PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

A DIFFERENTIABLE ADAPTIVE COMPUTATION TIME ALGORITHM FOR NEURAL NETWORKS

CRISTÓBAL EYZAGUIRRE

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
ÁLVARO SOTO

Santiago de Chile, July 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

A DIFFERENTIABLE ADAPTIVE COMPUTATION TIME ALGORITHM FOR NEURAL NETWORKS

CRISTÓBAL EYZAGUIRRE

Members of the Committee:

ÁLVARO SOTO

HANS LÖBEL

JORGE PÉREZ

SEBASTIÁN VICUÑA

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, July 2021

Gratefully to my family and friends

ACKNOWLEDGEMENTS

To my advisor Álvaro Soto who welcomed me into his research group when I was just starting to learn about artificial intelligence. Without his invaluable guidance and mentorship I would never have been able to write this thesis.

To the IALab group members for their help, teaching and support in all areas. I would particularly like to thank my office mates Felipe del Río and Vladimir Araujo who collaborated with the experimentation, writing and publication of some of the intermediate results presented here.

To my friends Francisco Rencoret and Raimundo Manterola who helped with the proof reading of this document in addition to providing stimulating discussions.

Finally, I want to thank my family for their support, encouragement and blind faith in my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Despite the substantial improvements in results brought about by neural network models, their extensive application has been limited by their high computational cost due to redundancies. Furthermore, this thesis postulates that these inefficiencies cannot be completely solved with static methods, since some redundancies are intrinsic to the problem being solved and, therefore, are data-dependent. Although dynamic architectures that adapt to the input have been proposed in response to this problem, they all share the limitation that they are not fully differentiable. Responding to this common limitation this work proposes the first implementation of a dynamic computation algorithm that is fully differentiable: a differentiable dynamic early exiting algorithm we call DACT.

We validated the advantages of our approach both in terms of results and interpretability using two of the most common use cases and find that: i) DACT can lead to significant performance gains when replacing existing dynamic approaches, and ii) DACT can help eliminate intrinsic redundancies when used to augment static models. Indeed, in the domain of NLP we find that our approach is better at reducing the number of Transformer blocks used by BERT models without loss in performance on a suite of tasks. Similarly, we show a significant reduction in the number of recurrent steps needed when applied to the MAC architecture, surpassing the results of both existing adaptive algorithms and comparable static ones while improving model transparency. Furthermore, our model shows remarkable stability, responding predictably to changes in hyper-parameters while trading off precision and complexity sensibly.

Keywords: Deep Learning, Dynamic Architectures, Early Exiting, Efficient Transformers, Model Interpretability.

RESUMEN

A pesar de las mejoras sustanciales en los resultados que aportan los modelos de redes neuronales, su aplicación generalizada se ha visto limitada por su elevado coste computacional debido a redundancias presentes en este tipo de arquitecturas. Más aún, esta tesis postula que dichas ineficiencias no pueden resolverse completamente con métodos estáticos, debido a que algunas redundancias son intrínsecas al problema que se resuelve y, por lo tanto, son dependientes de los datos. Aunque en respuesta a este problema se han propuesto arquitecturas dinámicas que se adaptan a la entrada, todas ellas comparten la limitación de que no son totalmente diferenciables. Ante esta limitación común, nuestro trabajo propone la primera implementación de un algoritmo de tiempo de cómputo dinámico que es totalmente diferenciable: un algoritmo dinámico diferenciable de *early exiting* que llamamos DACT.

Validamos las ventajas de nuestro enfoque, tanto en términos de resultados como de interpretabilidad, utilizando dos de los casos de uso más comunes, y descubrimos que el DACT puede conllevar: i) importantes ganancias de rendimiento cuando sustituye a los enfoques dinámicos existentes, o ii) eliminar las redundancias intrínsecas cuando se utiliza para complementar modelos estáticos. De hecho, en el dominio del procesamiento de lenguaje descubrimos que nuestro enfoque es mejor para reducir el número de bloques *Transformer* utilizados por los modelos *BERT* sin pérdida de desempeño en una serie de tareas. Del mismo modo, mostramos una reducción significativa en el número de pasos recurrentes necesarios cuando se aplica a la arquitectura *MAC*, superando los resultados tanto de los algoritmos adaptativos existentes como de aquellos estáticos comparables, a la vez que se mejora la transparencia del modelo. Además, nuestro modelo muestra una notable estabilidad, respondiendo de forma predecible a los cambios de los hiperparámetros, a la vez que equilibra la precisión y la complejidad de forma razonable.

Palabras Claves: Aprendizaje Profundo, Arquitecturas Dinámicas, Transformers Eficientes, Modelos Interpretables.

1. INTRODUCTION

1.1. Motivation

In the past few years, Deep Learning (DL) techniques have achieved state-of-the-art performance in most, if not all, computer vision (Krizhevsky, Sutskever, & Hinton, 2012; Redmon, Divvala, Girshick, & Farhadi, 2016; He, Gkioxari, Dollár, & Girshick, 2017; D. Hudson & Manning, 2018) and natural language processing (Devlin, Chang, Lee, & Toutanova, 2019) tasks. Unfortunately, the benefits of using a powerful model are generally also accompanied by a highly demanding computational load. Indeed, research done under the banner of increasing performance metrics such as accuracy have led to computationally inefficient models that are intensive both during training and inference. Among others, some immediate consequences are longer model training times and limited applicability to mobile devices (S. Han, Mao, & Dally, 2015).

The use of increasingly demanding models also has major implications in terms of the environmental impact of AI technologies, a problem that is gaining considerable attention. As an example, recent research provide an estimation of the carbon footprint of several NLP models, concluding that current AI models are becoming environmentally unfriendly (Strubell, Ganesh, & McCallum, 2019). Similarly, other works argue about the relevance of including computational efficiency as an evaluation criterion for research and applications related to artificial intelligence (Schwartz, Dodge, Smith, & Etzioni, 2019). In spite of this increasing need, research to improve computational efficiency of DL models is still limited.

An argument can also be made that reducing computation can lead to improved model transparency. As an example, recent work has proved that interpretability decreases with the number of layers in a model (Barceló, Monet, Pérez, & Subercaseaux, 2020). Similarly this work will show how reducing computation can boost the usefulness of traditional interpretability techniques such as analyzing attention (Y. Wang, Huang, Zhu, &

Zhao, 2016; Lee, Shin, & Kim, 2017; Ghaeini, Fern, & Tadepalli, 2018) and allowing for gradient based techniques such as Integrated Gradients (Sundararajan, Taly, & Yan, 2017).

This thesis argues that as models grow, internal computational redundancies multiply as well. We distinguish two sources for redundancies: model redundancies and intrinsic data-dependent ones. In the first category we include computational redundancies present in the model weights and structure such as repeated operations, inefficient implementations, unused connections, *etc*. The second kind of redundancies considers those that arise from the data itself. For example, convolutional networks for visual recognition waste a lot of computation processing sectors of the image in which the object is clearly not present (*e.g*. the sky). Although ultimately both redundancies are model redundancies (because it is the model that does the unnecessary processing), the second group encompasses those redundancies that are data-dependent.

Research in this area has primarily focused on adapting the massive models to more efficient **static** ones through elaborate data augmentations, model compression, or by using efficient approximations of intensive operations. The success of this kind of approaches lies in the fact that, while high parameter-counts help training (Zhang, Bengio, Hardt, Recht, & Vinyals, 2016; Belkin, Hsu, Ma, & Mandal, 2019; Kaplan et al., 2020), models that are similarly performant can be achieved with significantly less computation (S. Han et al., 2015; Sanh, Debut, Chaumond, & Wolf, 2020; Frankle & Carbin, 2018; H. Zhou, Lan, Liu, & Yosinski, 2019). Indeed, recent works have been successful in training performant small models using a process called distillation, through which small efficient models can be effectively trained by taking advantage of big inefficient models, extracting knowledge from them and using it to teach a student model.

Regardless of how successful static approaches have been in alleviating models computational redundancies, this work proposes that they are not capable of reducing intrinsic redundancies. Therefore this thesis focuses on an alternative approach which instead is concerned with creating **dynamic** architectures capable of adapting their computational

graphs depending on the input. Both research directions, efficient static models and dynamic models, are compatible and can be combined to take advantage of the benefits offered by each, and we look forward to future research that combines both approaches. In particular, this work focuses on modifying sequential reasoning neural networks (which most of them are) to make them dynamic by adding early stopping capabilities.

Of course, not all models benefit equally from every computation reduction technique, there are models and situations which are better suited for dynamic computation to reduce complexity. We present two distinct cases where that happens in this thesis. First, we argue that the tasks with which NLP Transformer models are pre-trained are more complex than the task they are later adapted and used for, resulting in some of the computation becoming redundant after fine-tuning model weights. Second, we also argue that some tasks exist where there is a significant variance in complexity between the different instances, therefore creating an opportunity for reducing computation for easy inputs and reserving the maximum computation for more complex ones.

1.2. Redundancy in Fine-Tuned Models

Recent works have analyzed the parameter redundancy and over-parametrization present in state-of-the-art models that make efficient static alternatives possible. Indeed, parameter counts can be reduced by upwards of 90% without compromising accuracy in several applications (LeCun, Denker, & Solla, 1990; S. Han et al., 2015; Frankle & Carbin, 2018; H. Zhou et al., 2019). This is especially valid for the massive Transformer (Vaswani et al., 2017) based models used in NLP such as BERT (Devlin et al., 2019) and RoBERTa (Y. Liu et al., 2019) which have gained popularity mainly due to their success in adapting to a wide variety of NLP tasks (Rogers, Kovaleva, & Rumshisky, 2020), although recently they have seen applicability to image (Dosovitskiy et al., 2020; Li, Yatskar, Yin, Hsieh, & Chang, 2019) and video understanding (Caron et al., 2021). The training of increasingly parameter heavy Transformers have led to significant computational redundancies (Kovaleva, Romanov, Rogers, & Rumshisky, 2019; Rogers et al., 2020) even for the pre-training task.

Consequently, recent works have explored strategies to develop more parameter-efficient static versions of BERT based models (Sanh et al., 2020; Jiao et al., 2019).

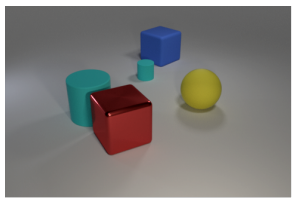
More interesting for the purposes of this thesis is the exacerbation of existing redundancies when adapting huge Transformer models like BERT to simpler tasks by fine-tuning. The usual pipeline consists of fine-tuning BERT by adapting and retraining its classification head to meet the requirements of a specific NLP task. However, the task BERT was trained to solve requires so much more than we normally need for everyday text analytics that common sense indicates that some of the internal processing done by the model is no longer necessary for the simple task. For example, it makes sense that attention heads that track named entities may not be needed for sentiment analysis.

Furthermore, because Transformers are multi-layered and build up increasingly advanced representations as the depth increases through composition, it makes sense that some of the final layers are unnecessary for simpler tasks. Indeed, exploratory work investigating where specific knowledge is generated in BERT shows that the last layers encode features specific to the pre-training task (Kovaleva et al., 2019), which provides a compelling argument for early stopping as we can reduce computation by removing layers that are no longer needed. In point of fact, recently several techniques have been successful in proposing adaptive extensions to BERT to control how many Transformers blocks are used (Xin, Tang, Lee, Yu, & Lin, 2020; W. Liu et al., 2020; W. Zhou et al., 2020).

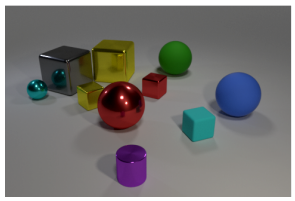
1.3. Redundancy from variable difficulty tasks

Some tasks contain questions that vary greatly in complexity. We argue that if the complexity needed to answer varies, then it makes intuitive sense that model complexity should vary accordingly. The ability to adaptively allocate more resources to difficult tasks is one that all humans possess and is evident in the increased requirements needed for complex mathematics compared to simple everyday tasks. Despite this, most state of

the art models use fixed processing pipelines that do not adapt the architecture depending on the data.



Q: *What* ***shape*** *is the red thing?*
A: *Cube*



Q: *There is a large ball that is on the* ***right side*** *of the rubber sphere that is* ***behind*** *the tiny cyan object on the* ***left side of*** *the large green matte thing; what is its* ***color****?*
A: *Blue*

Figure 1.1. Examples of questions in the CLEVR (Johnson et al., 2016) dataset that show a significant variation in the number of reasoning steps that are needed to answer them correctly.

This is the case of new Visual Question Answering (VQA) scenarios that have been proposed to support research in the area of visual reasoning, such as the CLEVR and GQA datasets (Johnson et al., 2016; D. A. Hudson & Manning, 2019a). These datasets pose challenging natural language questions about images whose solution requires the use of perceptual abilities, such as recognizing objects or attributes, identifying spatial relations, or implementing high-level capabilities like counting. As an example, Figure 1.1 shows two instances from the CLEVR dataset (Johnson et al., 2016). In this case, each visual question entails a different level of complexity to discover the correct answer. Specifically, while the first question involves just the identification of a specific attribute from a specific object, the second question requires the identification and comparative analysis of several attributes from several objects. Furthermore, this disparity is exacerbated in real world datasets such as Visual Genome (Krishna et al., 2017), as the open-ended nature of the questions and images could lead to some seemingly similar questions actually requiring different processes to solve.

