Depth-Wise Attention (DWAtt):
A Layer Fusion Method for Data-Efficient Classification

Muhammad ElNokrashy[*♣], **Badr AlKhamissi**[*♠†], **Mona Diab**[♦†]

[♣]Microsoft Egypt, [♠]EPFL, [♦]CMU

[♣]muelnokr@microsoft.com, [♠]badr.alkhamissi@epfl.ch, [♦]mdiab@andrew.cmu.edu

Abstract

Language Models pretrained on large textual data have been shown to encode different types of knowledge simultaneously. Traditionally, only the features from the last layer are used when adapting to new tasks or data. We put forward that, when using or finetuning deep pretrained models, intermediate layer features that may be relevant to the downstream task are buried too deep to be used efficiently in terms of needed samples or steps. To test this, we propose a new layer fusion method: Depth-Wise Attention (DWAtt), to help re-surface signals from non-final layers. We compare DWAtt to a basic concatenation-based layer fusion method (Concat), and compare both to a deeper model baseline—all kept within a similar parameter budget. Our findings show that DWAtt and Concat are more step- and sample-efficient than the baseline, especially in the few-shot setting. DWAtt outperforms Concat on larger data sizes. On CoNLL-03 NER, layer fusion shows $3.68 - 9.73\%$ F1 gain at different few-shot sizes. The layer fusion models presented significantly outperform the baseline in various training scenarios with different data sizes, architectures, and training constraints.

1. Introduction

The Transformer architecture (Vaswani et al., 2017) and variants (Fan et al., 2020) have become a mainstream, reliable choice for a wide range of Natural Language Processing (**NLP**) tasks, such as sentence classification, token labeling, retrieval, and question answering (Wang et al., 2018a, 2019a). This is in part due to architectural properties that enable enhanced parallelization and better modeling of long-range dependencies. Several models have since surfaced, setting new records on different benchmarks (Devlin et al., 2019; Lewis et al., 2020; Brown et al., 2020). In the vanilla architecture, a stack of Transformer blocks are applied sequentially to refine the representation of an input sequence, which is then fed to a task-specific module, such as a classifier head.

Recent works have shown that hidden representations from intermediate layers may benefit downstream tasks (Wallat et al., 2021). Other works have tested the fusion of hidden representations in tasks such as sequence-to-sequence machine translation (Shen et al., 2018; Liu et al., 2020, 2021). See also speech modeling works (Li et al., 2018).

In this work, we propose a method to combine the hidden representations of encoder layers to more easily utilize the full model. Moreover, we further
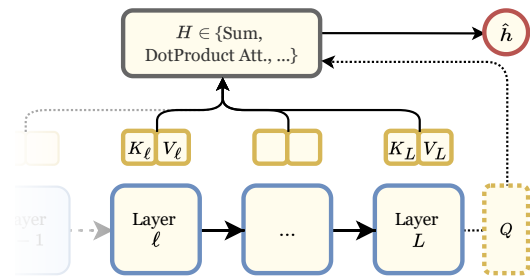

Figure 1: Basic architecture: The Mixer H can be a Sum of Affine Transformations, a Dot-Product Attention module, etc. Different variants may define K_i, V_i, and Q (key, value, and query transforms) and utilize them differently.

investigate a simpler alternative as a compelling baseline. This work expands on *Vertical Attention*, introduced in AlKhamissi et al. (2021), by providing a more detailed specification and an extensive evaluation and analysis.

2. Motivation & Hypothesis

A common goal when designing deeper networks is enhancing their ability to represent deeper chains of abstraction in different data domains and training objectives. As the model is tuned for a specific task, seemingly *unneeded* information is ignored. More general patterns that could benefit different downstream tasks become less likely to pass through intermediate layers to the final representation.

Some base models (like Large Language Models) posses properties which may implicitly mitigate

this effect—by leveraging their large parameter capacity and increased width. The additional parameters enable memorizing more subtleties or idiosyncrasies of the training data, while the increased model width allows for a less compact final representation, which may let through patterns not *directly* used by the general language modeling objective of choice. Some tasks may need such low-level knowledge across a large example space (Liu et al., 2021). A similar problem is the strength and clarity of gradients into earlier intermediate layers. Methods to alleviate this include skip connections (He et al., 2016), and alternatives to full back-propagation (Nøkland, 2016).

We propose an add-on module that can augment any pretrained deep sequence model by combining the representations of its intermediate layers to adapt better to novel tasks. To investigate our proposed modules, we experiment with the following tasks: *(a)* Named Entity Recognition (**NER**) in the few-shot setting on the `CoNLL-03` and `WikiAnn` datasets, and *(b)* Masked Language Modeling (**MLM**) on a small text dataset (`WikiText-2`). Two settings are considered: Finetuning (**FT**) and Feature Extraction (**FE**).

