$36.0\%^{2}$	$55.3\%^{1}$	$93.8\%^{2}$	$22.8\%^{1}$	$22.8\%^{1}$	$95.4\%^{1}$
unregularized	36.7%	50.6%	83.9%	29.6%	28.5%	88.5%
with dropout	$41.4\%^{2}$	$40.7\%^{1}$	$74.6\%^{1}$	$29.6\%^{3}$	$28.9\%^{4}$	$87.8\%^{1}$
with $L2$-regularization	$37.2\%^{2}$	$45.7\%^{1}$	$82.5\%^{2}$	$28.9\%^{1}$	$29.0\%^{1}$	$81.0\%^{1}$

Table 4: Regularization results in the mode finding task. The top three rows correspond to the best mean training accuracy, while the bottom rows correspond to the best mean validation accuracy when validating on sequences of twice the length.

	BERT	MTE	NAP	NON	sum	max
unregularized	99.6%	99.8%	99.6%	98.7%	99.8%	14.4%
with dropout	$93.9\%^{1}$	$93.3\%^{1}$	$94.3\%^{1}$	$91.8\%^{1}$	$93.3\%^{1}$	$24.5\%^{1}$
with $L2$-regularization	$99.5\%^{1}$	$99.9\%^{2}$	$99.7\%^{1}$	$98.8\%^{1}$	$99.9\%^{4}$	$14.4\%^{2}$
unregularized	95.3%	95.4%	94.9%	91.3%	95.8%	13.5%
with dropout	$94.8\%^{2}$	$95.4\%^{2}$	$93.8\%^{1}$	$92.6\%^{1}$	$95.7\%^{2}$	$13.4\%^{4}$
with $L2$-regularization	$94.7\%^{1}$	$96.0\%^{1}$	$94.9\%^{1}$	$94.7\%^{2}$	$95.8\%^{1}$	$13.7\%^{1}$

Overall we note that none of the architectures consistently benefits from regularization in our setup and regularization often decreases mean performance. Further, we point out that the best performance with regularization is most of the times achieved with the smallest regularization.

E CASE LEARNING CURVES

Figures 14, 15 and 16 show the case accuracies over the course of training. The corresponding color plot is given in Figure 17. Besides the observations made in the main text, a few additional insights can be noted: (1) Cases are mostly learned in the order of their occurrences (recall that 72.37% of the examples are from the *argmin* case, 20.09% are from the *first* case and 7.53% are from the *argmax* case). This is to be expected when training with gradient descent, cf. Chatterjee (2020). (2) This order is not always given in the *BERT* architecture. Besides the focus on the *first* case if the learning rate is too high - discussed in the main text - we also highlight a curiosity that occurs when the model dimension is too small (see plot highlighted in with red in Figure 14): The *first* case is learned and then unlearned in favor of the *argmin* case. Note that all 5 random seeds follow this pattern. Note also that for a different learning rate, the opposite holds as seen in the plot just below the highlighted plot. We highly encourage an interested reader to check out our code release,[3] which includes these results as well as visualization scripts to inspect them further.

[3]See supplementary material.