Modular architectures (Hu, Andreas, Rohrbach, Darrell, & Saenko, 2017; Johnson et al., 2017) are an effective way to solve complex multi-step reasoning problems and work by using a policy (sometimes presented explicitly as a Policy Network or Controller) to decompose the problem into smaller steps. Component steps are then tackled individually by either specialized or general purpose processing modules, which are combined to generate the final output. When the complexity and number of modules used is conditioned on the inputs models can adapt the processing pipeline to match the difficulty of each instance. However, this adaptive behavior requires additional supervision or elaborate reinforcement learning training schemes in order to train the policy.

On the other hand, semi-modular fully differentiable alternatives that repeat a single general purpose module (D. A. Hudson & Manning, 2019b; D. Hudson & Manning, 2018) are advantaged in that they can be trained end to end, but lose the capability to adapt to different inputs. Indeed, the number of times the module is repeated is fixed as a hyper-parameter that is later manually adjusted. We argue that in manually tuning it's value with the objective of maximizing accuracy results in models calibrated for the most complex cases, overestimating the computational load needed to solve easier ones. This results in suboptimal behavior, over-processing in simple cases and possibly under-processing in the more complex ones.

1.4. Contributions

The main contribution of this work is the proposal of a new dynamic approach to adaptive computation via early stopping based on a novel attention-based formulation. As key insight, this mechanism addresses the main common limitation of previous approaches, namely their non-differentiability. This persistent issue arises because varying the architecture requires the use of piecewise functions. To overcome this problem we use a soft approximation of the gating mechanism during training, and only adapt during inference. In addition, we derive criteria for stopping computation that allow us to give theoretically

backed guarantees about the correctness of the response. The result is a fully differentiable model that can be trained using gradient descent whose computation can be reduced by mathematically determining when an interruption of the processing pipeline does not harm its final performance.

We validate the performance of our approach by substituting our mechanisms instead of those used for early-stopping in existing dynamic Transformer models (Xin et al., 2020; W. Zhou et al., 2020). Furthermore, motivated by the evident advantages that such a mechanism might provide to modern module networks, we propose a new fully-differentiable adaptive visual reasoning model. Both experiments, in natural language processing and visual reasoning, demonstrate a significant improvement in computational efficiency while also improving interpretability.

It is important to note here that some of the intermediate results obtained in the process of writing this thesis have been published (Eyzaguirre & Soto, 2020) or are in the process of being published.

2. BACKGROUND INFORMATION

This section introduces relevant algorithms and models that will be used in this thesis.

2.1. Adaptive Computation Time Algorithm

Adaptive Computation Time or ACT (Graves, 2016) is an algorithm that allows recurrent neural networks to learn how many steps are needed to process a specific input. This is achieved by adding an additional sigmoidal output called the *halting value* after each step which will determine the probability of continuing to process the current input for another step. The resulting model allows for the propagation of gradients with respect to the weights that determine the halting, which enables the network to learn a halting policy.

The main challenge with adaptive computation is constructing a combination function from the *halting values* such that after some step N the output of the module wont change, making it unnecessary to keep computing. In the case of ACT, the authors propose combining sequential hidden states from RNNs by computing their weighted average where the weighting is defined by a piece-wise probability function p_n.

$$p_n = \begin{cases} R(n) & \text{if } n \geq N \\ h_n & \text{otherwise} \end{cases} \tag{2.1}$$

where $R(n)$ is the remainder of the amount to reach one:

$$R(n) = 1 - \sum_{i=1}^{n-1} p_i \tag{2.2}$$

It follows from the definition that for this to describe a valid probability distribution ($0 \leq p_i \leq 1$ and $\sum_{i=1}^{\infty} p_i = 1$) then N must be chosen such that the remainder is positive. In particular, the ACT algorithm sets the value of N to the last step before the remainder

becomes negative.

$$N = \min\{n' : \sum_{n=1}^{n'} h_n \geq 1\} \tag{2.3}$$

Consequently, N marks the last significant (non-zero probability) step as for every step $n \geq N$ the remainder $R(n)$ (and therefore the probability p_n) will be zero (as $p_n = R(n) = 1 - 1$).

The probability distribution described is then used to calculate the mean field updates for the hidden states of recurrent neural networks by computing the weighted average of the hidden state of every step s_n.

$$s = \sum_{n=1}^{\infty} p_n s_n \tag{2.4}$$

In practice only the first N steps have to be computed to obtain s as the rest of the steps are multiplied by zero, allowing for dynamic reductions in computation. A pseudo-code implementation is shown in Algorithm 1, in which a recurrent cell is allowed to iterate infinitely until the condition that $\sum_{n=1}^{n'} h_n \geq 1 - \epsilon$ is reached, terminating the loop. The slack term ϵ is introduced to allow the network to exit after a single iteration and is set to a small value (*e.g.* 1×10^{-3} in our experiments). An efficient PyTorch (Paszke et al., 2017) implementation that can run on accelerators and is batch processable was implemented for this thesis and has been made available online [1].

Finally, to encourage reduced computation, a proxy of total computation $\rho = N + R$ can be added to the loss to serve as a regularizer against high amounts of computation.

$$\hat{L} = L(x, y) + \rho \tag{2.5}$$

Although the formulation isn't fully differentiable due to the use of piece-wise functions in Equations 2.1 and 2.3, gradients are still propagated through the residual. Indeed, minimizing the residual provides gradients equivalent to maximizing the halting values prior to the last step, therefore incentivizing reduced computation.

[1] `https://github.com/ceyzaguirre4/adaptive_computation`

Algorithm 1 *ACT*

Input: S recurrent cell
Input: x input
Input: ϵ slack

$s_n \leftarrow \vec{0}$
$s \leftarrow \vec{0}$
$R \leftarrow 1$
while `True` **do**
    # Run recurrent cell S for the nth time
    $s_n \leftarrow S(s_n, x)$
    $h_n \leftarrow GetHalt(s_n)$

    # Calculate probability p_n and update the remainder
    if $R - h_n \geq \epsilon$ **then**
        $R \leftarrow R - h_n$
        $p_n \leftarrow h_n$
    else
        $p_n \leftarrow R$
        $R \leftarrow 0$
    end if

    # Combine the nth hidden state with previous ones
    $s \leftarrow s + p_n \cdot s_n$

    # Stop computation if answer cannot change
    if $R = 0$ **then**
        break loop
    end if
end while

Output: Combined hidden states s

2.2. Threshold Early Stopping

A different approach to allowing the neural network to learn how many recurrent steps are needed (as is the case in ACT (Graves, 2016)) is to first train a computationally expensive model, and later add shortcuts. The shortcuts typically take the form of classifiers which are added at lower depths between modules in the architecture to obtain an output without needing to execute deeper layers. The theory that backs early stopping techniques is that intermediate representations learnt by the computationally expensive model may

contain sufficient information to answer some of the "easier" inputs. The challenge is identifying when the output of the intermediate classifier is good enough.

Deciding which of the N classifiers to trust requires using a metric that indicates the confidence that each classifier has in the accuracy of its prediction. Then, an external hyper-parameter (normally found through trial and error) is used as a confidence threshold. Namely, if the confidence after running classifier n is high enough, then halt computation and avoid running subsequent layers and classifiers. Algorithm 2 illustrates how this kind of model generally works, iteratively running deeper and deeper layers until the threshold criteria is met, which indicates that the model is confident enough in its prediction that running further layers isn't necessary.

Algorithm 2 *ConfidenceThresholdEarlyStopping*

Input: M model with $N = max\_steps$ modules, each followed by a classifier
Input: confidence threshold $threshold$

for step $n = 1, 2, \ldots max\_steps$ **do**
    # Get output after running nth module and classifier
    $y_n \leftarrow GetOutputModule(M, n)$

    # Get confidence from output
    $h_n \leftarrow GetConfidence(y_n)$

    # Stop computation if confidence is high.
    if $h_n \geq threshold$ **then**
        break loop
    end if
end for

Output: Approximate final answer y_n

Because the halting decision is non differentiable no gradients are propagated to the confidence function, making it impossible for it to be learnt jointly. Consequently, most confidence functions are based on fortuitous heuristics such as the magnitude of the probability assigned to the top class in the output vector, the entropy of said vector (Xin et al., 2020; W. Liu et al., 2020), or the number of times intermediate classifiers have agreed on an output class (Sun, Cheng, Gan, & Liu, 2019).

2.3. Transformers

Transformers encoders such as BERT (Devlin et al., 2019), RoBERTa (Y. Liu et al., 2019), GPT (Radford et al., 2019), *etc*. are models composed by stacking multiple Transformer blocks sequentially, such that the output of one block feeds the following one (see Figure 2.1). Each block acts as a sequence-to-sequence model, receiving a $\mathbf{X}^{N\times d_1} = \{x_i^{d_1}|_{i=1}^{N}\}$ tensor containing N embedding vectors x_i, each of dimension d_1, and outputting an $N \times d_2$ tensor containing the transformed values of each of the input vectors.

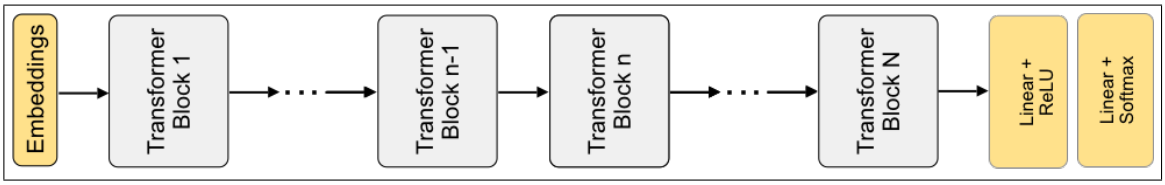


Figure 2.1. Transformer Encoder models consist of several stacked blocks, the number of which depends on the model and use-case. When used for natural language processing an additional embedding layer is typically prepended to facilitate processing. The output probabilities are obtained by passing one or more sequence elements through a classifier. A two layer multi-class class classifier is shown in the diagram.

Each Transformer block is composed of the same two operations. First, a self-attention layer is used such that each one of the N input vectors is transformed into a pondered sum of projections of all the other vectors called *value* vectors $v_i = \mathbf{W}_V^{d_2\times d_1} x_i$.

$$x_i' = \sum_{j=1}^{N} v_j \cdot \alpha_{i,j} \tag{2.6}$$

The weights $\alpha_{i,j}$ are obtained by comparing projected views of each *query* vector $q_i = \mathbf{W}_Q^{d_3\times d_1} x_i$ with every *key* vector $k_i = \mathbf{W}_K^{d_3\times d_1} x_i$ to obtain the unnormalized alignment scores $\beta_{i,j}$, and then normalizing with a Softmax function to obtain the attention weights:

$$\beta_{i,j} = \frac{q_i \cdot k_j}{\sqrt{d_3}} \tag{2.7}$$

$$\alpha_{i,j} = \frac{e^{\beta_{i,j}}}{\sum_{k=1}^{N} e^{\beta_{i,k}}} \tag{2.8}$$

Every output of the self attention layer is then fed into the same two layer neural network called the point-wise feed-forward layer. Importantly, residual connections and layer normalization are added after both the self-attention layer, and the point-wise feed-forward layer to aid convergence.

2.4. MAC

The MAC architecture (D. Hudson & Manning, 2018) is a recurrent architecture designed for iterative reasoning tasks that require an external knowledge base. The input query sequence is translated into a series of distributed *reasoning operations* by attending (Bahdanau, Cho, & Bengio, 2015) different parts of the query. Each *reasoning operation* represents a control signal for an iteration of the MAC cell and informs the other modules in the architecture. First, the control vector is used to soft-select a specific entry in the knowledge-base through another attentional mechanism. Then, a working memory vector, analogous to hidden states in recurrent models, is updated using a function of both the control signal and the knowledge-base entry. The models that results from stacking multiple cells consume the external knowledge-base iteratively and selectively in a series of discrete reasoning steps.

One particular use-case for such a model is answering questions about images (Goyal, Khot, Summers-Stay, Batra, & Parikh, 2017), as the input question can be transformed into a sequence of control signals, and the image can be decomposed into a set of regions. Indeed, MAC achieved state of the art results in the challenging visual reasoning dataset CLEVR (Johnson et al., 2016) while being more transparent compared to previous approaches thanks to the easily interpretable attention maps.

Replicating the results from the original MAC paper (D. Hudson & Manning, 2018) is non trivial as several hyper-parameters must be tuned and some modules must be modified.

As an additional contribution the PyTorch (Paszke et al., 2017) code developed for MAC during the completion of this thesis was made public online [2].