Adapting to New Tasks. For many pretraining projects, adaptability to novel language tasks is a crucial performance signal. it allows one to exploit general language understanding from self-supervised tasks like MLM on large datasets.

Useful signals include: *(a)* Performance versus a model trained on task-specific data only, *(b)* time to convergence, and *(c)* size of the needed task data to achieve the desired performance.

One could finetune the whole model (possibly infeasible), or use it as a feature extractor and train only a task-specific module. A commonly researched alternative is finding or introducing a subset of parameters specifically for finetuning (Houlsby et al., 2019; Ben-Zaken et al., 2022). We test our proposed method and an alternative in different adaptation settings with multiple n-shot training sizes, max training steps, and tiers of availability of language data during pretraining.

3. Tasks, Datasets, and Raised Questions

On many common benchmarks, state-of-the-art performance is often near saturation. We consider some useful synthesizable variants of the benchmarks that test particular training settings or aspects of performance. Our experiments aim to test the following aspects: *adaptability* (finetuning (FT) versus feature extraction (FE)), *sample efficiency* (few-shot training), *training step efficiency* (time to convergence), and *effect of model depth*.
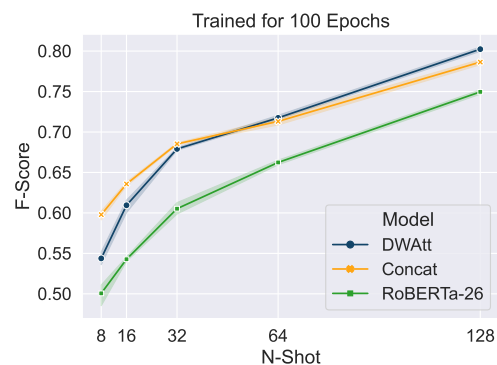

Figure 2: F1-Score on the `CoNLL-03` dev set after training for 100 epochs on few-shot training datasets with *N-Shot* samples per class sampled uniformly from the full training set. Both `DWAtt` and `Concat` improve on the additional layers baseline R_{26}. At 128 samples per class, `DWAtt` improves on both `Concat` and the enhanced baseline by 1.6% and 5.28% absolute, respectively.

3.1. Few-Shot Adaptation on NER

Dataset	Labels	Resource	Train	Dev
`CoNLL-03`	4	High	14k	3250
`WikiAnn`	3	High	20k	10k
		Mid	1k-5k	1k
		Low	100	100

Table 1: Train and Dev subset sizes in sentence count. For `WikiAnn`, we list the average size of a language in the corresponding resource tier, determined by train size.

NER, a moderately complex task, evaluates performance at a token-level, and reasonably can benefit from intermediate features. See section 8.1.

CoNLL-2003 The `CoNLL-03` dataset (Tjong Kim Sang and De Meulder, 2003) provides a mainstream NER benchmark in the English language. For few-shot trials, we sample training points uniformly at random from the full training set.

WikiAnn The `WikiAnn` dataset (Rahimi et al., 2019) is a multilingual NER benchmark which we use to test the effect of resource availability in the pretraining phase. For few-shot experiments, we sample a fixed training set of size $N = 100$ and train for each language separately. We also compare performance on the English subset between RoBERTa and the multilingual XLM-RoBERTa.

3.2. Masked Language Modeling

The `WikiText-2` dataset ([Merity et al., 2017](#)) is a subset of the English Wikipedia developed for use in long-term dependency language modeling. It contains the first 2 million words of a dump of the English Wikipedia at the time of creation. We report the perplexity on the dev and test sets. We hypothesize that retaining better pretraining performance points to increased reliability. See Section 8.1.

4. Models

Let L denote a deep network's layer stack. Then H[1] is a learned function that mixes the layers' intermediate representations $\{\mathbf{z}_n \mid n \in |L|\}$, and $\mathbf{h}$ is the final representation.

$$\mathbf{h} = H\left(\cdots, \{(n, \mathbf{z}_n) \mid n \in |L|\}\right) \quad (1)$$

The H module is a function of the layer indices n and the corresponding representation vectors $\mathbf{z}_n$, but can take other signals, like $\mathbf{z}_L \mapsto \mathbf{q}$ for query. See diagram in Figure 1. We consider the following layer fusion models.