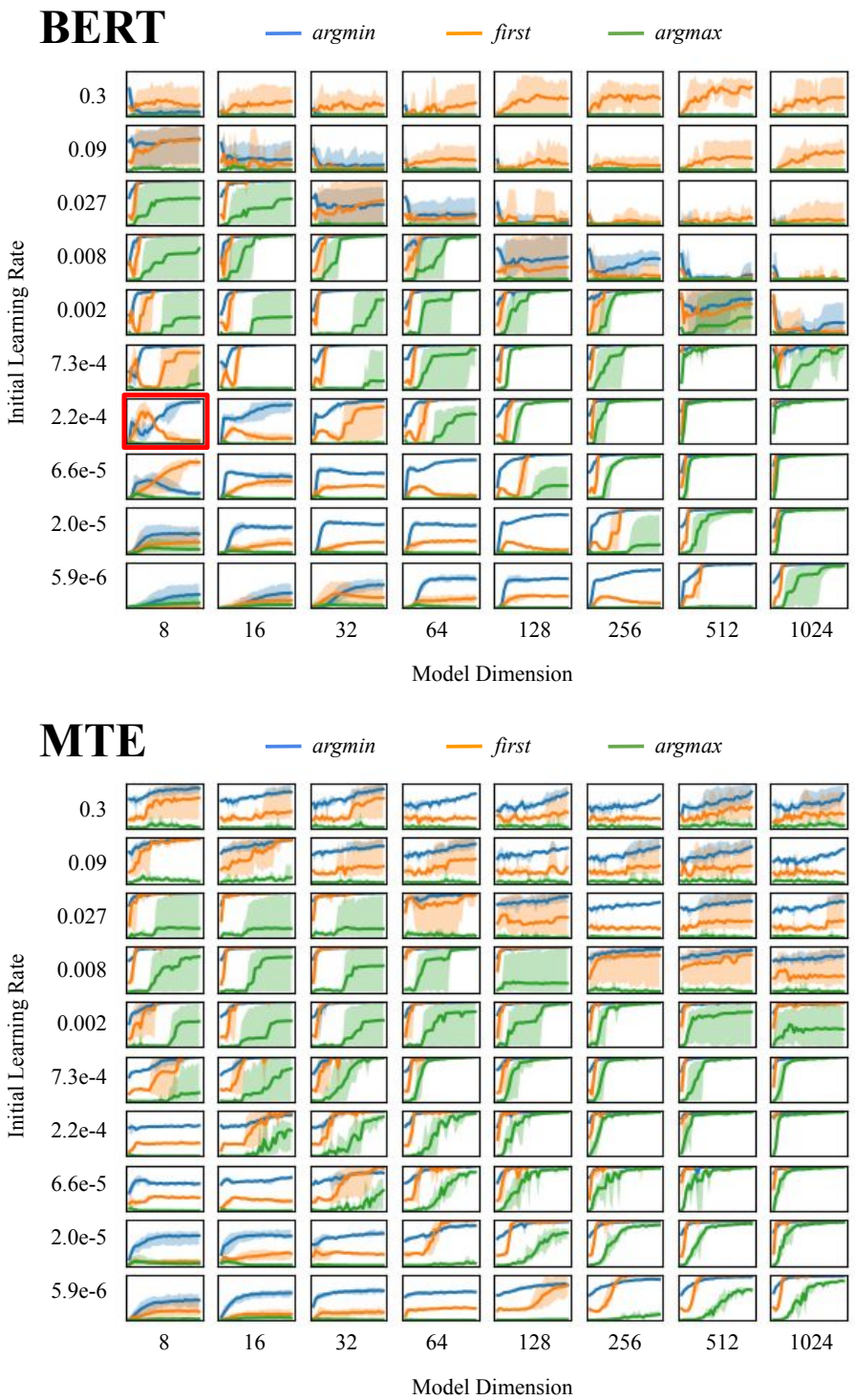

Figure 14: Case accuracies over the course of training on the *argmin-first-argmax* case distinction task with output across all tokens, cf. Section 5.1. Each small sub-plot shows the case accuracies (y-axis, bottom is set to 0%, top to 100%) over the course of training (x-axis). Solid lines represent the mean accuracy over the 5 random seeds while shaded areas fill the spread between min- and max-accuracy achieved. Models *BERT* and *MTE* are shown here, cf. Figures 15 and 16.