[2] https://github.com/ceyzaguirre4/mac-network-pytorch

3. RELATED WORK

Chapter 1 highlights the need for suitable mechanisms to control computational complexity in DL models for several use-cases, including improving interpretability, efficiency, and reducing the environmental impact of models. However, in spite of this increasing need, research to make DL models more dynamic is still limited.

Out of the taxonomy of dynamic deep neural networks proposed by prior work (Y. Han et al., 2021), this investigation focusses on instance-wise dynamism, *i.e.* when the processing pipelines are data-dependent, adapting the computational graph to the content of each specific input. This is different from neural networks that result in varying amounts of computation due to the format of the input such as multi-resolution convolutional networks, language models that accept variable length inputs, or graph networks. An example of a dynamic model is a convolutional neural network which increases the number of layers used for difficult images compared to the those used for simpler ones. Importantly, the input dependance requires control modules that decide which parts are executed, and these must be conditioned on the input. Furthermore, instance-wise approaches are easily adapted into global ones by simply using a fixed prior and removing the conditioning on the input.

The desired behavior of adapting the architecture to specific inputs is often obtained through one of three mechanisms (Y. Han et al., 2021):

- A proxy of model confidence can be used within a user defined (fixed) policy that decides when to skip sections of the architecture typically by exiting early (Huang et al., 2017; Teerapittayanon, McDanel, & Kung, 2016; Xin et al., 2020; Park et al., 2015; W. Zhou et al., 2020). Confidence scores are generally taken directly from the output probabilities, or calculated as a function of those. Models that skip intermediate sections take advantage of skip connections (He, Zhang, Ren, & Sun, 2015) and avoid processing residual blocks when the residual is estimated to be of small magnitude (and therefore thought irrelevant to the

output). On the other hand, early exiting models are those that take advantage of intermediate classifiers that can guess the final output "early", ie. without completely running the inference pipeline (see Section 2.2).

- Another model (called the *Policy Network*) conditioned on the input decides the topology of the architecture in a typically two stage process by combining multiple specialized modules (Wu et al., 2018; Andreas, Rohrbach, Darrell, & Klein, 2016; Johnson et al., 2017). In the case of specialized modules, the generation of the sequences required costly supervision or elaborate reinforcement learning training schemes. Some module networks mentioned in Section 1.3 belong in this class of adaptive models as the number and complexity of modules used is sometimes conditioned on the input question.
- Gating mechanisms built into the architecture allow the model to control the flow of information by opening or restricting sections of the computational graph. Despite their generality and applicability, these models are limited by their lack of differentiability and require reinforcement training (Lin, Rao, Lu, & Zhou, 2017; X. Wang, Yu, Dou, Darrell, & Gonzalez, 2018) or approximating the gradients (Bejnordi, Blankevoort, & Welling, 2020; Veit & Belongie, 2018; Verelst & Tuytelaars, 2020; Xie, Zhang, Zhu, Huang, & Lin, 2020; Voita, Talbot, Moiseev, Sennrich, & Titov, 2019).

This work proposes a novel gating mechanism that is fully-differentiable (and therefore can be trained straightforwardly by gradient descent during training), while still enabling the model to exit early during inference using a confidence-based policy. We test our approach in two distinct paradigms. First against early-exiting Transformer models that use policies based on model confidence, where we find it outperforms state-of-the-art techniques in lower computation regimes. Second, we apply both ACT and DACT algorithms to a complex recurrent model to show the advantages our differentiable version offers in terms of computation and interpretability.

3.1. Dynamic Transformers

The recent rise in popularity of Transformer-based language models like BERT (Devlin et al., 2019), led to a renewed interest in improving the efficiency of this kind of deep models. Most research in this area focuses on static approaches: training more compact static models by weight tying to reduce parameters (Lan et al., 2020; Bai, Kolter, & Koltun, 2019; Dehghani, Gouws, Vinyals, Uszkoreit, & Kaiser, 2019), or using distillation to train student models with extracted knowledge from larger pre-trained teachers (Sun et al., 2019; Jiao et al., 2020; Sanh et al., 2020). However, some investigations instead focus on dynamic approaches. Typically, these use the previously trained model as a base, but allow it to dynamically adapt to different computational paths for each input instance during inference.

Recently, a series of algorithms have been proposed to reduce computation in Transformer language models based on early exiting (Kaya, Hong, & Dumitras, 2019). Models such as DeeBERT (Xin et al., 2020), FastBert (W. Liu et al., 2020), and PABEE (W. Zhou et al., 2020) introduce intermediate classifiers after each Transformer block. These classifiers are then trained independently from the rest of the model (not end-to-end). After both training stages, a "halting criterion" is used to dynamically determine the number of blocks needed to perform a specific prediction. Instead of using a brittle confidence approach (Guo, Pleiss, Sun, & Weinberger, 2017) to determine when to stop, recent approaches rely on computing the Shannon's entropy of the output probabilities (Xin et al., 2020; W. Liu et al., 2020) or counting the number of times intermediate classifiers have agreed on an output class (W. Zhou et al., 2020). As is the case with most confidence threshold based early-exit approaches, a common limitation is that they are non-differentiable and use a fixed metric which is not adapted for the purpose.

Notably, the fact that our formulation is end-to-end differentiable allows us to fine-tune the weights of the underlying backbone (*i.e.* the Transformer blocks and embedding layers) using a joint optimization process that trains the intermediate classifiers. This

stands in contrast to all existing methods for dynamic Transformers, where training is done in two stages first pre-training the backbone (and not the intermediate classifiers) and later freezing the backbone and only modifying the weights of the classifiers.

Furthermore, we highlight that both DeeBERT and FastBERT can be expressed and trained with our model. For DeeBERT this can be achieved by hamstringing DACT and training it in three stages. The first two stages consist of the same training regime as DeeBERT, first training only the transformer blocks and the final classification layer, and subsequently freezing the transformer blocks and training the intermediate classifiers independently. An appropriate *confidence function* to force the model to adapt based on entropy is a simple logistic regression that receives the output entropy and consists only of a bias to encode the entropy threshold, and a single weight to scale the layer response. Training at this point will yield different learnt entropy thresholds for each layer (which is positive) but, in order to fully represent DeeBERT, we need to further handicap the model by tying the biases in all the *confidence functions*. FastBERT can be obtained similarly by simply replacing the second training stage with the distillation process described by (W. Liu et al., 2020).

In consequence, DACT-BERT can be seen as a generalization of existing dynamic Transformer architectures, allowing us to train these in an end-to-end fashion while eliminating the need for manual tuning of the entropy hyperparameter. In this sense, in our experiments, we note that the training of only the *confidence functions* results in a optimization problem that produces highly stable solutions, suggesting that SGD is able to find suitable optimum entropy thresholds [1].

3.2. Adaptive Computation Time for RNNs

In the context of recurrent networks, Graves introduces Adaptive Computation Time or ACT (Graves, 2016), an algorithm for dynamic early-stopping in Recurrent Neural

[1] for a small enough learning rate.

Networks (RNN). The key idea behind ACT is to add a sigmoidal halting unit to an RNN that, at each inference step, determines if the RNN should stop or continue its processing. These activation values are then used to construct the model's final output as a weighted sum of the hidden states of all previous recurrent steps. This is achieved through a series of non-differentiable piece-wise operations mainly used to enforce a hard limit so that no subsequent iteration changes the model output. As we show in this work, this results in noisy gradients that do not handle properly the information about the number of processing steps being used.

Despite its limitations, ACT has been applied to multiple tasks beyond the synthetic cases reported in the original work (Graves, 2016). It has been used to improve results on the LAMBADA language modeling dataset using a Universal Transformer architecture (Dehghani, Gouws, Vinyals, Uszkoreit, & Kaiser, 2018), achieving state-of-the-art performance. Also, on the challenging task of character level language modeling, it has been used to dynamically increase the attention span of a Transformer model, achieving state-of-the-art performance on the text8 and enwiki8 datasets (Mahoney, 2011). Furthermore, on the natural language reasoning corpus SNLI dataset, it has been reported to boost performance and interpretability (Neumann, Stenetorp, & Riedel, 2016). In terms of visual recognition, ACT has been used to dynamically choose the number of executed layers for different pre-defined regions in an input image, improving performance in terms of computational efficiency and model interpretability (Figurnov et al., 2016).

While similar in function, our approach to adaptive computation has substantial differences with respect to ACT. ACT achieves halting by forcing that the weights used to combine each step's output into the final answer sum exactly one. To attain this behavior a non-differentiable piecewise function is used, namely: if the sum of the confidence scores is more than one, then change the last weight so that the sum is exactly one. In contrast, our approach maintains the full gradient by using a smooth gating-function for training, and only halting during inference. Furthermore, instead of averaging hidden states from recurrent networks, we calculate compound probabilities using the output distributions

of each intermediate classifier, which carries the additional advantage of eliminating the assumption that the hidden states are approximately linear.

4. PROPOSED METHOD

In this section we present our method to implement a dynamic early-stopping algorithm that, similar to previous works in adaptive computation for RNNs (Graves, 2016) utilizes a *halting-neuron* that the model can use to cut computation early. The novelty of our method lies mainly in the fact that, to the best of our knowledge, it remains the only fully-differentiable dynamic computation approach. The main intuition for understanding how our algorithm works for early exiting is that we use a novel soft gating approach such that at any point the model can decide to limit the contribution of future steps by using *halting values*. As we will show later our gating allows us to compute upper and lower bounds for the probabilities of each output class, something we leverage to derive a criterion to stop early during inference. Then, during inference, we can use the *halting values* to determine when it makes sense to stop early. Section 4.1 describes the differentiable approach used to approximate a binary gating mechanism, and Sections 4.2 and 4.3 shows how the proposed gating can be exploited to reduce computation during inference.

4.1. Gating Mechanism

Different from ACT (Graves, 2016), our formulation is not specific to RNNs but rather it can be applied to any classification model or ensemble M that can be decomposed as a sequence of modules or submodels m_n, $n \in [1, \ldots, N]$. For example, recurrent networks are composed by iterative steps, CNNs by residual blocks, and ensembles by component models. This is because, as this section will show, we combine the output probabilities of intermediate classifiers (which can be added after modules of most types of neural networks) instead of combining hidden states from recurrent networks (as in Graves' case).

Each module must be structured such that it has the outputs shown in Figure 4.1. That is, each submodel m_n should produce its own prediction y_n about the output using the input X as well as any relevant information from a previous one m_r , $r < n$. Additionally,

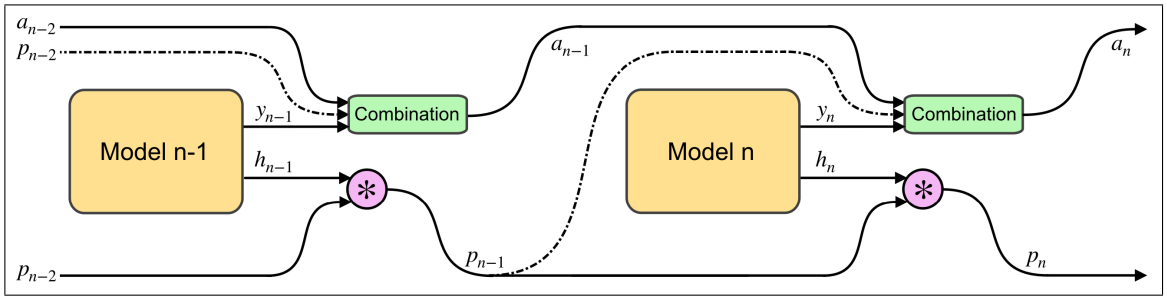


Figure 4.1. The *accumulated output* a_n is built by linearly combining a_{n-1} with the output of the nth model (following Equation 4.2). Each step can limit the contribution of future steps by maintaining or reducing the value of the scalar p_{n-1} used for the linear combination (illustrated with a dotted line). Any h_n valued roughly zero will force p_n to this value, effectively disallowing the outputs of future models from altering the current *accumulated output*, and effectively imposing that this a_n become the final output Y.

each submodel m_n should also produce a sigmoidal halting value $h_n \in [0, 1]$ that represents how uncertain is m_n about the correctness of its output y_n. We set the initial halting value $h_0 = 1$ to represent maximal uncertainty before seeing the input. The use of scalar halting values h_n is the main mechanism to achieve dynamic early-stopping. The key idea is that each module m_n can restrict subsequent ones (m_s where $s > n$) from altering the final answer given by the ensemble. With this goal in mind, let define:

$$p_n = \prod_{i=1}^{n} h_i = h_n p_{n-1} \tag{4.1}$$

The value of p_n can be interpreted as the probability that any subsequent submodel m_s, $s > n$ might change the value of the final answer of the ensemble. Consequently, we define the initial value $p_0 = 1$.

According to the previous formulation, h_n represents the uncertainty of submodel m_n, while p_n represents the uncertainty of the full ensemble considering the first n models. From Eq. 4.1 it is easy to see that the values of p_n are monotonically decreasing with

respect to index n. Also, notice that a small value of h_n forces the future values of p_n to be close to 0.