Layer Concatenation. A sum of linear transforms. Note that this is equivalent to concatenating all $\{\mathbf{z}_n\}$ then transforming the concatenation into the model width $d_{\mathbf{z}}$.

$$\mathbf{h} = \sum_n \mathbf{W}_n(\mathbf{z}_n) \quad (2)$$

Depth-Wise Attention. DWAtt uses dot-product attention with keys $\mathbf{k}_n$, values $\mathbf{v}_n$, and query $\mathbf{q}$.

$$\mathbf{k}_n = \mathsf{PE}(n) \quad (3)$$

$$\mathbf{v}_n = \mathsf{LN}_n\left(f_n^V(\mathbf{z}_n)\right) \quad (4)$$

$$\mathbf{q} = 1 + \mathsf{elu}\left(\mathbf{z}_L + f^Q(\mathbf{z}_L)\right) \quad (5)$$

Where $\mathsf{PE}(n)$ is a learned positional embedding vector for layers $\{n\}$. LN is LayerNorm ([Ba et al., 2016](#)). f^Q, f_n^V are MLPs with a bottleneck at $1/2$ the model width $d_{\mathbf{z}}$. elu is from [Clevert et al. (2015)](#).

$$f(\mathbf{z}) = \mathbf{W} \cdot \mathsf{LN}(\mathsf{gelu}(\mathbf{U}\mathbf{z})) \quad (6)$$

It is then comparable to a single linear layer of size $d_{\mathbf{z}} \times d_{\mathbf{z}}$. See Section 9 for details. Note: the scoring step in Attend reduces to a single, vector-by-static matrix multiplication as the keys $\{\mathbf{k}_n\}$ need no input. Assume a time step t, then let $\mathbf{K} = \{\mathbf{k}_n\}$ and $\mathbf{V} = \{\mathbf{v}_n\}$ be the matrix forms of the keys and values for $n \in |L|$. Then:

$$\mathsf{Score}(\mathbf{q}, \mathbf{K}) = \mathsf{softmax}_n\left(\mathbf{q} \cdot \mathbf{K}^\top\right), \quad (7)$$

$$\mathsf{Attend}(\cdots) = \mathsf{Score}(\mathbf{q}, \mathbf{K}) \cdot \mathbf{V}, \quad (8)$$

$$\mathbf{h} = \mathbf{z}_L + \mathsf{Attend}\left(\mathbf{q}, \{\mathbf{k}_n\}, \{\mathbf{v}_n\}\right). \quad (9)$$

---

[1] In this work, symbols in red have learned parameters.

4.1. Baselines

Name	New Layer	+ $O(\cdot)$	+ #
Base	-	-	-
Base$_{+n}$	$n\times$ Base	$7n\,(d + d^2)$	25.19M
Average	MeanPool	-	-
Concat	Affine	Ld^2	25.18M
DWAtt	DWAtt	$Ld^2 + d^2 + Ld$	26.38M

Table 2: The **size and count of parameters of new layers** in different configurations. Counts are of models building on RoBERTa$_{24}$. `Base`$_{+n}$ (deeper baseline) is chosen to be close in size to `DWAtt` and `Concat`. For example, for base models with 12 layers, only n=1 layers are added; n=2 for 24 layer models. L refers to the depth of the *base* model, while d refers to the model width.

Base Model. The base model (RoBERTa or XLM-RoBERTa in BASE or LARGE sizes) is used as-is. The task module is changed where needed. In **FE** only the task module (or vocabulary module in MLM) is trained.

Extra Transformer Layers. On top of the base model, we add 2 more Transformer layers before the classification head. In **FE** mode, we train only the added layers. We refer to this model as R$_{26}$ (RoBERTa) or XLM-R$_{26}$ (XLM-RoBERTa).

5. Experimentation Setup

5.1. Base Models

Monolingual experiments using `CoNLL-03`, the English subset of `WikiAnn`, and `WikiText-2` all build on RoBERTa (**R**) ([Liu et al., 2019](#)) via the pretrained `roberta-large` model. Multilingual experiments using `WikiAnn` build on XLM-RoBERTa (**XLM-R**) via the pretrained `xlm-roberta-large` model. Experiments using other variants are explicitly specified. Pretrained models are accessed via the HuggingFace package ([Wolf et al., 2019](#)).

5.2. Training

In figures where the epoch count is reported: Each chart shows an experiment with the indicated N_{epochs} max epochs, which starts LR at max then decays it linearly to zero.