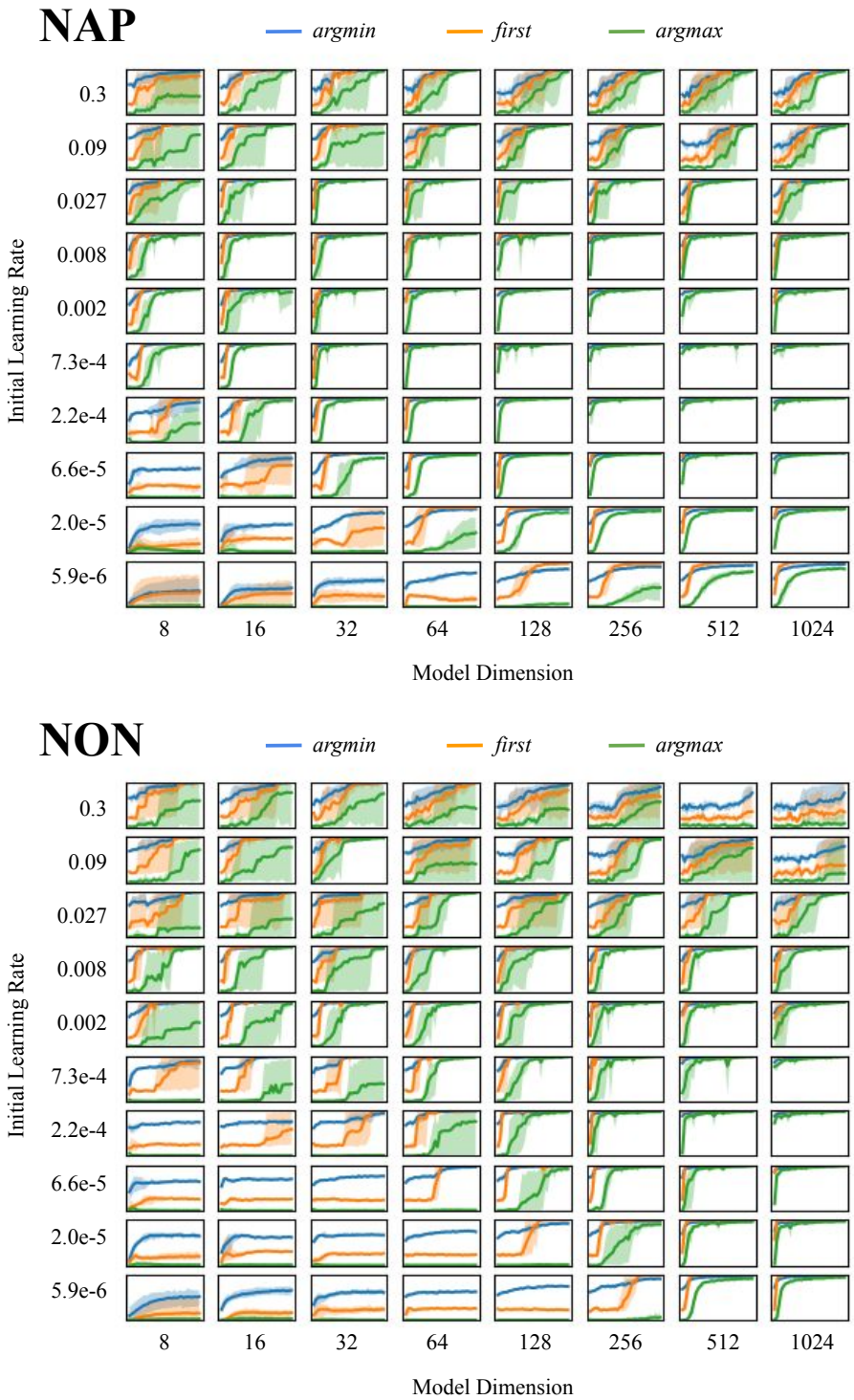

Figure 15: Case accuracies over the course of training on the *argmin-first-argmax* case distinction task with output across all tokens, cf. Section 5.1. Each small sub-plot shows the case accuracies (y-axis, bottom is set to 0%, top to 100%) over the course of training (x-axis). Solid lines represent the mean accuracy over the 5 random seeds while shaded areas fill the spread between min- and max-accuracy achieved. Models *NAP* and *NON* are shown here, cf. Figures 14 and 16.

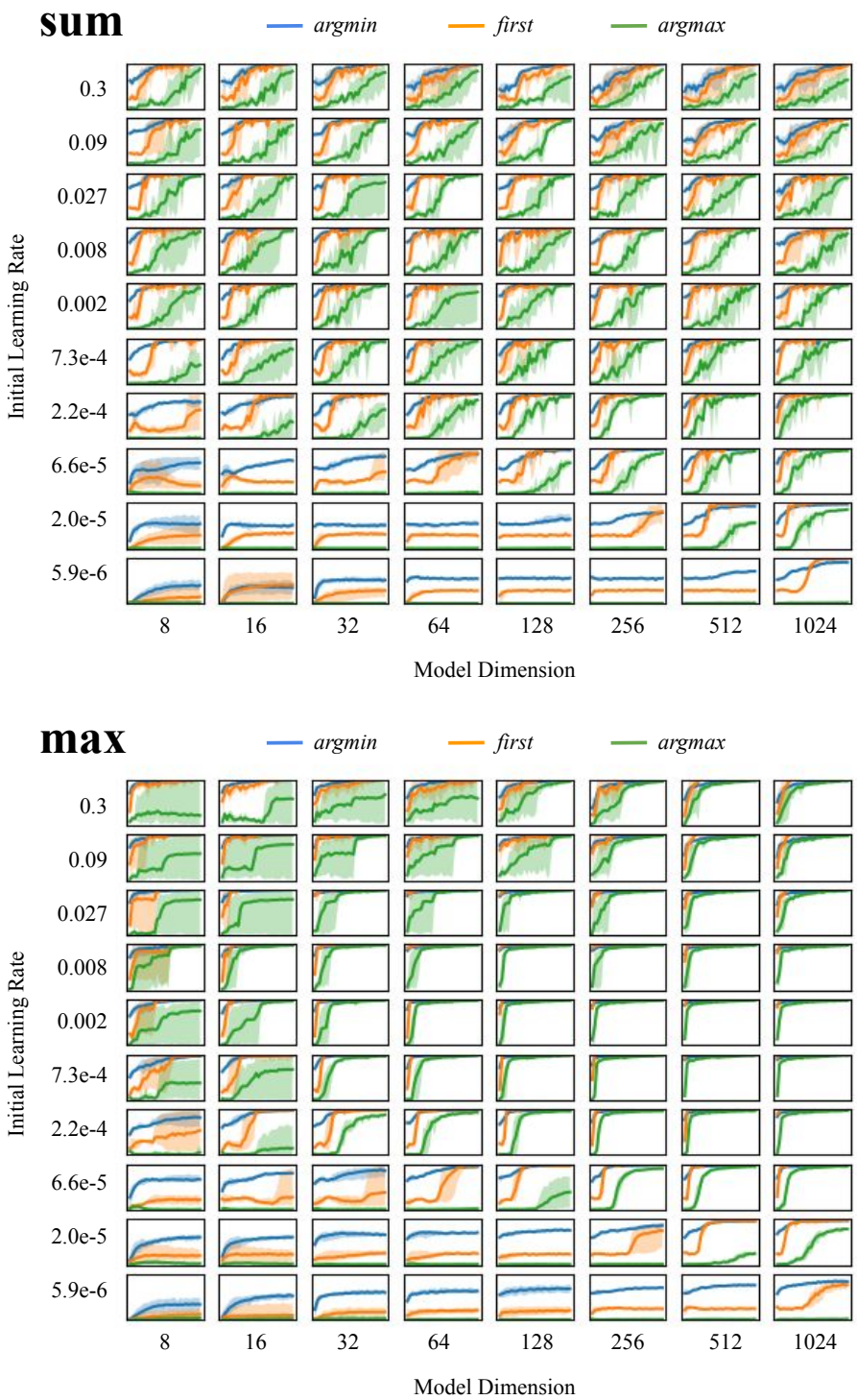

Figure 16: Case accuracies over the course of training on the *argmin-first-argmax* case distinction task with output across all tokens, cf. Section 5.1. Each small sub-plot shows the case accuracies (y-axis, bottom is set to 0%, top to 100%) over the course of training (x-axis). Solid lines represent the mean accuracy over the 5 random seeds while shaded areas fill the spread between min- and max-accuracy achieved. Models *sum* and *max* are shown here, cf. Figures 14 and 15.