We still need to describe how to combine all *intermediate outputs* y_n ($n \in [1, \ldots, N]$) to form a_N. We achieve this by defining auxiliary accumulator variables a_n which contain the ensemble's answer up to step n. By using Eq. 4.1, we can construct a_n in such a manner that for some step n with a low associated p_n then $a_n \approx a_N$:

$$a_n = \begin{cases} \overrightarrow{0} & \text{if} \quad n = 0 \\ y_n p_{n-1} + a_{n-1}\left(1 - p_{n-1}\right) & \text{otherwise} \end{cases} \tag{4.2}$$

It follows from this definition that a_N can always be rewritten as a weighted sum of *intermediate outputs* y_n. Additionally, the sum of the weights is always equal to 1, thus describing a valid probability distribution over the *intermediate outputs* y_n. Both proofs are included in Appendix A.

Therefore, by describing what is effectively a *pair-wise* linear interpolation, we obtain a method for implicitly attending the outputs of each model in the ensemble, including succeeding ones. In this manner, what we propose is essentially a *mixture of experts* type ensemble (Jacobs, Jordan, Nowlan, & Hinton, 1991) where we remove the controller and replace the gating model for the implicit distribution described above. As a main result, by adding probabilities instead of hidden values as in ACT, we remove the assumption of ACT that the hidden states of the underlying RNN are approximately linear.

4.2. Penalizing complexity

Following the principle of Ockham's razor, we wish to reduce complexity when it is not needed by choosing simpler models in lieu of more complex ones when both provide similar results. To achieve this, we define a proxy of total computation ponder cost ρ as:

$$\rho = \sum_{n=1}^{N} p_n \tag{4.3}$$

By adding the *ponder cost* to the loss function L we encourage the network to minimize the contribution of more complex models. This is used in the next Section (4.3) to reduce computation.

$$\hat{L}(\boldsymbol{x}, \boldsymbol{y}) = L(\boldsymbol{x}, \boldsymbol{y}) + \tau\rho(\boldsymbol{x}) \tag{4.4}$$

where τ is the *time penalty*, a hyper-parameter used to moderate the trade-off between complexity and error.

Alternatively, different regularizers that bias the model towards reduced computation can be used. For example, we found that adding the L1 norm of the halting values to the loss instead of the ponder cost further binarizes the values of p_n.

4.3. Reducing computation time

The previous formulation allows us to train a model incorporating the DACT methodology. In brief, we modified the training process of the model to allow simpler models to cap the maximum impact of all subsequent ones (gating in Equations 4.1 and 4.2), and encourage the model to use this functionality as soon as possible (Equations 4.3 and 4.4). In this section we derive equations for calculating upper and lower bounds for the probabilities of each of the classes in the output vector after every module. Consecutively, we show how to obtain a halting criteria that provides theoretical guarantees of correctness for both multi-class and binary classification. As a consequence DACT based models are able to halt to save computation without affecting the outcome.

The choice of the criteria for halting (and therefore reducing computation) depends greatly on the task and how close of an approximation is required. In this work our goal is to achieve the same top-1 accuracy with and without using DACT, *i.e*. ensure that halting does not change which class has the highest probability. This is equivalent to establishing a halting criterion that identifies step n such that the class with highest probability in a_n will remain the same after running the rest of the modules. Algorithm 3 shows how the criteria is used. During training the forward pass is identical to that described in Section

4.1, iteratively improving the final output a_n by running additional modules. Then during inference the model can cut computation once it can be determined that the remaining steps cannot change the final answer. Notably, and in contrast with Algorithm 1 (ACT) and Algorithm 2 (threshold early stopping), no non-differentiable functions are used during training.

Algorithm 3 *DifferentiableAdaptiveComputationTime*

Input: M model with $N = max\_steps$ modules
Input: x input
Input: $is\_training \in \{\texttt{True}, \texttt{False}\}$
$p_n \leftarrow 1$
$a_n \leftarrow \vec{0}$
for step $n = 1, 2, \ldots N$ **do**
    # Get output and confidence after running nth module
    $y_n \leftarrow GetOutputModule(M, n)$
    $h_n \leftarrow GetHaltModule(M, n)$

    # Combine the nth output with the previous answer.
    $a_n \leftarrow (y_n * p_n) + (a_n * (1 - p_n))$

    # Compute the halting probability for future steps
    $p_n \leftarrow p_n * h_n$

    # Stop computation during inference if answer cannot change
    if $is\_training = \texttt{False}$ and not $AnswerCanChange(y_n, p_n, n, N)$ **then**
        break loop
    end if
end for
Output: Approximate final answer a_n

We approach the problem of identifying the step such that top class cannot change by studying the stability of the final answer. First, we know that y_n (the *intermediate output* of the nth classification model) is restricted to $0 \leq y_n \leq 1$ as a result of using either *Softmax* or *Sigmoid* functions. Since the maximum change of the accumulated answer a_n in the remaining $N - n$ iterations is limited by p_n, we can calculate the maximum difference between the predicted probabilities for the topmost class and the *runner-up*. Consequently, we can achieve reduced computation by halting once this difference is insurmountable.

Without loss of generality consider the case where, for some step n, the class with the highest probability in the accumulated answer a_n corresponds to class c^* with probability $\Pr(c^*, n)$, and the runner-up (second best) class is c^{ru} with probability $\Pr(c^{ru}, n)$. The minimum value for the probability of the class c^* after the remaining steps is obtained when all the future steps assign a minimum probability (0) to this class. We can use this result to obtain a lower bound to the probability:

$$\Pr(c^*, N) \geq \Pr(c^*, n) \prod_{i=n}^{N-1} (1 - p_i) \tag{4.5}$$

Leveraging that $p_n \geq p_{n'}$ (for any n' greater than n) in conjunction with Eq. 4.2, we can establish that the minimum value for the class at c^* after another $N-n$ steps is always:

$$\Pr(c^*, N) \geq \Pr(c^*, n)(1 - p_n)^{N-n} \tag{4.6}$$

Likewise, the maximum value that the probability for the runner-up class c^{ru} can take after all unused steps ($\Pr(c^{ru}, N)$) is achieved when the maximum probability (1) has been assigned to this class at every remaining step. Replacing this value into Eq. 4.2 yields an upper bound to the value that the probability for the class c^{ru} can take:

$$\Pr(c^{ru}, N) \leq \Pr(c^{ru}, n) \prod_{i=n}^{N-1} (1 - p_i) + \sum_{i=n}^{N-1} p_i \prod_{j=i+1}^{N-1} (1 - p_j) \tag{4.7}$$

Then, since $0 \leq p_n \leq 1$ and $p_n \geq p_{n'}$ $(\forall n' \geq n)$, we obtain that the maximum value for the class c^{ru} is:

$$\Pr(c^{ru}, N) \leq \Pr(c^{ru}, n) + p_n(N - n) \tag{4.8}$$

We say that the difference between the top class and the runner up is insurmountable once we prove that $\Pr(c^*, N) \geq \Pr(c^{ru}, N)$, and thus we can cut computation since the remaining steps cannot change the final answer of the model. Mathematically, this means

the *halting condition* is achieved when:

$$\Pr(c^*, n)(1 - p_n)^{N-n} \geq \Pr(c^{ru}, n) + p_n(N - n) \tag{4.9}$$

which is the criterion used in this work to stop processing.

A similar procedure can be followed to derive a criterion for binary classification problems where the answer is determined by rounding a scalar sigmoidal output y_n, which results in the following *halting condition*:

$$\begin{cases} 0.5 \geq y_n(1 - p_n)^{N-n} + p_n(N - n) & \text{if } y_n \leq 0.5 \\ 0.5 \leq y_n(1 - p_n)^{N-n} & \text{otherwise} \end{cases} \tag{4.10}$$

5. EXPERIMENTS

We evaluate the performance of our approach with two different model paradigms and in different domains. First in Section 5.1 we use DACT gating to enable BERT Transformer models to early exit and compare the results to existing dynamic algorithms that use policies based on confidence thresholding. Later in Section 5.2 we apply DACT to a complex static recurrent visual reasoning model to reduce computation and increase transparency. We also include results from the existing non-differentiable ACT algorithm as proof of its limitations.

5.1. Dynamic Early-Stopping Transformers

5.1.1. Experimental Setup

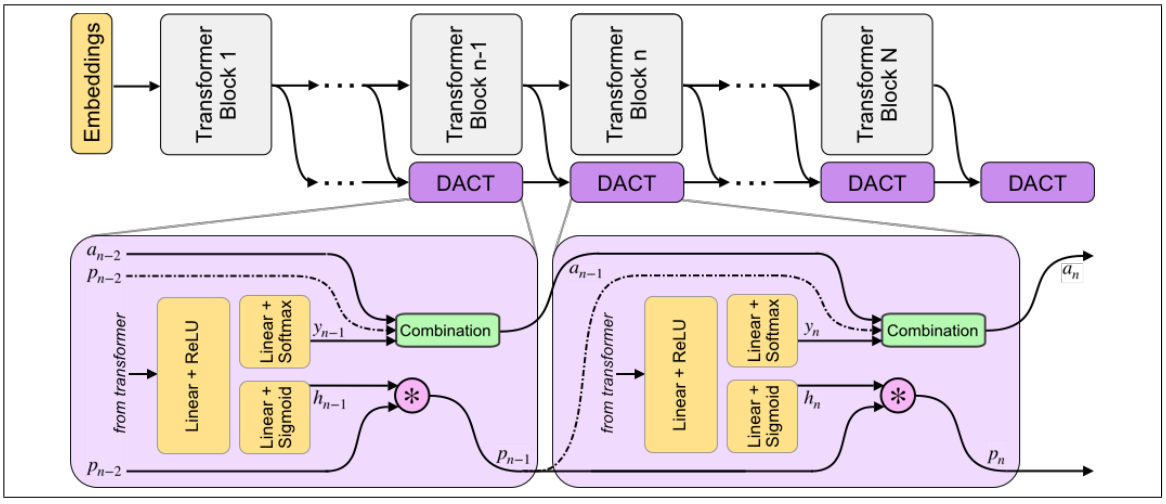


Figure 5.1. DACT-BERT adds an additional classification layer after each Transformer block, along with a sigmoidal *confidence function*. DACT-BERT combines the Transformer hidden state and the outputs and confidences of all earlier layers into an accumulated answer a_n. Later, during inference, the model is halted once $a_n \approx a_N$.

As shown in Figure 5.1, DACT-BERT introduces additional linear layers after each module, similar to those used in previous works such as the *off-ramps* in (Xin et al., 2020)

or the student classifiers in the work of (W. Liu et al., 2020). As in both these cases, we define the discrete unit of computation to be a single BERT Transformer block, *i.e.* our gating mechanism will trade precision for additional complexity in discrete units of full additional Transformer blocks. In addition to the output vector y_n with the predicted class probabilities, each n-th DACT module computes an accompanying scalar confidence score (or halting value) h_n. Following BERT (Devlin et al., 2019), both, y_n and h_n, are estimated by using the classification token ($[CLS]$) that is included in BERT as part of the output representation of each layer.

As described in Section 4 during training we combine the output vectors (Equation 4.2) using a function of the halting values (Equation 4.1) to obtain the final predicted probabilities. The intermediate results a_n, which resemble a weighted average of all previous intermediate outputs y_n, encode the models best guess after unrolling n Transformer layers. Then, during inference, the confidence scores can be used to effectively reduce computation by avoiding running all the layers using the appropriate halting criterion (Equations 4.9 and 4.10).

The regularizer used to bias the model towards reduced computation is the sum of the halting values instead of Equation 4.3 as we find empirically that this helps training convergence and further binarizes the halting probabilities while still encouraging reduced computation.

$$\hat{L}(\boldsymbol{x}, \boldsymbol{y}) = L(\boldsymbol{x}, \boldsymbol{y}) + \tau \sum_{i=1}^{n} h_i \tag{5.1}$$

The training of the module follows a two step process. First the underlying Transformer model must be tuned to the relevant task. This ensures a good starting point onto which the DACT module can then be adapted to and speeds up convergence. This is followed by a second fine-tuning phase where both the DACT module as well as the underlying Transformer are jointly trained for the relevant task. Importantly, thanks to the fully-differentiable nature of DACT, our approach adapts the Transformer layers as well as

the intermediate classifiers, potentially generating new representations in the Transformer which are useful for the new lower computation scenario.

We test our method using both BERT and RoBERTa backbones, evaluating both models on six different tasks from the GLUE benchmark (A. Wang et al., 2018) (MNLI, MRPC, RTE, SST-2, QNLI and QQP). We chose DeeBERT (Xin et al., 2020) and PaBEE as our dynamic baselines, also using both BERT and RoBERTa backbones for better comparison. We do not include results for FastBERT since both DeeBERT and FastBERT use the same entropy-threshold halting criterion. In fact, the main difference between them is the use of distillation in FastBERT, which can also be used with our proposed model, but this escapes the scope of our research as it is more aligned with static models.

Our model was developed using PyTorch (Paszke et al., 2017) on top of the public implementations of DeeBERT and PaBEE, as well as the HuggingFace Transformers library (Wolf et al., 2019) [1]. Each experiment uses a single 11GB NVIDIA graphics accelerator, which allows for training on the complete batch using 32-bit precision and without the need for gradient accumulation or checkpointing.