Few-shot experiments sample NC points uniformly at random from the full training set. Sampling is not stratified, so $N = 8$ Shot for $C = 4$ classes means 32 points in total sampled without replacement.
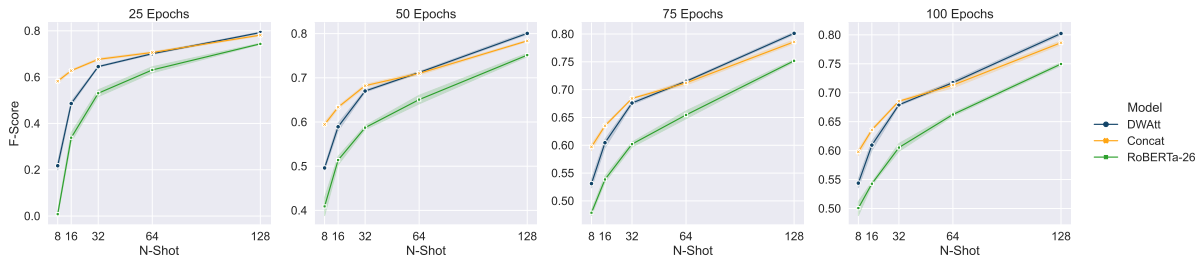
Figure 3: **F1-Score on the CoNLL-03 devset.** All pretrained weights are frozen (**FE**). In each chart from left to right, training is constrained to 25, 50, 75, and 100 max epochs. For each *N-Shot* experiment, NC samples ($C = 4$ classes) are randomly selected and trained on for the entire experiment. The scores are averaged across 5 trials with random initialization of weights and data sampling. We report the best observed dev score from the full training of each experiment and trial.

5.3. Evaluation

NER experiments report Micro-Average F1 using seqeval (Nakayama, 2018). Where applicable, experiments are given 5 trials whose average and confidence interval are reported by `seaborn` (Waskom, 2021). For easier reading, we report scores that lie in $[0, 1]$ as percentages $[0, 100]\%$.

6. Results and Analysis

6.1. CoNLL: Few-Shot Adaptation

Micro-F1 is reported at each *N-Shot* in $\{8, 16, 32, 64, 128\}$. Each model in Figure 3 adds the specified module on top of a pretrained RoBERTa$_{\text{LARGE}}$. Concat (orange) noticeably outperforms R$_{26}$ at all few-shot settings, while DWAtt (blue) outperforms Concat at the higher data sizes.

6.2. Step and Sample Efficiency
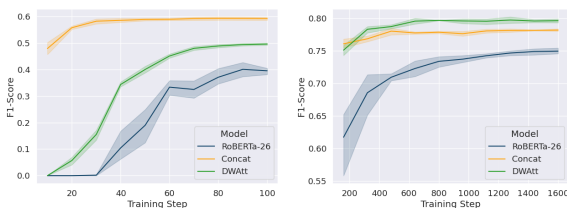
(a) Few-shot at $N = 8$. (b) Few-shot at $N = 128$.

Figure 4: **Training Behavior.** Validation F1-Score across steps in **FE** training on CoNLL-03. Concat generalizes more readily at smaller N compared to DWAtt. At both sizes, layer fusion methods are able to extract more from pretrained models than traditional last-layer fitting.

In **Figure 3**, Concat outperforms noticeably at the lowest end (N=8 at N_{epochs}=25), improving on

R$_{26}$ by $\mathbf{58}\%$—possibly owing to its simpler connectivity and gradient path. While DWAtt's improvement given increased data and training time may be explained by its being more selective due to the attention module. In the largest setup (N=128, N_{epochs}=100; Figure 2), Concat improves on R$_{26}$ by 3.68%, while DWAtt improves on it by $\mathbf{5.28}\%$. R$_{26}$ demonstrates the difficulties of extracting the patterns needed by a downstream task from a feature extractor that has already fit its last layer representation (L=24 for RoBERTa$_{\text{LARGE}}$) for its pretraining task.

In **Figure 4**, we focus on model validation behavior during training with N_{epochs}=50. At N=8, only Concat manages to adapt well, and rapidly, converging in 30% of training time. At N=128, DWAtt has enough data to reach similar performance at a similar pace, and quickly pulls ahead.

We repeat in **Figure 5** the experiments from Figure 3 but on English only from WikiAnn, on R$_{26}$. Yet again Concat performs better with less data, while DWAtt is better at the higher end (N=128, N_{epochs}=100) by $+2.08\%$ and $+0.92\%$ over Concat and R$_{26}$, respectively.

6.3. Feature Extractor Adaptability

Figure 6 shows the effect of finetuning (**FT**) all model parameters for each of the same configurations used in other CoNLL-03 experiments. At $N = 8$, Concat in feature extraction (**FE**) training already gets most of the improvement observable from full FT across the board. At $N = 128$ the effect is stronger. While layer fusion methods help close the gap—with DWAtt giving nearly half the improvement of FT using only FE, at $+5.28\%$—FT still manages overshadow all FE configurations even with only R$_{26}$.