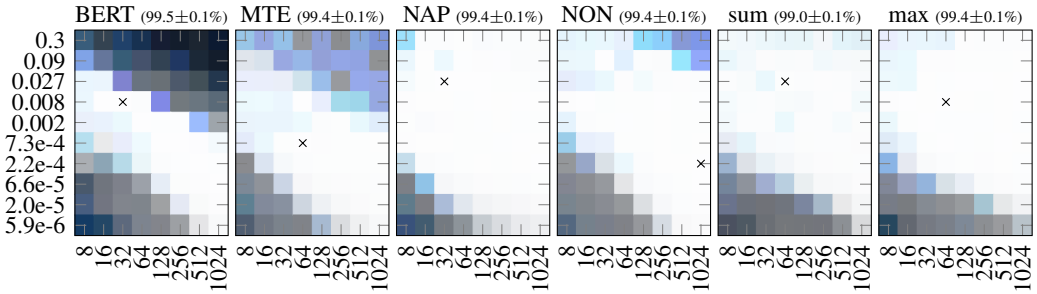

Figure 17: Learning rate (y-axis) vs. model dimension d (x-axis) on the argmin-first-argmax case distinction task (with output across all tokens). The pixels' R (red), G (green) and B (blue) values correspond to min-, mean- and max-accuracy, respectively, of the corresponding hyperparameter combination. The plot shows the accuracy when evaluating on sequences of length $N = 128$. Crosses indicate the combination for best mean accuracy, reported behind the model name.

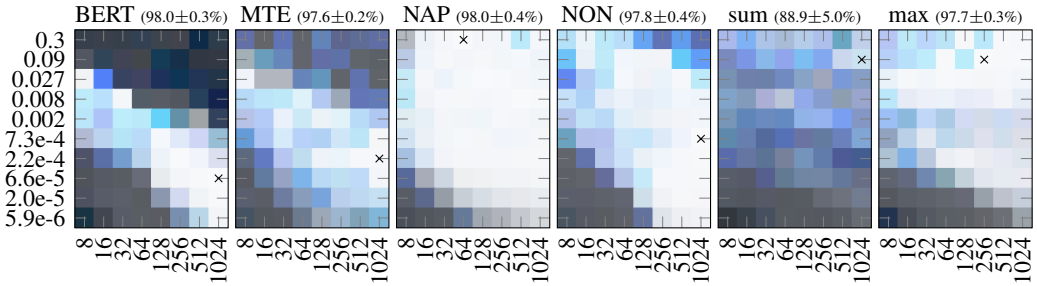

Figure 18: Rerun of the experiments of Figure 5 with $L = 4$ instead of $L = 2$. Crosses indicate the combination for best validation accuracy, which we report behind the model name.

F Case Distinction Task - Additional Results

Figure 17 shows the accuracy in the case distinction task when evaluating with the same sequence length as used during training, i.e., $N = 128$.

F.1 Varying Model Dimension with 4 and 6 Layers

Figures 18 and 19 show the results when varying the model dimension for deeper models, specifically 4 and 6 Transformer-layer deep models. The results do not differ which is why we choose to set the default to $L = 2$ layers, which in turn allows us to run more experiments. Note that we also present an experiment in F.6 where we explicitly vary the depth up to $L = 64$ Transformer-layers.

F.2 Varying Batch Size

In Figure 20 we provide the case accuracy results of an additional experiment, varying the batch size. In this experiment we train the models using different batch sizes, adjusting the number of training steps accordingly to keep the total number of training points seen constant. With this experiment we aim to show the training behaviour of the different architectures if we go from single example batches (many, potentially noisier updates) to batches of size 128 - a batch size in which each batch contains in expectation several examples per case, but fewer updates are made to the network parameters. Besides replicating several insights made in the main text, this experiment shows: (1) smaller batches require a smaller learning rate, supporting our argument that hyper-parameters should not be optimized independent of each other. (2) The focus of *BERT* on the *first*-case when the learning rate is too high is amplified in smaller batches.

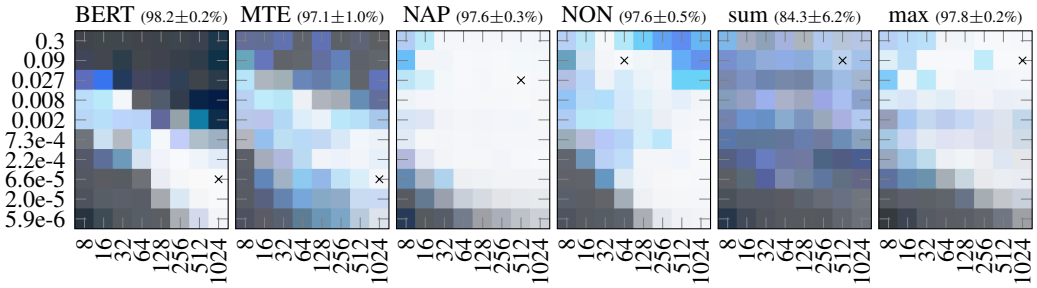

Figure 19: Rerun of the experiments of Figure 5 with $L = 6$ instead of $L = 2$. Crosses indicate the combination for best validation accuracy, which we report behind the model name.