5.1.2. Results

To compare the trade-off that exists between computational efficiency and the corresponding performance obtained, we compute efficiency-performance diagrams. Efficiency was measured as the percentage of Transformer layers used out of the total number of layers executed without an efficiency mechanism, which is equivalent to the total number of layers in the corresponding static baseline (*e.g.* 12 in the case of BERT). The specific metrics for performance are those suggested in the GLUE paper (A. Wang et al., 2018) for each task.

In our experiments, we fine-tune the backbone model for the GLUE tasks using the default values of the hyper-parameters. For the second stage we vary the value of τ in

[1] The code for DACT-BERT will be released upon publication.

Equation 5.1 to compute our computation-performance diagrams, selecting from a set of fixed values: $\tau \in \{5 \times 10^{-5}, 5 \times 10^{-4}, 5 \times 10^{-3}, 5 \times 10^{-2}, 5 \times 10^{-1}\}$. On DeeBERT, to create the computation-performance diagrams, the entropy threshold was varied continuously, usually in increments of 5×10^{-2}. In the case of PaBEE we fluctuate the patience value between 1 and 12, effectively trying out the full range. The results for the unmodified static backbones are also included as a reference, as are the results obtained by the half-depth DistilBERT pre-trained model.

The area under the curve (AUC) in the Performance vs. Efficiency plot shown in Figure 5.2 shows our approach improves the trade-off between precision and computation. As was to be expected, all models perform similarly when saving little computation as they replicate the results achieved by the non-adaptive BERT backbone that performs a similar number of steps. On the other hand, when using limited amounts of computation our model outperforms the alternatives in almost every task (with a single exception). We attribute this advantage in trading off computation and performance to the combined effect of fine-tuning the backbone weights for reduced computation, and using a more robust adaptive mechanism. Indeed, it makes intuitive sense that, as we move away from the 12 step regime for which the underlying static model was trained, more modification of the weights is required. Recall that of all the Dynamic Transformer algorithms only DACT-BERT can modify the Transformer weights because of its full-differentiability.

Importantly, because our model learns to regulate itself, it shows a remarkable stability in the amount of computation saved, as the same values of ponder penalties give rise to similar efficiency outputs. By contrast, DeeBERT proves to be extremely sensitive to the chosen value for the entropy hyper-parameter, exhibiting important fluctuations in both computation and performance indicators for small changes in its value (see RTE in Figure 5.2). This is even more true for the case of PaBEE, since the limited granularity to control the amount of computation through hyper-parameter changes often does not allow to reduce the computation to less than 50%. Compared to DeeBERT, our approach's

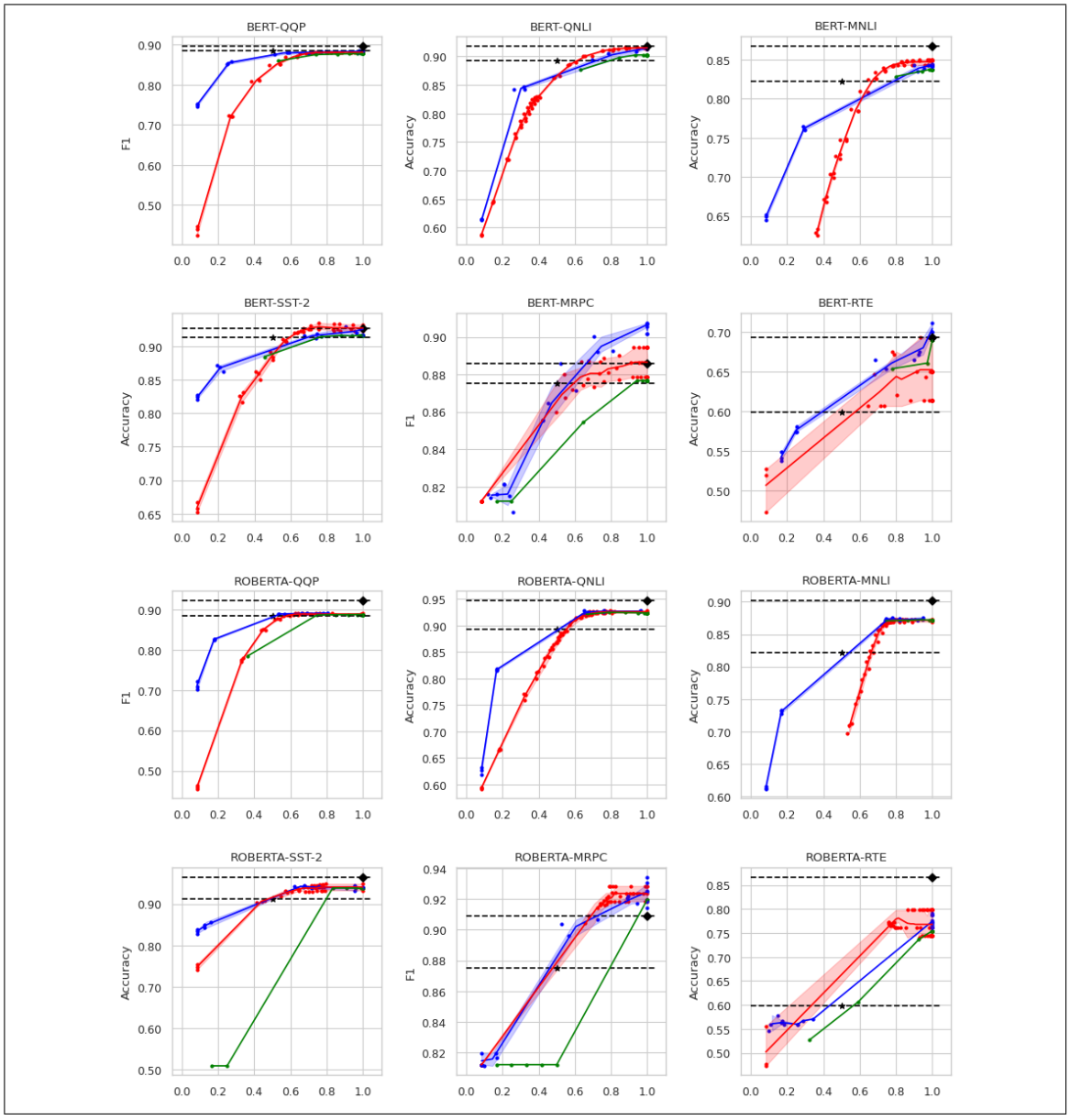
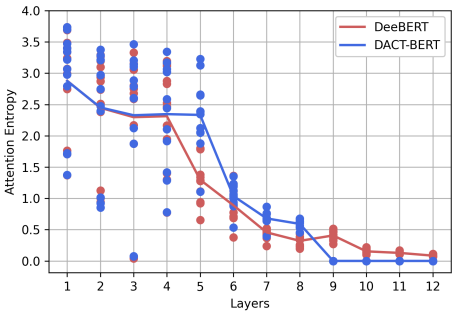
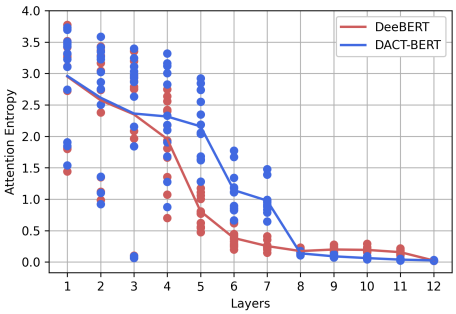


Figure 5.2. Performance vs efficiency trade-offs for BERT-base and RoBERTa-base models using DACT-BERT (blue), DeeBERT (red) and PaBEE (green). DACT-BERT and DeeBERT experiments were repeated three times for each hyper-parameter. Individual runs are shown with colored dots, and the average along with its confidence interval is shown using a band. In all figures the x-axis shows the average number of layers the model chose to unroll as the fraction of those used by the respective static backbone (shown with a black diamond). **DistilBERT's** relative performance is shown at the 50% computation mark using a black star.

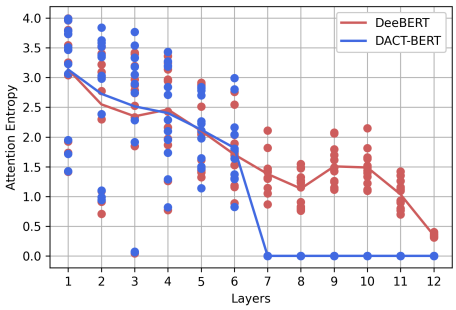
robustness advantage seems to come from training the efficiency mechanism instead of relying on a somewhat arbitrary heuristic for its control.



(a) MRPC



(b) QNLI



(c) RTE

Figure 5.3. Attention entropy distribution per layer in the backbone for DACT-BERT (blue) and DeeBERT (red) for three different GLUE tasks. Each point represent the entropy for one attention head in each layer and the line shows the mean entropy for all the attentions in a given layer.

Additional advantages of our model can be observed in Figure 5.3 which shows the average entropy of each head's attention distribution for DACT-BERT and DeeBERT, following the analysis suggested by the BERTology paper (Rogers et al., 2020). First, it can be easily seen that our approach uses less layers (exact frequencies are shown in Fig. 5.4). That is, even when using on average the same number of layers (as is the case in 5.3a), DACT-BERT completely disregards the outputs from the last blocks, enabling us to prune whole layers without changing the model accuracy for reduced model size. On the one hand, we explain this difference by noting that the entropy will remain high throughout

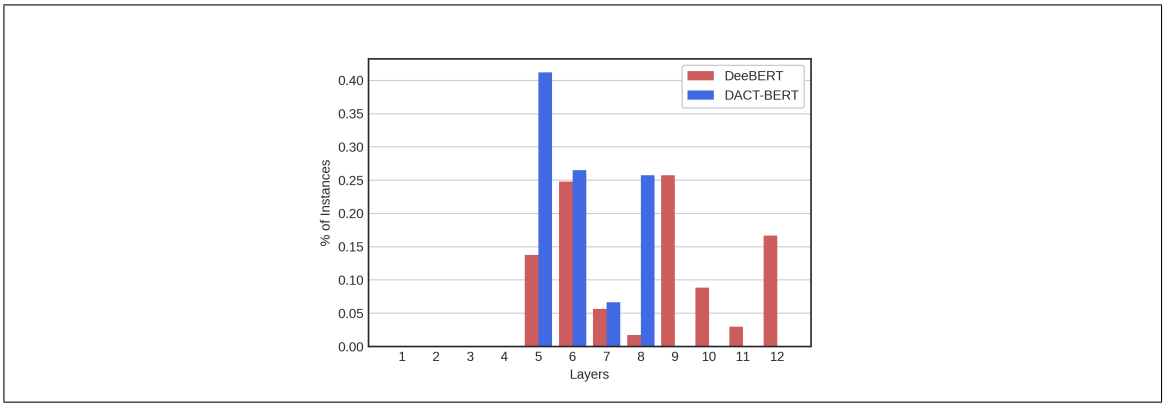


Figure 5.4. Layer Frequencies used by Dynamic Transformers.

the whole model for the case of difficult questions as it will be uncertain about the answer. On the other hand, any layer in DACT-BERT is capable of quitting computation if it believes future layers cannot answer with more certainty than its own (regardless of how certain the model actually is). Figures 5.3c and 5.4 support this hypothesis, showing that difficult tasks in which the model performs worse such as RTE final layers are used more frequently by the entropy based model (DeeBERT).

5.1.3. Interpretability

Our model is more transparent than both static BERT and confidence-threshold dynamic versions in two different ways. First, we increase interpretability by reducing the number of layers and therefore parameters that contribute to an answer. We hypothesize that this will lead to a more transparent model, as previous works (Barceló et al., 2020) proved that interpretability decreases with the number of layers. While this makes intuitive sense and has some backing in recent BERT interpretability works (Rogers et al., 2020; Kovaleva et al., 2019), there is a need to further validate or refute this hypothesis which we leave this to future work and specialists in the field. We do however provide code for calculating the attributions that the input tokens have over the final prediction using the Integrated Gradients technique (Sundararajan et al., 2017) for our model, DeeBERT, and static baselines[2].

[2]This code will be published along with the main model code following publication.

A second source of interpretability provided by DACT-BERT comes from adding a fully differentiable module which manages the number of steps to be taken. Because of this, there is now an additional source of insight that can be explored to understand the inner workings of the model. Namely, attribution techniques can be used to understand which layers and neurons are important for the halting prediction, which gives us further insight on what is the model looking at in order to make the prediction decision. Figure 5.5 illustrates this idea by showing an example of each layer's halting value, with the corresponding attributions of each token to its value. This type of analysis is novel to our work, as the closest analogue in DeeBERT would consist of calculating the attributions of the entropy (which appeared to be meaningless in our experiments), and no parallel exists for PaBEE as the calculation of its patience halting criterion is not differentiable.