6.4. Scaling by Depth

In **Figure 7**, we compare the same add-on configurations on the BASE and LARGE variants of
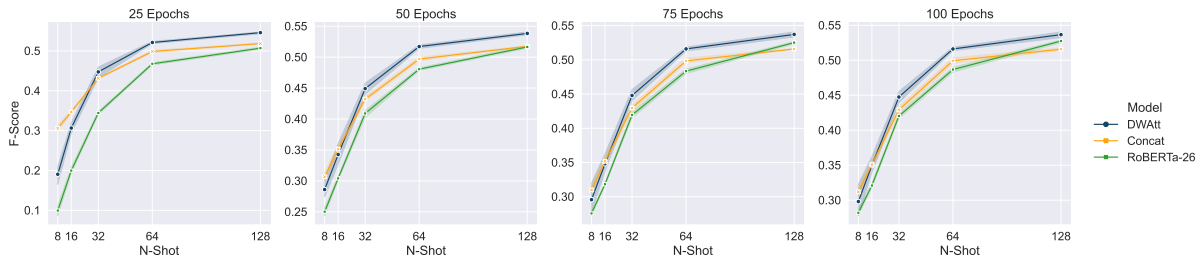
Figure 5: F1-Score on the `WikiAnn` *English* devset on the *RoBERTa*$_{LARGE}$ base model. The methods start behaving similarly at higher training epochs. `DWAtt` leads `Concat` leads R$_{26}$ at medium shot scenarios.
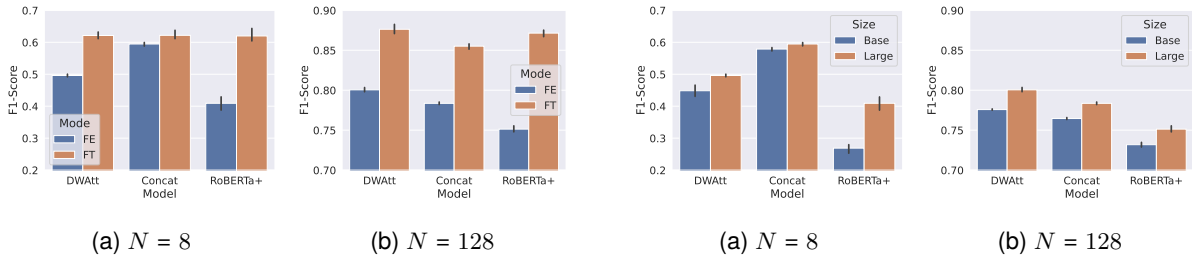

(a) $N = 8$ (b) $N = 128$

Figure 6: **Feature Extraction (*FE*) versus Finetuning (*FT*).** Best validation F1-Score on `CoNLL-03` within $N_{epochs} = 50$ training epochs. Finetuning consistently outperforms alternatives, but `Concat` and `DWAtt` approach its performance even in **FE** training at lower data sizes.

RoBERTa. At $N = 8$, `Concat` is an outlier in its performance gain over the alternatives. In all $N \in \{8, 128\}$, both `Concat` and `DWAtt` match or beat the performance gain observed from `RoBERTa+`[2] when changing from BASE to LARGE.

6.5. `WikiAnn`: Adapting to High, Mid, and Low Resource Languages

Figure 9 shows average performance grouped by the token count of pretraining data according to Conneau et al. (2019). Low-resource languages had 10–100M tokens, Mid-resource languages had 200–300M tokens , while High-resource languages had 2–20G tokens. While there is a slight lead for layer fusion methods in the Low-resource column, the difference does not translate to a similar lead in Mid-resource scores over High-resource.

Compare the scores in both **Figures 5** & **8**. Starting from $N_{epochs}=25$ and only $N=32$ samples, performance using RoBERTa$_{LARGE}$ as the base model beats the corresponding run on XLM-R$_{LARGE}$ at $N_{epochs}=25$ with 100 samples.

(a) $N = 8$ (b) $N = 128$

Figure 7: **BASE versus LARGE Pretrained Models.** Validation F1-Score on `CoNLL-03` on the `DWAtt`, `Concat`, and enhanced (+layers) configurations of RoBERTa BASE and LARGE. *(a)* At $N = 8$, `Concat`$_{BASE}$ shows a clear lead even on `RoBERTa+`$_{BASE}$. *(b)* At $N = 128$, `Concat`$_{BASE}$ outperforms `RoBERTa+`$_{LARGE}$, while `DWAtt`$_{LARGE}$ outperforms `Concat`$_{LARGE}$. *(c)* This also supports the claim for higher data and training efficiency from `Concat` and `DWAtt` in **FE** training compared to traditional last-layer fitting.

Model	Test Perplexity
RoBERTa+	2.98
Concat	3.44
DWAtt	2.91
Base	2.90

Table 3: MLM perplexity on `WikiText-2` test set.

6.6. Masked Language Modeling

We report the test perplexity of the checkpoint with the best validation loss from FT training. `DWAtt` performs closer to the baseline `Base` than `Concat` and R$_{26}$.