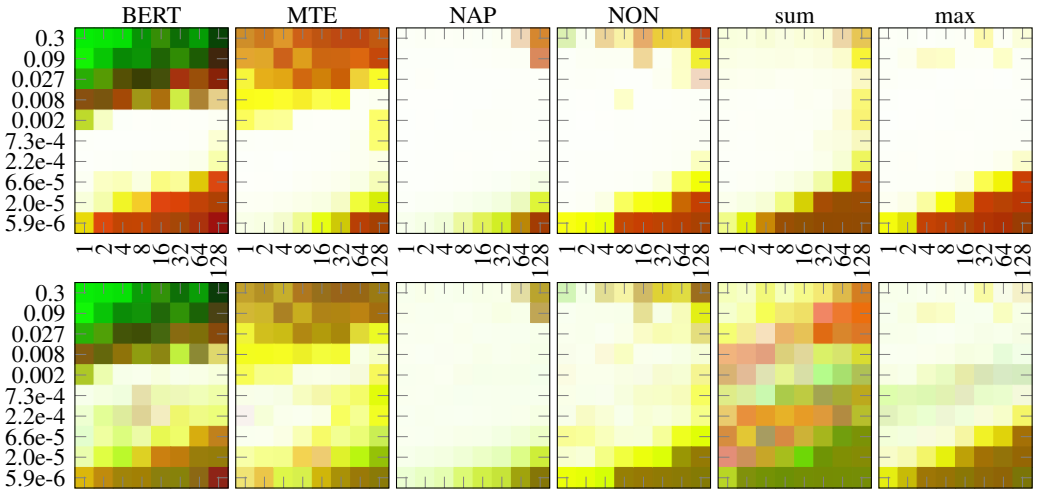

Figure 20: Learning rate (y-axis) vs. batch size (x-axis) on the argmin-first-argmax case distinction task (with output across all tokens). RGB pixel values correspond to *argmin*-, *first*- and *argmax*-case-accuracies, respectively. **Top row:** Training accuracy (sequence length $N = 128$). **Bottom row:** Validation accuracy on sequences of length $N = 64$.

F.3 Varying Initialization Scale

In Figure 21 we provide the case accuracy results of an additional experiment, varying the initialization standard deviation of the truncated normal distribution with which we initialize the weight matrices. The results show additionally to the observations made previously that the $\mathrm{softmax}$ based models *BERT* and *MTE* struggle to learn if the initialization is too large. This is likely due to the fact that the $\mathrm{softmax}$ is already saturated if the initialization too large. Also, larger initializations seem to require a larger learning rate.

F.4 Local vs. Global Focus under Varying Data Bias

To investigate the local vs. global focus further, we conduct another experiment under varying data bias. Specifically we look at a task where each token has to output its own position (*identity*-case), if there is a 64 in the input sequence. If there is no 64, each token has to output the *argmin* position. The results when varying the percentage of *identity*-case data points are given in Figure 22. Red represents that only the *identity*-case is learned while turquoise represents that only the *argmin*-case is learned. We can see that *BERT* struggles to learn the *argmin*-case (which requires global information) as soon as it is not the majority case anymore.

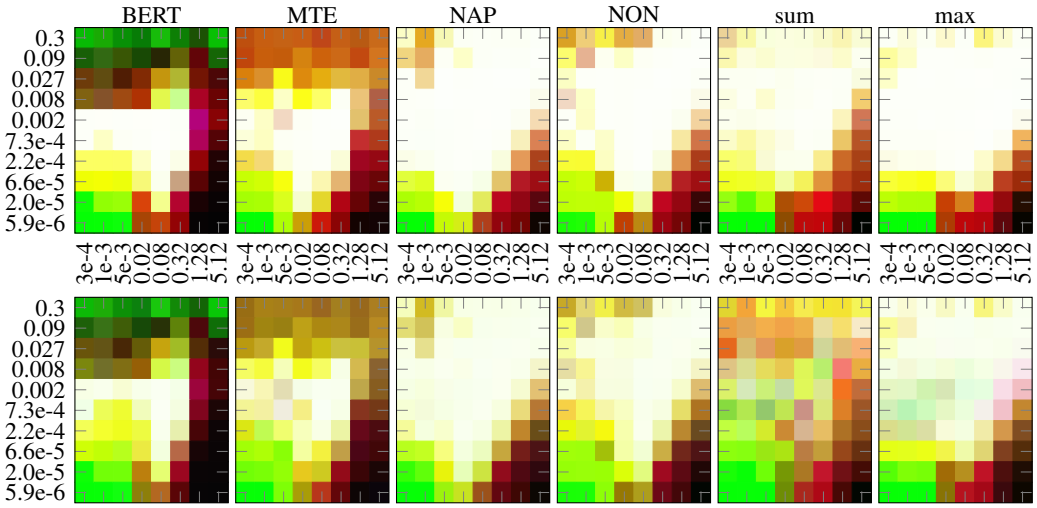

Figure 21: Learning rate (y-axis) vs. initialization scale (x-axis) on the case distinction task (output across tokens). RGB pixel values correspond to the case-accuracies. **Top row:** Training accuracy ($N = 128$). **Bottom row:** Validation ($N = 64$).