5.2. Adaptive Computation for Recurrent Visual Reasoning

5.2.1. Experimental Setup

The MAC network (D. Hudson & Manning, 2018) is a state-of-the-art recurrent architecture that decomposes problems into reasoning steps that, when applied to the CLEVR dataset (Johnson et al., 2016), sets state-of-the-art performance with 98.9% accuracy [3]. In the context of VQA, the full network receives as an input question Q and image I and iterates for a fixed number of times (usually 12) where each step first attends the question, then the image, and finally, it updates an internal memory representation. An important consideration here is that the static model includes a soft gating mechanism, but in the original formulation it cannot be used to reduce computation. We use DACT to enhance the model with the ability to iterate a variable number of times. As Chapter 4 states, this requires each recurrent step should produce its own prediction y_n about the correct answer to Q in addition to modifying the gate to return sigmoidal output $h_n \in [0, 1]$ that represents how uncertain each step about the correctness of its output y_n.

[3] Our PyTorch (Paszke et al., 2017) reimplementation achieves 98.6%.

Figure 5.5. Halting (top) and output (bottom) attribution to each input token for one sample from the MRPC task. The tokens attributions towards the halting value are computed for every halting neuron, after each transformer block. For both, the color intensity shows the degree on how much each token contributes to the final output. Green is used for tokens that have positive attribution to the output, while red means on tokens with negative attribution to the halt or output.

We again train our DACT-MAC in two stages to help convergence and speed up training, and replicate this training regime for both of the baselines. First we pre-train a variant of MAC on CLEVR without using any gating or self-attention for ten epochs (with all hyper-parameters set to their defaults), and then reset all optimizers and train three main variants starting from the saved weights. As our first baseline, we add the soft gate to the static MAC and train more, slightly improving the results. Second, we also implement

and train several versions that use the ACT algorithm (Graves, 2016) with different ponder costs. Finally, we do the same with DACT, loading the pre-trained weights and adding the necessary gating algorithm before training again. All variants are trained for another additional 30 epochs, saving the weights with the highest associated accuracy on the validation set. It is noteworthy that all models have the same number of weights since we reuse the gating mechanism already present in MAC for both dynamic variants.

5.2.2. Results

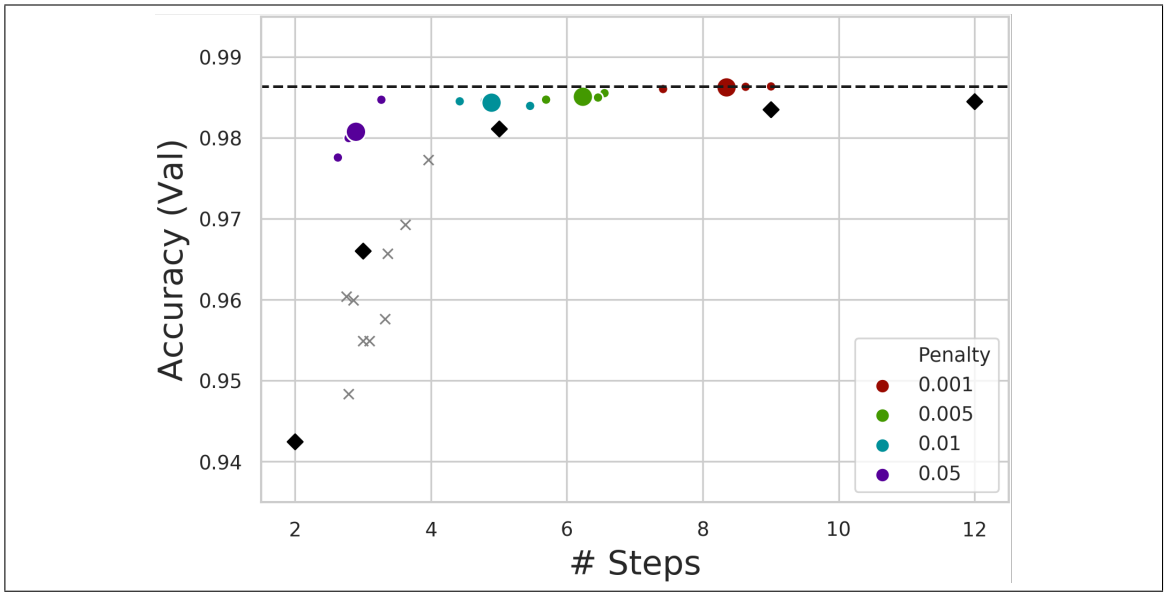


Figure 5.6. Scatterplot showing the relationships between computation (measured in average steps, horizontal), and precision (measured in accuracy, vertical)for each model, where every experiment was repeated three times. The results obtained with DACT are shown in color, with individual runs represented as small circles while the averages for each penalty are shown as larger ones. The averaged results for ACT are shown as gray *X*es. No color is used as the value for the ponder cost did not impact the number of steps. The diamonds show the average accuracy obtained by MAC at different network lengths, while the dotted line represents the accuracy of the best performing 12 step MAC.

As one of the main goals of adaptive computation is to maximize performance at lower computational cost, we evaluate each model's accuracy with respect to the average number of steps taken to reach the best score. As illustrated in Figure 5.6, models resulting from the application of DACT to MAC substantially outperform non-adaptive versions of MAC with similar computation cost in the CLEVR dataset [4]. Additionally, in our experiments, DACT trained with a ponder cost of $\lambda = 1 \times 10^{-3}$ repeatedly obtains an accuracy comparable to the best achieved by any MAC and, on average, surpasses all tested alternatives. This apparent contradiction (obtaining better results with less computation) can be explained by considering that DACT-augmented-MACs have the same representational capacity as regular MACs, but can choose to reduce computation when needed.

The same results also show that, when provided with sufficient resources, MAC increases its performance reducing the gap with respect to DACT versions. This tendency, however, does not hold beyond 12 iterations, as shown also in the original MAC paper (D. Hudson & Manning, 2018). We train a 15 step MAC with gating using the same training scheme, and the results are worse than those from its 12 step counterpart, revealing the inadequacy of the original gating mechanism. In contrast, DACT-enabled-MACs with the maximum amount of steps set to 15 can be fine-tuned from existing 12 step models to obtain the best results of any model tested at 98.72% accuracy. In addition to improving performance, these results prove that using our algorithm on MACs makes them more robust to increments in the value of the maximum number of steps.

On the other hand, models trained with the existing algorithm (ACT) are unsuccessful in surpassing the accuracy of computationally equivalent MACs. In particular, DACT responds as expected to variations in the ponder cost, adapting its computation accordingly, however, ACT proves to be insensitive to the ponder cost. As an example, a variant of ACT without ponder cost ($\lambda = 0.0$) performs 3.2 steps on average and obtains an accuracy of 95.8%. Furthermore, repeating the same experiment with identical hyper-parameters may

[4]Lower computation MAC variants are trained by fixing the number of steps to the closest integer.

result in the model converging to dramatically different computation-to-performance equilibriums.

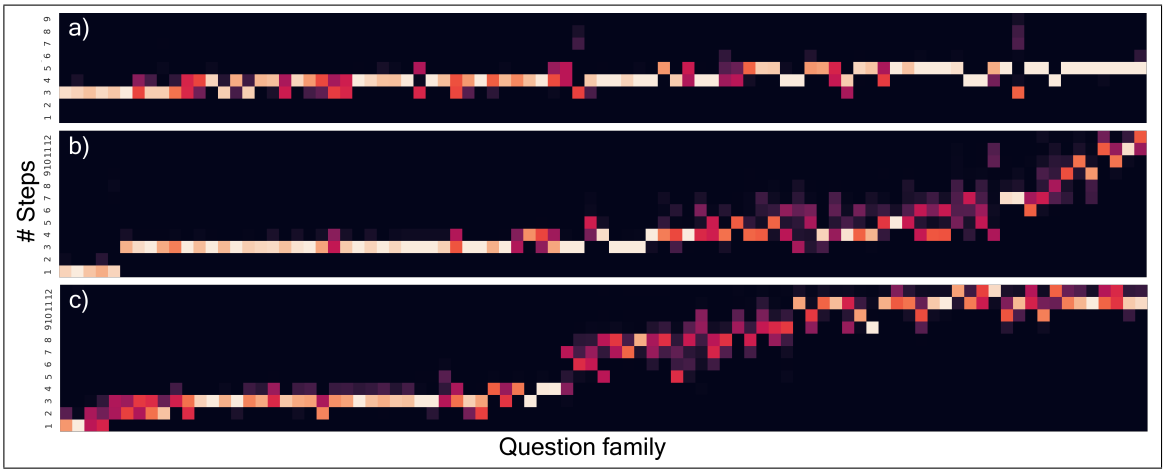


Figure 5.7. Questions in CLEVR are synthetically generated following templates, for example, by replacing <C> and <M> with a color and material in the template *"How many <C> <M> things are there?"*. Accordingly, adding adaptability to the model does not increase performance but rather, similar complexity to solve. The figure shows the average amount of computation used by three models for each question family, sorted by the average number of steps used by the respective model. The first image (a) illustrates how ACT fails to learn how to answer the most straightforward questions in less than three steps, or the hardest in more than five [5]. Below it, b) shows the results for a variant of DACT that averages approximately the same number of steps but uses more of the available spectrum, significantly improving model performance. The last image shows a variant of DACT, which uses 50% more reasoning steps on average and thus achieves even better performance.

We also evaluate how well the model adapts to variations in question complexity, since the rationale behind adapting the number of steps is to enable the models to allocate more computation to complex questions. As expected, DACT iterates fewer times for easy questions and more times when the input question is complex, improving model performance at no additional computational cost compared to static MACs that use a comparable amount of computation. In Figure 5.7, questions are clustered by family type which translates to

[5]This ACT variant was cherry-picked as it achieved the highest accuracy while also doing the maximum amount of steps observed for ACT.

groups that require similar step sequences to solve and therefore are of similar complexity (the figure is further explained in Appendix B, where we include examples for each family). This figure shows a remarkable correlation between computation and question complexity, despite not including any type of supervision about these factors. We take this to mean that the model learns to recognize types of questions and allocates computation accordingly.

Finally, in order to evaluate the generality of the suggested approach to real data, we evaluate the combined DACT-MAC architecture on the more diverse images and questions present in the GQA dataset (D. A. Hudson & Manning, 2019a). We start by again pre-training a non-gated MAC (4 steps, 5 epochs) and then fine-tuning ACT, DACT and gated MAC variants for another 15 epochs. The results shown in Table 5.1 show that DACT is effective in reducing the number of steps needed while maintaining most of the performance of the architecture that always iterates the maximum number of times (four steps). However, we found in our experiments that for GQA the chosen architecture (MAC) doesn't benefit from iterating more than two steps, and even then the advantage gained over its non recurrent single-step version is marginal. Accordingly, adding adaptability to the model does not increase accuracy but rather results in a small but measurable reduction in performance.

Regardless of the above, the experimental results highlight the advantages of our algorithm with respect to ACT, showing once again that DACT obtains better results for the same number of steps. Additionally, while our method continues to adapt computation in a coherent manner to the *time penalties*, ACT remains mostly irresponsive to the values these take. Furthermore, the high correlations between computation and question type are also present for the GQA dataset as 5.8 shows, revealing once more that DACT learnt to meaningfully adapt complexity without supervision.

Method	Ponder Cost	Steps	Accuracy
MAC+Gate	NA	2	77.51
	NA	3	77.52
	NA	4	77.52
	NA	5	77.36
ACT	1×10^{-2}	1.99	77.17
	1×10^{-3}	2.26	77.04
	1×10^{-4}	2.31	77.21
	0	2.15	77.20
DACT	5×10^{-2}	1.63	77.23
	1×10^{-2}	2.77	77.26
	5×10^{-3}	3.05	77.35
	1×10^{-3}	3.69	77.31

Table 5.1. Our proposed method (DACT) achieved better accuracy than existing adaptive algorithms on the GQA *test-dev* set, while also adapting computation coherently to the values taken by the ponder cost hyperparameter. However, the task did not benefit from increased computation, so all adaptive models incur in a small metric loss compared to non-adaptive variants.

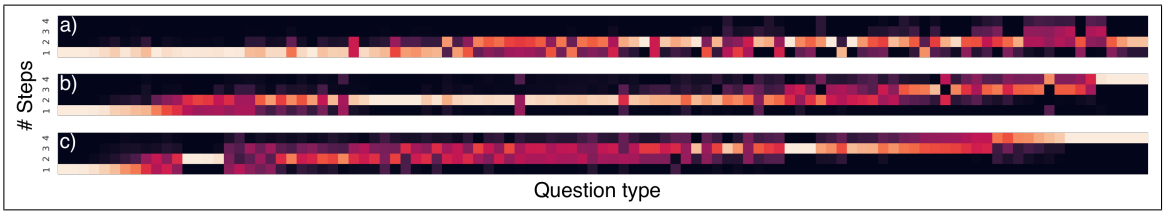


Figure 5.8. The figure shows the the distribution of the number of steps used by DACT for each one of the 105 different *question types* in the GQA dataset. In order from top (a) to bottom (c) we show how decreasing the *time penalty* (5×10^{-2}, 1×10^{-2}, 5×10^{-3} for a,b, c respectively) results in increased total computation.