7. Related Work

Adaptors. Adaptors are layers introduced into a pretrained model layer stack and finetuned on a specific downstream task, to avoid changing the

---

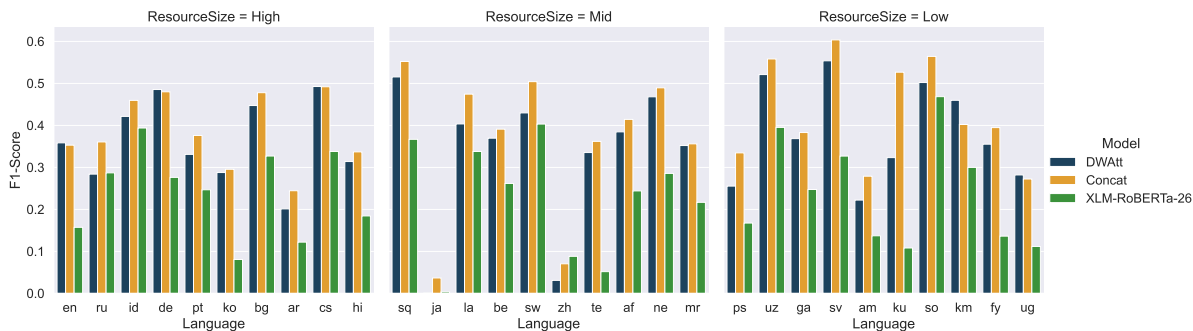[2] `RoBERTa+` BASE is $L=12 + 1$ layers, while LARGE is $L=24+2$.

Figure 8: F1-Score on the `WikiAnn` devset. Each language is trained for $N_{epochs} = 25$ and tested separately. Each model is initialized from XLM-R$_{LARGE}$ weights, augmented with the specified module (DWAtt, Concat, or additional layers), then trained on exactly 100 uniformly-sampled training points for each language. The languages are sorted by token count of pretraining data from Conneau et al. (2019).
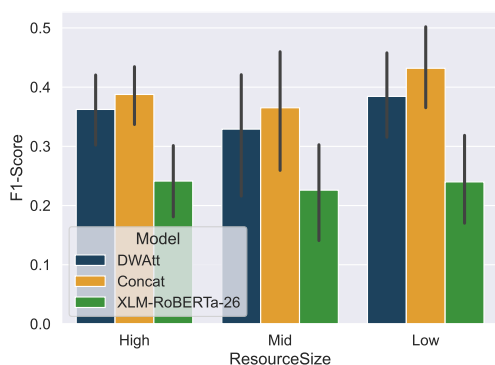

Figure 9: F1-Score on the `WikiAnn` devset. Languages are grouped into three tiers of resource size. See Figure 8 and Table 1 for details.

(larger) model's weights. (Houlsby et al., 2019), among others.

Layer Aggregation (static weights). Bapna et al. (2018) defines, for each decoder layer, a trainable softmax-normalized vector of weights to get the weighted sum of the encoder intermediates. Liu et al. (2021) provides one static, learned vector for each intermediate layer's representation which acts as element-wise scaling. Both works are similar to `Concat` in applying a static, learned mixing transform to all layers. For each layer: The first provides a scalar weight, the second provides a vector for element-wise multiplication, while `Concat` applies a linear transform.

In Shen et al. (2018), the **DenseNMT** is an encoder-decoder NMT architecture densely connected in the style of **DenseNet** (Huang et al., 2016). Each encoder layer takes a concatenation of all previous layer representations. Similarly for decoder layers. See also Wang et al. (2019b). These methods compare structurally to `Concat`, which is applied only once on the full layer stack.

Wang et al. (2018b) generates the weight for an encoder layer via an MLP on the layer's representation, irrespective of the rest of the model: $w_i = f^a(\mathbf{z}_i)$ then $\mathbf{h} = \sum_i (w_i \mathbf{z}_i)$.

Dynamic Layer Mixing. The following methods use signals from the input to change the transformation itself dynamically. See also `DWAtt` which uses a Dot-Product Attention module for comparison. Li et al. (2018) applies an LSTM depth-wise on the intermediate vectors of the stack of LSTM cells applied on input sequences, as expected. We attempted adding an LSTM cell applied depth-wise to the Transformer encoder stack but observed lacking performance. Note that the referenced work utilizes a non-basic cell with peep-hole expressions, and some architecture connectivity that complicates experiments in the Feature Extraction context. Dou et al. (2019) utilizes a dynamically-weighted routing mechanism to mix transformations of each intermediate representation, then concatenate all such.