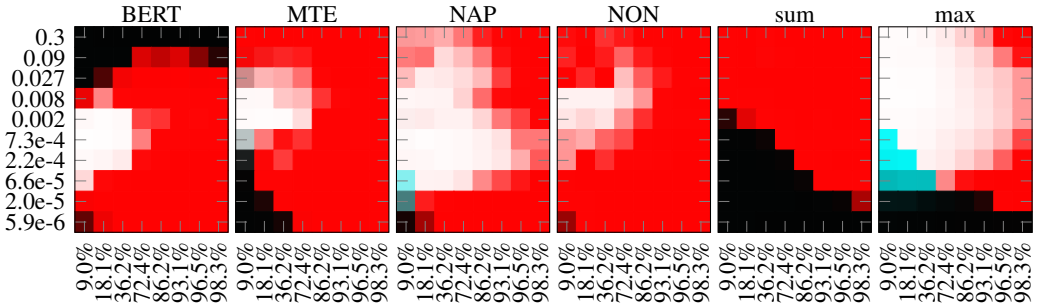

Figure 22: Learning rate (y-axis) vs. percentage of *identity*-case (x-axis). R = *identity* accuracy, G and B = *argmin* accuracy.

F.5 First Token Output - Varying Model Dimension

Section 5.2 discusses the case accuracies when training on the case distinction task with outputs taken from the first token. In Figure 23 we addtionally give best the min-, mean- and max-accuracies over the course of training. The top row corresponds to in-distribution/training accuracy ($N = 128$) while the bottom row corresponds to out-of-distribution generalization accrucay when validating on sequences of half the length ($N = 64$). Again we note a correlation between optimal learning rate and model dimension, especially in the *BERT* and *MTE* architecture. We also note that *BERT* and *MTE* have a large performance variation across random seeds in this setup.

F.6 First Token output - Varying Depth

In this section we investigate whether our results are tied to the shallow architecture of $L = 2$ Transformer layers. We therefore vary the number of Transformer-layers L and report the results on the case distinction task with outputs taken from the first token in Figure 24. The results lead us to the following observations: (1) The *BERT* architecture does seem to perform better when the number of Transformer-layers is increased to $L = 4$. However, the performance degrades if we further increase the depth. (2) The *NAP* architecture achieves a higher best mean accuracy and performs well on a wide range of depths. (3) The *max* architecture performs well on the biggest range of hyperparameters. This is due to the beneficial architectural prior as discussed earlier.

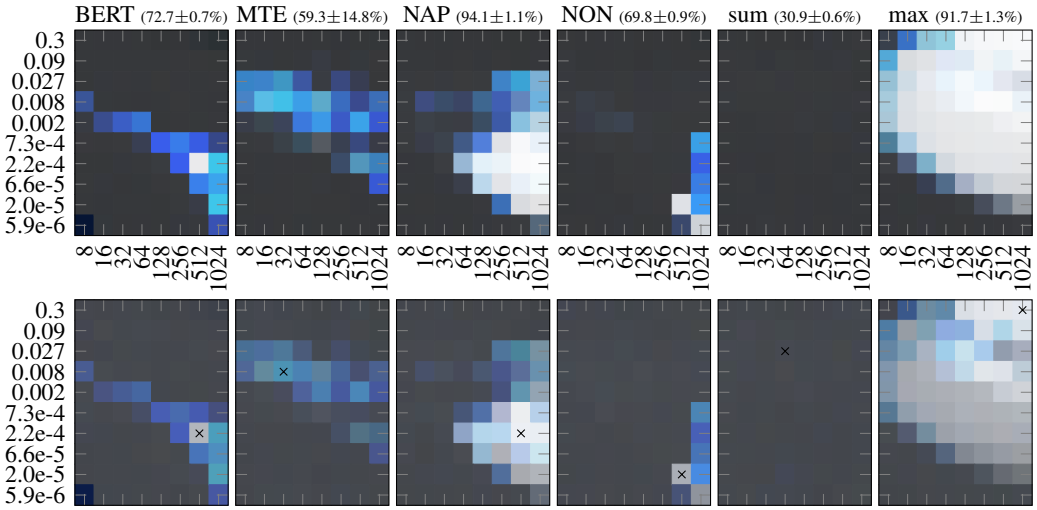

Figure 23: Learning rate (y-axis) vs. model dimension d (x-axis) on the case distinction task with output from the first token. RGB pixel values correspond to min, mean and max accuracy. **Top row:** Training accuracy (sequence length $N = 128$). **Bottom row:** Validation accuracy when validating on sequences of half the length ($N = 64$). Crosses indicate the combination for best validation accuracy, which we report with standard deviation behind the model name.