5.2.3. Interpretability

As in previous works (Johnson et al., 2017; D. Hudson & Manning, 2018), we also analyze the attention maps provided by the model. In particular, we examine both the linguistic and visual attentions generated at each step. As previous works, we also raise the question of whether our proposed architecture can improve interpretability. Figure 5.9

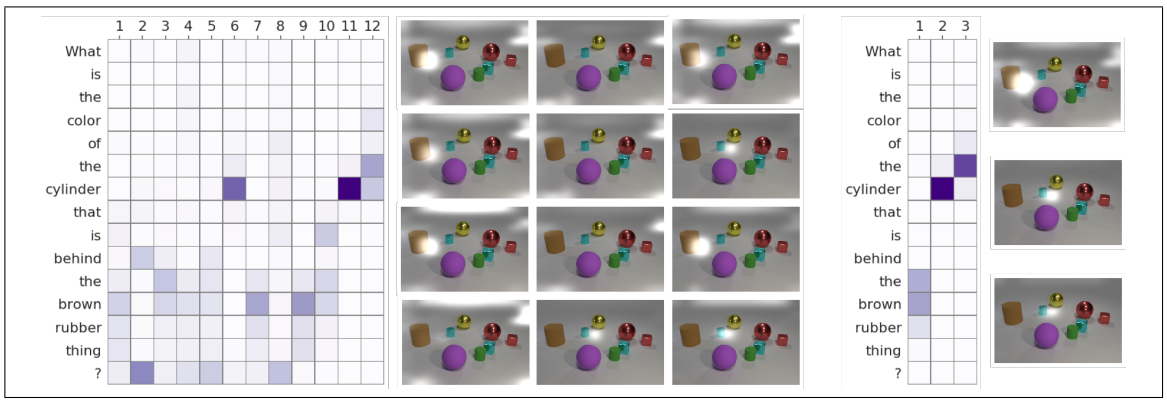


Figure 5.9. Linguistic and visual attention maps for both the standard MAC architecture (left) and our DACT enabled variant trained with $\tau = 5 \times 10^{-3}$ (right). Besides the obvious and substantial reduction in the number of steps used to answer, our model also contributes to the overall interpretability of the inference. This is achieved by adding a proxy of the number of steps taken to the loss function, effectively coercing the model into only using fewer (and therefore more likely to be semantically strong) steps. The question attentions above show that the last two steps are similar for both models, but that only one of the other ten steps used by MAC was necessary.

shows examples of the attention maps generated by the 12 step MAC. Since the MAC architecture only considers the last state in memory for the final classification, the final controls tend to be significant. Indeed, our test indicates that the last few execution steps generate similar attention maps to those produced by our adaptive variant. However, as Figure 5.7 shows, very few queries need all 12 steps of computation, so most of the steps execute either repetitions of other operations, or are just padding (*e.g.* attending punctuation).

The above stands in contrast to our DACT enabled variant, which in practice provides a *free lunch* by maintaining the performance while increasing interpretability without (in the case of MAC) adding additional parameters. We achieve this by adding the differentiable approximation to the number of steps taken, the *ponder cost* (Eq. 4.3), to the loss function. Consequently, since the model is coerced into only using significant steps, we find that those taken are more likely to be semantically meaningful. Simply stated, our approach

improves the transparency of the model by eliminating steps (and therefore attention maps) that do not contribute to the final answer.

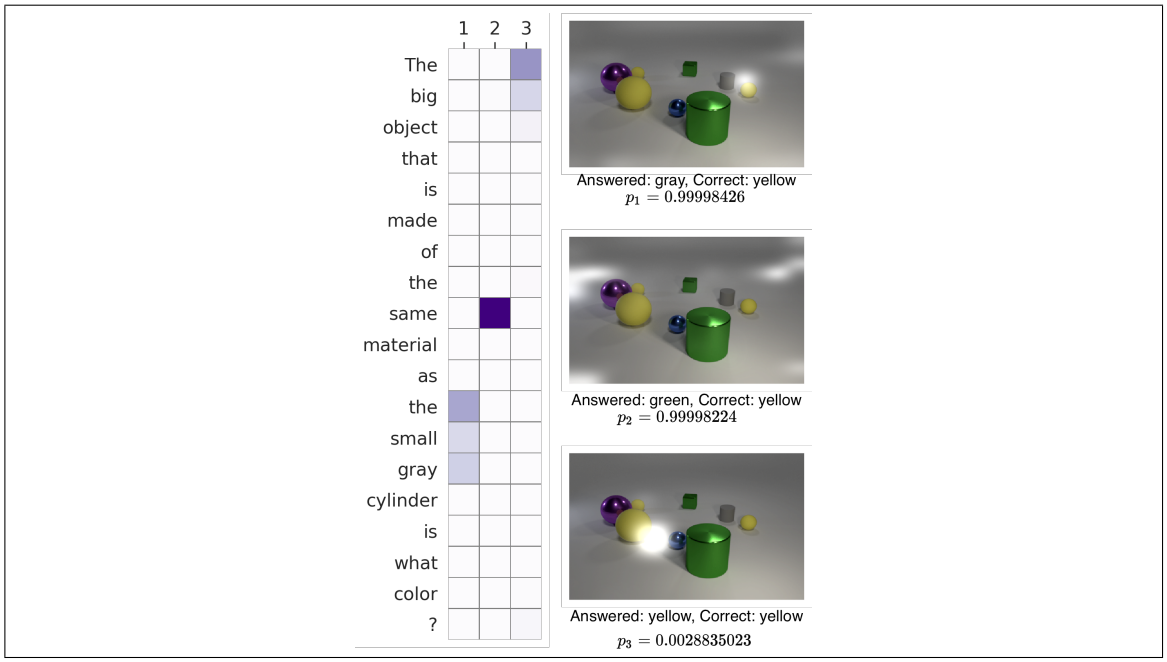


Figure 5.10. Attention maps, intermediate answers, and halting probabilities captured from DACT for the image and question shown. Three steps were needed to arrive at the answer. The first two steps output wrong answers with high uncertainty ($p_n \approx 1$). The last step, however, has identified the relevant object and can thus answer correctly and with confidence.

In addition, the formation of the final output of the model from the sub-outputs enables us to check what the model would answer at each timestep. When analyzed in conjunction with the halting probabilities both yield valuable insights on the internal representation of the model. For instance, in Figure 5.10, the first step has minimal information from the question and image and consequently is very uncertain of the given answer. However, this limited information is enough for the model to identify that the question involves the color of some object, and therefore the answer is the color of the only object it has seen. We expect the increased transparency of the model will assist future studies on explainability and the detection of biases in datasets.

6. CONCLUSIONS AND FUTURE WORK

This thesis suggests that the redundant computation problem that exists in most deep learning models cannot be completely solved using static methods. In particular, we argue that certain intrinsic computational redundancies require the architecture to adapt to the input. Responding to a common limitation of existing methods for dynamic computation, this work proposes the first implementation of a dynamic computation algorithm that is fully differentiable.

We prove the benefits of using a differentiable approach in two distinct situations. First, we find that DACT outperforms state-of-the-art methods that use hand-crafted policies based on early stopping heuristics, providing a increasingly significant advantage w.r.t. existing approaches as the difference in computation with the base model grow. We also show that existing adaptive computation algorithms for RNNs do not function as expected when used along with complex visual reasoning models, while our algorithm offers significant improvements and decreased computation. In both of these scenarios our model responds predictably to changes in the hyper-parameter that controls computation and it finds an appropriate tradeoff between precision and complexity.

We show that our differentiable approach can lead to significant performance gains when replacing existing dynamic approaches, or eliminate intrinsic redundancies when used to augment static models. However, an important limitation of the method presented here is that it is only applicable to early exiting classification problems. We hope this work serves as a catalyst to motivate more research in dynamic architecture models, specifically proposing new differentiable approaches.

Finally, we hope to see more dynamic models that reduce computation time. In particular, we hope to see dynamic and static complexity reduction techniques being combined and applied in both research and production environments leading to more environmentally scalable and interpretable models.

REFERENCES

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016, Jun). Neural module networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Retrieved from http://dx.doi.org/10.1109/CVPR.2016.12 doi: 10.1109/cvpr.2016.12

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate..

Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep equilibrium models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 690–701). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/8358-deep-equilibrium-models.pdf

Barceló, P., Monet, M., Pérez, J., & Subercaseaux, B. (2020). Model interpretability through the lens of computational complexity. In *Advances in neural information processing systems 33*. Curran Associates, Inc.

Bejnordi, B. E., Blankevoort, T., & Welling, M. (2020). Batch-shaping for learning conditional channel gated networks. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=Bke89JBtvB

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, *116*(32), 15849–15854.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., & Joulin, A. (2021). Emerging properties in self-supervised vision transformers. *arXiv preprint*

arXiv:2104.14294.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2018). Universal transformers. *CoRR*, *abs/1807.03819*. Retrieved from http://arxiv.org/abs/1807.03819

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2019). Universal transformers. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=HyzdRiR9Y7

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/N19-1423 doi: 10.18653/v1/N19-1423

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... others (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Eyzaguirre, C., & Soto, A. (2020). Differentiable adaptive computation time for visual reasoning. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 12817–12825).

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D. P., & Salakhutdinov, R. (2016). Spatially adaptive computation time for residual networks. *CoRR*, *abs/1612.02297*. Retrieved from http://arxiv.org/abs/1612.02297

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Ghaeini, R., Fern, X. Z., & Tadepalli, P. (2018). Interpreting recurrent and attention-based neural models: a case study on natural language inference. *arXiv preprint arXiv:1808.03894*.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., & Parikh, D. (2017). Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Conference on computer vision and pattern recognition (cvpr).*

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *CoRR*, *abs/1603.08983*. Retrieved from `http://arxiv.org/abs/1603.08983`

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). *On calibration of modern neural networks.*

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *arXiv preprint arXiv:2102.04906*.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 2961–2969).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition..

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., & Saenko, K. (2017). Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the ieee international conference on computer vision* (pp. 804–813).

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. Q. (2017). Multi-scale dense networks for resource efficient image classification. *arXiv preprint*

arXiv:1703.09844.

Hudson, D., & Manning, C. (2018). Compositional attention networks for machine reasoning. In *ICLR*.

Hudson, D. A., & Manning, C. D. (2019a). Gqa: A new dataset for real-world visual reasoning and compositional question answering. *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hudson, D. A., & Manning, C. D. (2019b). *Learning by abstraction: The neural state machine.*

Jacobs, R., Jordan, M., Nowlan, S., & Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*(1), 79-87.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., ... Liu, Q. (2019). Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., ... Liu, Q. (2020). *Tiny{bert}: Distilling {bert} for natural language understanding.* Retrieved from `https://openreview.net/forum?id=rJx0Q6EFPB`

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. B. (2016). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, *abs/1612.06890*. Retrieved from `http://arxiv.org/abs/1612.06890`

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C., & Girshick, R. (2017). Inferring and executing programs for visual reasoning..

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kaya, Y., Hong, S., & Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning* (pp. 3301–3310).

Kovaleva, O., Romanov, A., Rogers, A., & Rumshisky, A. (2019, November). Revealing the dark secrets of BERT. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 4365–4374). Hong Kong, China: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/D19-1445 doi: 10.18653/v1/D19-1445

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., ... Fei-Fei, L. (2016). Visual genome: Connecting language and vision using crowdsourced dense image annotations.. Retrieved from https://arxiv.org/abs/1602.07332

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., ... others (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, *123*(1), 32–73.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=H1eA7AEtvS

LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems* (pp. 598–605).

Lee, J., Shin, J.-H., & Kim, J.-S. (2017). Interactive visualization and manipulation

of attention-based neural machine translation. In *Proceedings of the 2017 conference on empirical methods in natural language processing: System demonstrations* (pp. 121–126).

Li, L. H., Yatskar, M., Yin, D., Hsieh, C.-J., & Chang, K.-W. (2019). Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*.

Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). Runtime neural pruning. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 2178–2188).

Liu, W., Zhou, P., Wang, Z., Zhao, Z., Deng, H., & Ju, Q. (2020, July). FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 6035–6044). Online: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/2020.acl-main.537` doi: 10.18653/v1/2020.acl-main.537

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Mahoney, M. (2011). *Large text compression benchmark.*

Neumann, M., Stenetorp, P., & Riedel, S. (2016). *Learning to reason with adaptive computation.*

Park, E., Kim, D., Kim, S., Kim, Y.-D., Kim, G., Yoon, S., & Yoo, S. (2015). Big/little deep neural network for ultra low power inference. In *2015 international conference on hardware/software codesign and system synthesis (codes+ isss)* (pp. 124–132).

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). A primer in BERTology: What we know about how BERT works. *ArXiv*, *abs/2002.12327*.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.*

Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2019). Green AI. *CoRR*, *abs/1907.10597*.

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Sun, S., Cheng, Y., Gan, Z., & Liu, J. (2019, November). Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 4323–4332). Hong Kong, China: Association for Computational Linguistics. Retrieved from `https://www.aclweb.org/anthology/D19-1441` doi: 10.18653/v1/D19-1441

Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. In *ICML*.