8. Discussion

8.1. Notes on Selected Evaluations

NER: A Token-level Downstream Task The NER task is selected to provide a more complex performance signal than sentence classification (which was the case in AlKhamissi et al., 2021). See Section 8.3 where we hypothesize on the cases that would benefit from DWAtt. Briefly, tasks which look at individual tokens deeply and do not need increased spatial abstraction (for example because of increased length) may be more likely to benefit from methods like DWAtt, especially if full fine-tuning is not available. Further, prior work such as Wallat et al. (2021) suggests that there can be useful information in intermediate layers, and specifically probes for it by a token-level task.

MLM: Pretraining objective We build on pre-trained large models in order to exploit a more general linguistic and logical capability than could be expected from training on a downstream task dataset much smaller than the smallest of unsupervised LM data collections. We believe that fine-tuning must maintain this more general objective of the base model in order to make the claim that task performance generalizes outside the specific sub-domain of the test set. If fine-tuning results in great downstream results but equally great pretraining objective regression (e.g. MLM), we believe that method should be regarded as less reliable and desirable than one which maintains the pretraining objective scores.

8.2. Levels of Abstraction

For non-recurrent deep neural networks, there exists a functional limit to the depth of abstraction obtainable, which is proportional to the depth of the network. Abstraction here refers to the depth of composed rules that a model may learn to apply on low-level stimulus, such as pixels or tokens. For feed-forward models like an MLP or transformers, this corresponds to the depth in terms of stacked nonlinear layers. As an example, to handle program-like systematic inputs, depth-recurrent and memory-augmented architecture were utilized in works such as Dehghani et al. (2019); Graves et al. (2014). For traditional Transformer models, like RoBERTa$_{LARGE}$, we can say that the limit is proportional to the number of layers in the network (e.g. $O(|L|)$; $|L| = 24$).

Hiding Within Scale. In practice, this is seldom an obvious problem because large networks would have enough width-wise parameter capacity to directly encode "intuition", i.e. shortcuts to knowledge and abstraction. They may do so by tying the low-level representation of some inputs to intermediate signals for the high-level concepts they tend to manifest. As an artificial example: A small, shallow model for classifying sentiment may tie a token such as `scary` to become a strong signal for negative sentiment, say in a movie review setting. Sensible in the domain of kids movies; but may in fact signify a positive sentiment instead when observed in reviews for horror movies. The key point is that it is an early shortcut, not whether it is accurate.

Intuition as Shortcuts From Raw Input. Thus, earlier layers can encode information at a higher level than $|L|$-steps of abstraction would suggest. This would be needed, and expected, for models of a reasonable depth and width to be able to satisfy the feature extraction needs of the *pretraining* task

at the last layer. Should a *downstream* task require a different signal from what the *pretraining* task exposed, it may have to *shift* a large subset of the weights to relearn or resurface that information from the shallower levels.

Proposition. The methods we've discussed may enable downstream tasks to query the model for hidden but useful information. For the downstream task to make use of such features, it would likely need to transform them further. By applying the 2-layer MLPs f_n^V on these intermediate features (in `DWAtt`), and a linear transform in `Concat`, the task can extract a more useful representation from each level/layer.

8.3. Modeling Capacity

The two models presented and highlighted—`DWAtt` and `Concat`—are aggregate views of the features of all intermediate Transformer layers $\{\mathbf{z}_n \mid n \in |L|\}$ (see Section 4). `DWAtt`'s added module requires access to the last layer's $\mathbf{z}_L$ to form the query, while `Concat` does not. Neither method makes any use of external signals such as, for example, a task embedding vector.

These three points together present an underlying property of the modeling capacity of `DWAtt` and `Concat` when *fully-tuned*: *The depth-wise layer mixing arrives at a model which is at most as expressive as the underlying sequence-modeler*. See Figure 6 where layer fusion under FE nears but does not exceed FT, while all methods are similar under FT.

Modeling Dimensions. `DWAtt` and `Concat` operate *depth-wise* over a sequence-modeling model. By that very nature, it may not be the best option when the task at hand requires increased or improved *spatial* abstraction—the ability to learn connections on a spatial axis (e.g. between tokens in sequences in a text Transformer). Then, adding extra Transformer layers or full finetuning may be better options.

9. Design Details

Model	Param	Value
All$_{\textbf{Large}}$	Transformer Layers	24
All	Learning Rate	1e-5
DWAtt	γ^Q, query bottleneck	0.5
DWAtt	γ^V, values bottleneck	0.5
DWAtt	$d_{\textbf{pos}}$, keys latent	24

Table 4: Architecture Parameters

9.1. Training

All trainings use the AdamW (Loshchilov and Hutter, 2017) optimizer with a linear decay learning rate (LR) schedule. Training on `WikiText-2` uses a batch size of 8 samples, and a max LR of 5×10^{-5}. Training on `CoNLL-03` and `WikiAnn` uses a batch size of 16 and a max LR of 5×10^{-5}.