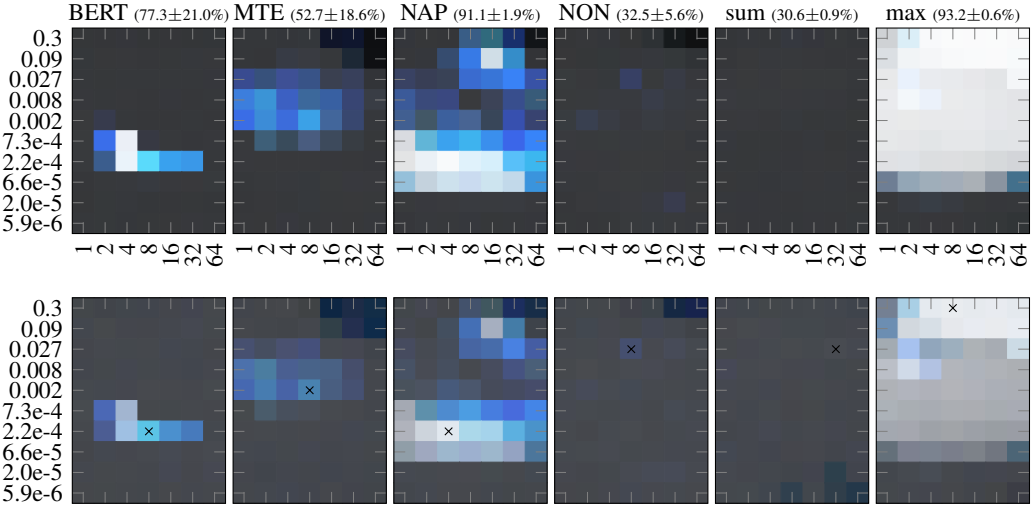

Figure 24: Learning rate (y-axis) vs. Transformer-layers L (x-axis) on the case distinction task (output from the first token). RGB pixel values correspond to min, mean and max accuracy. **Top row:** Training accuracy (sequence length $N = 128$). **Bottom row:** Validation accuracy when validating on sequences of half the length ($N = 64$). Crosses indicate the combination for best validation accuracy, which we report with standard deviation behind the model name.

G Mode Finding Task - Additional Results

G.1 Varying Vocabulary Size

Figure 25 shows the results of an additional experiment, varying the vocabulary size S while keeping the sequence length $N = 128$ constant during training. For this experiment, we also vary the total number of training steps and set it to $400 \cdot S$, to keep the number of examples seen per vocabulary token approximately constant. We also include zero-shot generalization results when testing on

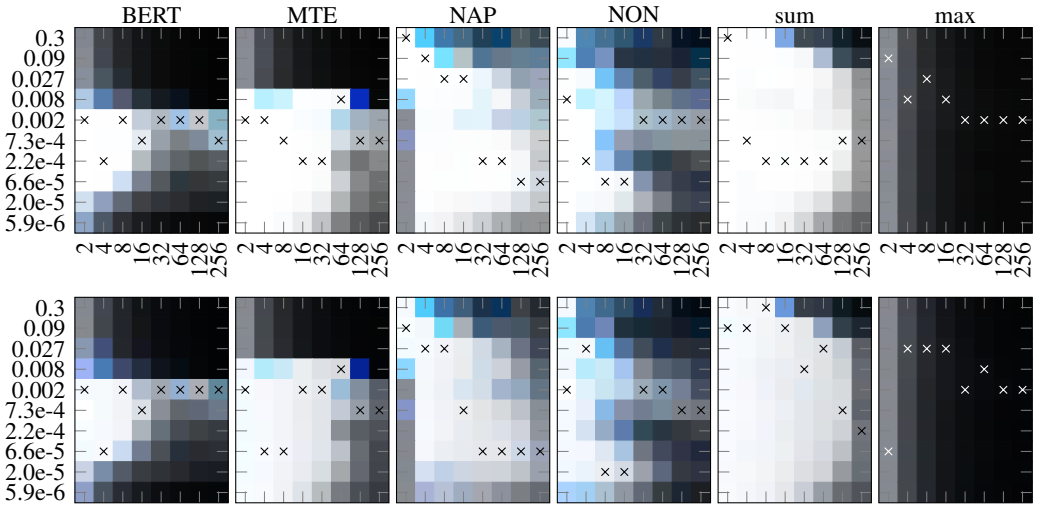

Figure 25: Learning rate (y-axis) vs. vocabulary size S (x-axis) on the mode finding task. RGB pixel values correspond to min, mean and max accuracy. **Top row:** Training accuracy (sequence length $N = 128$). **Bottom row:** Validation accuracy when validating on sequences of twice the length ($N = 256$). Crosses indicate the learning rate for best mean accuracy, reported in Table 5.

Table 5: Best mean accuracy per vocabulary size, taken from the combinations indicated in Figure 25. First six rows correspond to training accuracies, bottom six rows correspond to validation accuracies. Bold numbers indicate a min-accuracy higher than the best max accuracy of all other models.

	$S = 2$	$S = 4$	$S = 8$	$S = 16$	$S = 32$	$S = 64$	$S = 128$	$S = 256$
BERT	100%	99.9%	99.9 %	92.1%	72.5%	76.2%	77.4%	74.4%
MTE	100%	100%	99.9%	99.8%	99.3%	97.3%	73.3%	64.9%
NAP	100%	99.9%	99.8%	99.6%	99.7%	**99.6%**	97.4%	**84.6%**
NON	100%	99.9%	99.2%	97.3%	74.7%	71.5%	65.2%	61.5%
sum	100%	100%	99.9%	99.8%	99.7%	99.2%	97.5%	60.6%
max	55.7%	30.1%	17.3%	10.4%	6.6%	4.6%	3.6%	3.1%
BERT	100%	98.2%	95.8 %	88.0%	65.7%	68.6%	68.0%	53.0%
MTE	99.2%	98.4%	96.1%	93.6%	90.5%	85.4%	61.6%	38.9%
NAP	99.6%	98.4%	95.8%	93.1%	90.6%	90.4%	84.3%	**64.3%**
NON	100%	97.7%	93.1%	85.7%	66.4%	58.3%	50.2%	46.4%
sum	99.0%	97.9%	96.7%	94.4%	91.6%	89.1%	85.9%	45.6%
max	53.8%	29.3%	16.1%	9.5%	6.0%	4.2%	3.0%	2.1%

sequences of twice the length ($N = 256$). Compared to the case distinction task we can do such a generalization evaluation here as we do not learn any positional embeddings in this setup. We make the following observations: (1) *max* completely fails to learn in any of the vocabulary sizes. Note that the shading to the left merely corresponds to the majority class base rate. (2) *NAP* struggles when the vocabulary consists of only 2 tokens. This is expected, as the mean subtraction in the normalization effectively removes the task relevant information (the mode) in this case. Note however, that for a high enough learning rate, the model learns to use the bias parameter b introduced in the normalization - effectively reverting to sum pooling. (3) While all models learn the task well on small vocabularies, *NAP* outperforms all other approaches significantly when S gets larger then the training sequence length, cf. Table 5.