Teerapittayanon, S., McDanel, B., & Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (icpr)* (pp. 2464–2469).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., ... Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Veit, A., & Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the european conference on computer vision (eccv)* (pp. 3–18).

Verelst, T., & Tuytelaars, T. (2020). Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 2320–2329).

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., & Titov, I. (2019, July). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 5797–5808). Florence, Italy: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P19-1580 doi: 10.18653/v1/P19-1580

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018, November). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP* (pp. 353–355). Brussels, Belgium: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/W18-5446 doi: 10.18653/v1/W18-5446

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., & Gonzalez, J. E. (2018). Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the european conference on computer vision (eccv)* (pp. 409–424).

Wang, Y., Huang, M., Zhu, X., & Zhao, L. (2016). Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 606–615).

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... Rush, A. M. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, *abs/1910.03771*.

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., & Feris, R. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 8817–8826).

Xie, Z., Zhang, Z., Zhu, X., Huang, G., & Lin, S. (2020). Spatially adaptive inference with stochastic feature sampling and interpolation. In *European conference on computer vision* (pp. 531–548).

Xin, J., Tang, R., Lee, J., Yu, Y., & Lin, J. (2020, July). DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 2246–2251). Online: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/2020.acl-main.204 doi: 10.18653/v1/2020.acl-main.204

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhou, H., Lan, J., Liu, R., & Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., & Wei, F. (2020). *Bert loses patience: Fast and robust inference with early exit.*

APPENDIX

A. PROOFS

In this section we prove that our method for building the final answer Y can be interpreted as attending the *intermediate outputs* y_n, with attention weights that follow a valid probability distribution. We include two proofs by induction to show that, for any n, the accumulated answer a_n can be expressed as a weighted sum of all *intermediate outputs* up to the nth step, and that these weights always add up to one.

Proposition. Every accumulated answer a_n can be expressed as a weighted sum of all *intermediate outputs* up to the nth step.

PROOF. Assume α_i exists for each y_i such that every $a_{n-1} = y_{n-1}\alpha_{n-1} + \cdots + y_0\alpha_0$. This is trivial to prove for $n = 1$ as $p_0 = 1$ makes $a_1 = y_1 p_0 + a_0(1 - p_0)$ become $a_1 = y_1$.

$$\begin{aligned} a_n &= y_n p_{n-1} + a_{n-1}(1 - p_{n-1}) \\ &= y_n p_{n-1} + (\alpha_{n-1} y_{n-1} + \cdots + \alpha_0 y_0)(1 - p_{n-1}) \\ &= y_n p_{n-1} + \sum_{i=0}^{n-1} y_i(\alpha_i(1 - p_{n-1})) \end{aligned}$$

□

Proposition. Every accumulated answer a_n can be expressed as a weighted sum of all *intermediate outputs* up to the nth step, and the sum of the weights is equal to one.

PROOF. The base case is again trivial to prove since $p_0 = 1$ when $n = 1$. Using the proof above we define β_i to be the weights used to express a_n as a weighed sum of y_i $\forall i \in [1, n]$.

$$\beta_i = \begin{cases} p_{n-1} & \text{if} \quad i = n \\ \alpha_i(1 - p_{n-1}) & \text{otherwise} \end{cases}$$

Assume α_i exists for each y_i such that every $a_{n-1} = \alpha_{n-1}y_{n-1} + \cdots + \alpha_0 y_0$ and $\sum_{i=0}^{n-1} \alpha_i = 1$.

$$\begin{aligned}
\sum_{i=0}^{n} \beta_i &= p_{n-1} + \sum_{i=0}^{n-1} \alpha_i (1 - p_{n-1}) \\
&= p_{n-1} + \sum_{i=0}^{n-1} \alpha_i - p_{n-1} \sum_{i=0}^{n-1} \alpha_i \\
&= p_{n-1} + 1 - p_{n-1} \\
&= 1
\end{aligned}$$

□

B. AVERAGE COMPUTATION PER QUESTION TYPE

B.1. CLEVR Question Families

For any given synthetic image in the CLEVR dataset (Johnson et al., 2016), a series of queries are generated by chaining a sequence of modular operations such as *count*, *filter*, *compare*, *etc*. These functional programs can then be expressed in natural language in multiple ways, for instance translating $count(filtercolor(red, scene()))$ into *"How many <C> <M> things are there?"*, a translation which is accomplished by instantiating the text templates specific to each program following (Johnson et al., 2016) by replacing *<Z>, <C>, <M>, and <S>* with the size, color, material and/or shape of objects present in the image. As a result, questions with the same functional program can be clustered together into question families that share a similar complexity. Figure B.1 includes a text template for each of the question families present in CLEVR, sorted by the average number of steps used for validation questions belonging to the specific family. Note that families with fewer supporting objects are more likely to be answered in less steps, and finding the number of objects that possess a pair of qualities (*[both]*) is regarded as generally easier than finding those that possess *[either]*.

B.2. GQA Question Types

In the case of the GQA dataset (D. A. Hudson & Manning, 2019a) natural language questions are generated for each image using the image-scene graph pairs present in Visual Genome (Krishna et al., 2016). Questions that are generated such that the functional program needed to answer them are similar are said to belong to the same *question family*. Figure B.2 shows the average number of steps used for each family in ascending order (less computation first).

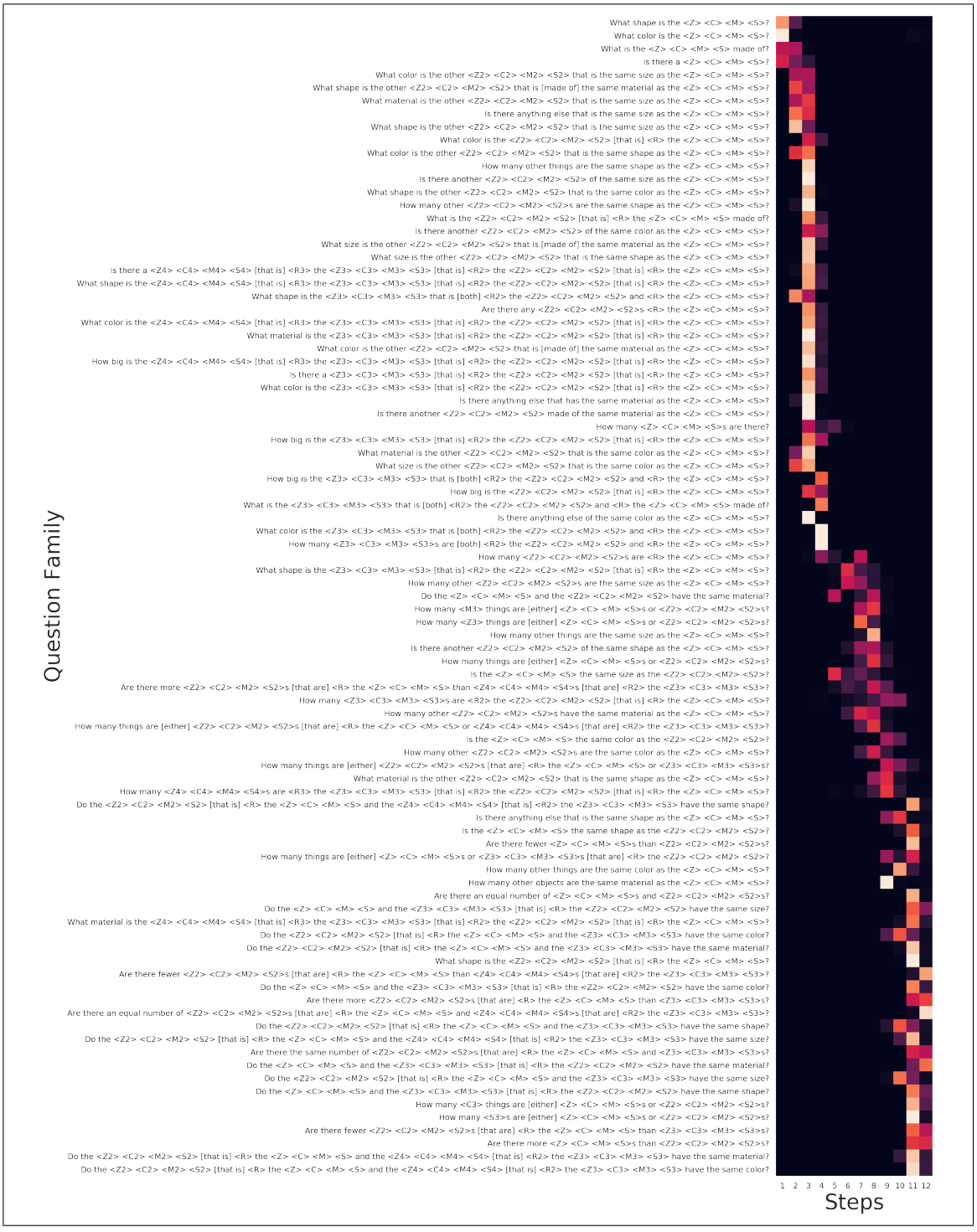


Figure B.1. Average number of steps used by DACT-MAC ($\lambda = 5 \times 10^{-3}$) for each of the question families present in CLEVR, along with one template for each family to typify the whole group.

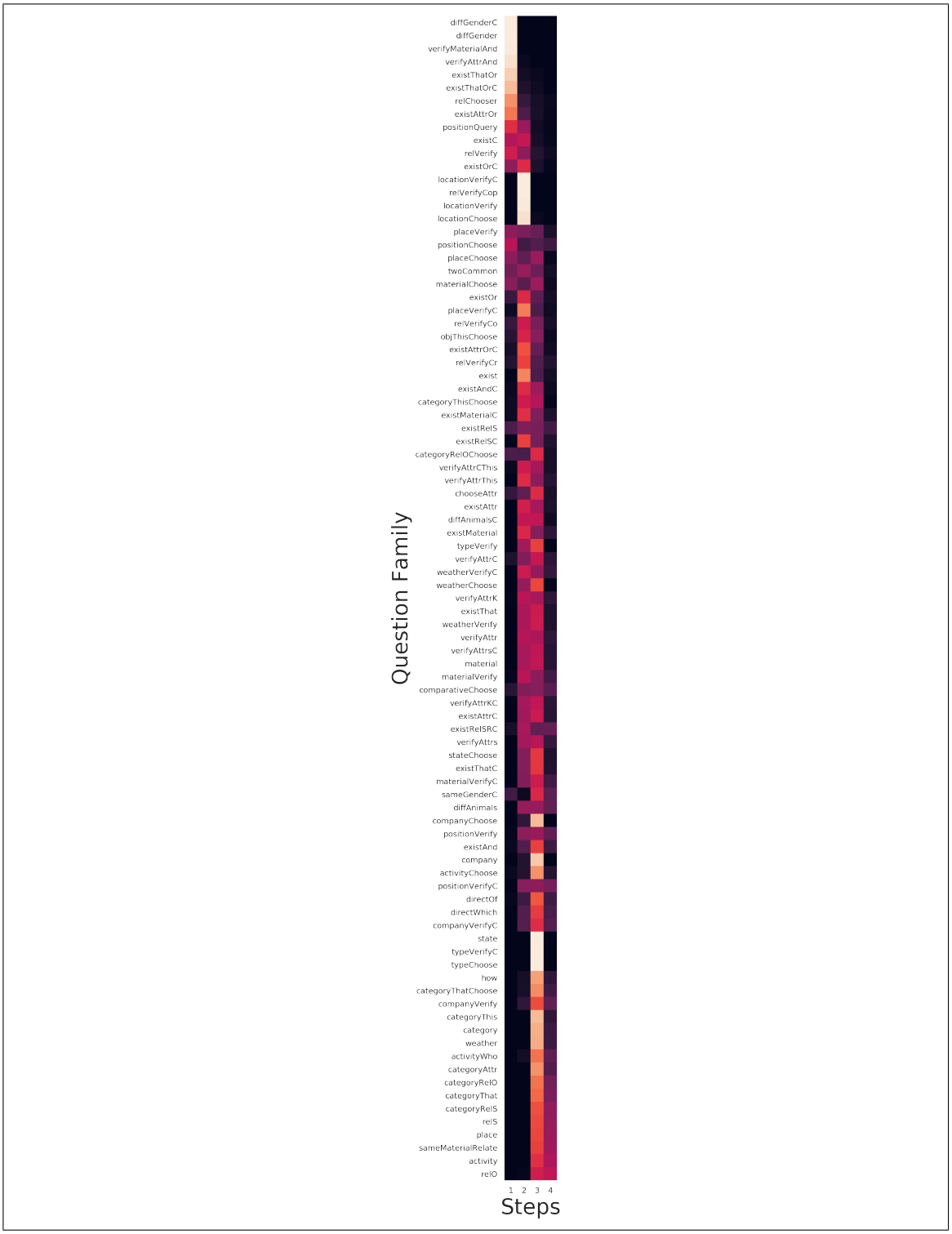


Figure B.2. Average number of steps used by DACT-MAC ($\lambda = 5 \times 10^{-3}$) for each of the question types in GQA, along with the type identifier.