9.2. Layer Index Embedding

$\mathbf{k}_n^{\text{pos}} \in \mathcal{R}^{d\,\textbf{pos}} \sim \mathcal{U}(0, 1)$ are static positional embedding vectors of the layer index n. $\mathbf{W}^K$ is a learned affine transform.

$$\mathbf{k}_n = \text{PE}(n) = \mathbf{W}^K \mathbf{k}_n^{\text{pos}} \qquad (10)$$

9.3. MLP Modules

For each role $\in \{Q, V\}$ that uses an MLP, one is defined for a layer n as:

$$f_n(\mathbf{x}) = \mathbf{W}_n \cdot \text{LN}\left(\text{gelu}\left(\mathbf{U}_n\,\mathbf{x}\right)\right) \qquad (11)$$

Where $\mathbf{U}_n, \mathbf{W}_n$ are the down and up projections of the bottleneck ($d_{\textbf{z}} \mapsto \gamma d_{\textbf{z}} \mapsto d_{\textbf{z}}$), respectively. This means, for example, that for value transforms one set of weights is assigned to each layer n with nothing shared. gelu is the activation function from Hendrycks and Gimpel (2016).

10. Conclusion & Future Work

We present `DWAtt`—a new method of reusing the latent representations of a deep neural network. We analyze `DWAtt` and a similar, simpler method—`Concat`—from multiple aspects of performance and scaling on NER and MLM tasks. Results suggest similar layer fusion methods can be a strong tool for downstream adaptation. Performance gains from 1% to 6% and as high as 30% can be seen in various experiments for different few-shot sizes and training times, under Finetuning or Feature Extraction, and on base models of different depths. `DWAtt` and `Concat` have shown improved performance even in Feature Extraction training against full Finetuning. We believe this effect may extend to other tasks besides sequence labeling and to other sequence modeling architectures besides the Transformer—and propose such analysis for future work. We believe *add-on*-style additions to pretrained models, such as adapters and depth-wise mixing, to be a fertile ground for research into low-cost adaptation of large models that has not been saturated yet.

11. Bibliographical References

Badr AlKhamissi, Mohamed Gabr, Muhammad El-Nokrashy, and Khaled Essam. 2021. Adapting marbert for improved arabic dialect identification: Submission to the nadi 2021 shared task. In *Proceedings of the Sixth Arabic Natural Language Processing Workshop*, page 260–264. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. *CoRR*, abs/1808.07561.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient finetuning for transformer-based masked language-models. In *ACL*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus).

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. *ArXiv*, abs/1807.03819.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zi-Yi Dou, Zhaopeng Tu, Xing Wang, Longyue Wang, Shuming Shi, and Tong Zhang. 2019. Dynamic layer aggregation for neural machine translation with routing-by-agreement. *CoRR*, abs/1902.05770.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2020. Accessing higher-level representations in sequential transformers with feedback memory. *ArXiv*, abs/2002.09402.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *ArXiv*, abs/1410.5401.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *ArXiv*, abs/1606.08415.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *ICML*.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR*, abs/1608.06993.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Jinyu Li, Changliang Liu, and Yifan Gong. 2018. Layer trajectory lstm. *arXiv*, abs/1808.09522.

Fenglin Liu, Xuancheng Ren, Guangxiang Zhao, and Xu Sun. 2020. Layer-wise cross-view decoding for sequence-to-sequence learning. *CoRR*, abs/2005.08081.

Xuebo Liu, Longyue Wang, Derek F. Wong, Liang Ding, Lidia S. Chao, and Zhaopeng Tu. 2021. Understanding and improving encoder layer fusion in sequence-to-sequence learning. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

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

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

Arild Nøkland. 2016. Direct feedback alignment provides learning in deep neural networks.

Afshin Rahimi, Yuan Li, and Trevor Cohn. 2019. Massively multilingual transfer for NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 151–164, Florence, Italy. Association for Computational Linguistics.

Yanyao Shen, Xu Tan, Di He, Tao Qin, and Tie-Yan Liu. 2018. Dense information flow for neural machine translation. *CoRR*, abs/1806.00722.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Jonas Wallat, Jaspreet Singh, and Avishek Anand. 2021. BERTnesia: Investigating the capture and forgetting of knowledge in BERT. *arXiv:2106.02902 [cs]*. ArXiv: 2106.02902.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. *CoRR*, abs/1905.00537.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019b. Learning deep transformer models for machine translation. *CoRR*, abs/1906.01787.

Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. 2018b. Multi-layer representation fusion for neural machine translation. pages 3015–3026.

Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.