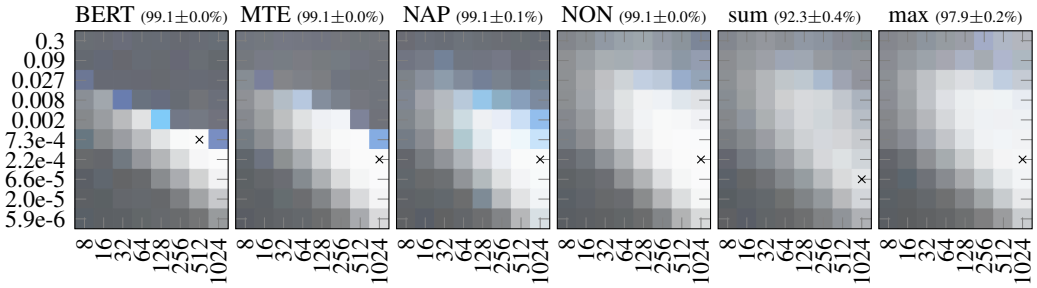

Figure 26: Full results of the GNN experiments on the protein-protein-interaction task. The plot shows a variation of learning rate (y-axis) vs. model dimension d (x-axis). RGB pixel values correspond to min, mean and max validation node classification F1-score. Crosses indicate the best combination (validation), from which we report the test performance behind the model name.

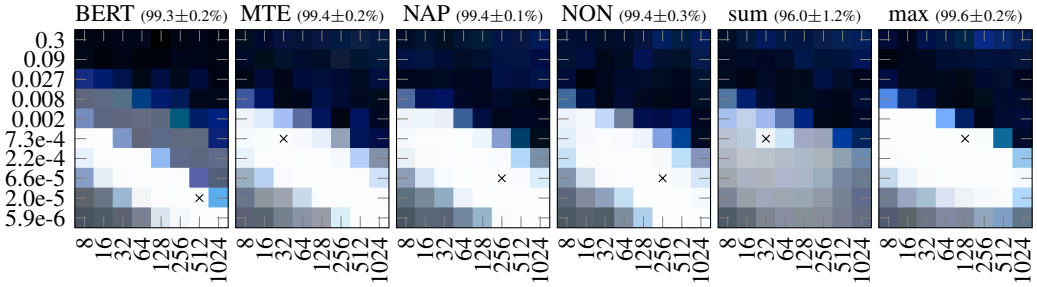

Figure 27: Full results of the RL experiments. The plot shows a variation of learning rate (y-axis) vs. model dimension d (x-axis). RGB pixel values correspond to min, mean and max success rate when evaluating on 10,000 newly generated games. Crosses indicate the combination for best success rate, which we report with standard deviation behind the model name. Note here that training is stopped early if a success rate of 99% is reached.

H PPI Experiment

For the PPI node classification task we train a GNN based on our architectures with aggregation over neighbors instead of the whole sequence. Here we use $L = 3$ transformer layers, as the number of layers specifies the k-hop neighborhood from which information is aggregated in a GNN ($k = L$). The remaining settings are left at the default ($M = 4$ attention heads, no regularization). We train with one graph per update for 32 epochs.

All runs are repeated with 5 different random seeds. We track the validation performance over the course of training. Figure 9 shows the highest validation performance in each hyperparameter combination, with the results of all models given in Figure 26. For the test performance reported we take the hyperparameter combination with the best average validation performance and use the checkpoints with highest validation performance over the course of training.

I Baby AI Experiment

To test our architecutres in an RL environment, we base ourself on the open source implementation of the working memory graph agent presented by Loynd et al. (2020). Specifically, we take a factored representation of the environment and let the agent learn, what it needs to remember. Please refer to Loynd et al. (2020) for the core ideas here. In our experiments, we slightly modify their code, replacing the multi layer perceptron for the policy and value outputs with a simple linear layer each. Also, we replace the ReLU non-linearities used with GeLUs. These modifications are done to reflect a similar setup as investigated in the rest of the paper. Similarly we fix $M = 4$, $L = 2$ and set the "WMG Memo size" equal to the model dimension d. We chose level 3 of the Baby-AI

Table 6: Hyperparameters used in the reinforcement learning experiment. These are based on the hyperparameters which Loynd et al. (2020) report in their Appendix in Table 8.

Hyperparameter	Value
A3C t_{max}	1
Adam eps	1e-12
Discount factor γ	0.95
Entropy term strength β	0.1
Gradient clipping threshold	128.0
Reward scale factor	8.0
WMG Memos	2

environment (Chevalier-Boisvert et al., 2019) based on the results presented by Loynd et al. (2020). That is, we chose a task which trains decently fast and which should be solvable for a variety of hyperparameter combinations given the allocated training time. For the hyperparameters which do not alter the architecture, we used those which Loynd et al. (2020) reported to work best for their architecture. For completeness, we list them in Table 6. The complete result of all architectures investigated are given in Figure 27. Also here we ran every hyperparameter combination with 5 different random seeds.