Multilingual Machine Translation with Hyper-Adapters

Christos Baziotis[⋆†▽] **Mikel Artetxe**[△] **James Cross**[△] **Shruti Bhosale**[⋆△]
[▽] Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
[△] Meta AI Research, Menlo Park, CA, USA

Abstract

Multilingual machine translation suffers from negative interference across languages. A common solution is to relax parameter sharing with language-specific modules like adapters. However, adapters of related languages are unable to transfer information, and their total number of parameters becomes prohibitively expensive as the number of languages grows. In this work, we overcome these drawbacks using *hyper-adapters*—hyper-networks that generate adapters from language and layer embeddings. While past work had poor results when scaling hyper-networks, we propose a rescaling fix that significantly improves convergence and enables training larger hyper-networks. We find that hyper-adapters are more parameter efficient than regular adapters, reaching the same performance with up to 12 times less parameters. When using the same number of parameters and FLOPS, our approach consistently outperforms regular adapters. Also, hyper-adapters converge faster than alternative approaches and scale better than regular dense networks. Our analysis shows that hyper-adapters learn to encode language relatedness, enabling positive transfer across languages.

1 Introduction

Multilingual neural machine translation (MNMT) models (Ha et al., 2016; Johnson et al., 2017) reduce operational costs and scale to a large number of language pairs (Aharoni et al., 2019) by using a shared representation space. This approach benefits low-resource languages through positive transfer from related languages, but introduces a *transfer-interference trade-off* (Arivazhagan et al., 2019)—as the number of languages grows, the performance in more resource-rich languages starts to drop. Prior work shows that constrained model capacity prevents models from representing all languages equally well (Arivazhagan et al., 2019). While naively increasing capacity is certain to improve performance (Arivazhagan et al., 2019; Zhang et al., 2020), it comes with large computational costs.

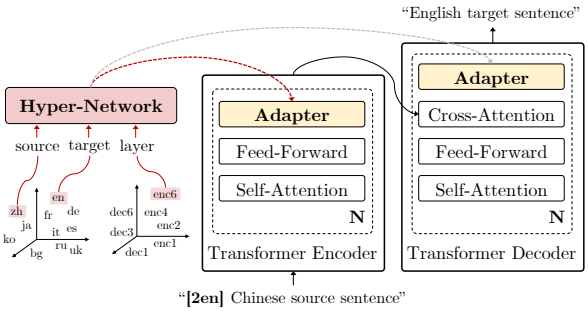

Figure 1: We inject language(-pair)-specific adapters in MNMT, by generating them from a hyper-network.

A common remedy for the capacity bottleneck is to relax the information sharing with language-specific parameters (Blackwood et al., 2018; Sachan and Neubig, 2018; Wang et al., 2019; Tan et al., 2019; Zhang et al., 2020; Fan et al., 2021). Adapter modules (Rebuffi et al., 2017) have been successfully employed in various natural language processing tasks to address similar capacity-related issues (Houlsby et al., 2019; Pfeiffer et al., 2020). In MNMT, adapters have been used to adapt (via finetuning) pretrained generic models to specific language-pairs or domains(Bapna and Firat, 2019), to improve zero-shot performance (Philip et al., 2020), or to reduce interference (Zhu et al., 2021).

However, using regular language(-pair) adapters has certain limitations. First, they can be very parameter-inefficient. While each adapter layer might be small, the total number of layers is proportional to the number of languages. This quickly becomes very costly, in particular in massively multilingual settings. In addition, there is no information sharing between the adapters of related languages. For instance, an adapter for Nepali cannot benefit from the more abundant Hindi data, which prevents positive transfer between the two languages.

⋆Correspondence to c.baziotis@ed.ac.uk or shru@fb.com
†Work done during an internship at Meta AI

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 1170 - 1185
December 7-11, 2022 ©2022 Association for Computational Linguistics

In this work, we train MNMT models with extra language-specific modules generated by a hyper-network (Ha et al., 2017). We use adapters for the language-specific modules, dubbed *hyper-adapters*, but it is trivial to replace them with any other architecture. Hyper-adapters (Figure 1) are a function of jointly trained language and layer embeddings. This approach naturally encodes language relatedness and enables knowledge transfer between related languages. It also substantially improves parameter efficiency, as the number of hyper-adapter parameters is invariant to the number of languages. We also address optimization obstacles (Sung et al., 2021) overlooked by prior work (Karimi Mahabadi et al., 2021; Ansell et al., 2021), and propose a rescaling fix that improves convergence and enables us to successfully scale to large hyper-networks.

We present experiments on a large multilingual translation benchmark. Unlike prior work (Bapna and Firat, 2019; Philip et al., 2020) that finetunes adapters for language-specific adaptation, we train regular- and hyper-adapters jointly with the main network. We show that with the same parameter budget and FLOPS, hyper-adapters are consistently better than other regular adapter variants. We also match the performance of regular adapters with hyper-adapters up to 12 times smaller. Hyper-adapters, also converge faster than other approaches and improve scalability, as small dense networks with hyper-adapters yield similar results to larger regular dense networks. Our analysis reveals that hyper-adapters do indeed exploit language similarity, unlike regular adapters. By comparing models on benchmarks with artificially constructed properties, we find that the gains of hyper-adapters grow as the redundancy (e.g., language similarities) in the training data increases.

Our main contributions are:

1. We present a novel approach that injects language-specific parameters in MNMT, by generating them from a hyper-network. We also successfully train large hyper-networks by addressing unresolved optimization obstacles.
2. We present multilingual translation experiments. Hyper-adapters consistently outperform regular adapters with the same parameter count or match the results of much larger (up to 12x) regular adapters. They also converge faster and scale better than other methods.
3. We present an analysis using a series of probes. We verify that hyper-adapters encode language relatedness, unlike regular adapters. We also find that the gains of hyper-adapters are proportional to the redundancy in the training data.

2 Background: Multilingual NMT

In this work, we train universal MNMT models following Johnson et al. (2017). We prepend a special token $\langle 2\text{XX} \rangle$ to the source and target sequences, that denotes the target language. Given a source sentence $\boldsymbol{x} = \langle x_1, x_2, ..., x_{|\boldsymbol{x}|} \rangle$, a target sequence $\boldsymbol{y} = \langle y_1, y_2, ..., y_{|\boldsymbol{y}|} \rangle$ and a target language token $\boldsymbol{t}$, we train our models as follows:

$$\boldsymbol{H} = \text{encoder}([t, \boldsymbol{x}])$$
$$\boldsymbol{S} = \text{decoder}([t, \boldsymbol{y}, \boldsymbol{H}])$$

We use the Transformer architecture (Vaswani et al., 2017) as the backbone of all our models.

2.1 Language-Specific Parameters

With universal MNMT, the issue of *negative interference* between unrelated languages emerges, and high-resource language directions are bottlenecked by constrained model capacity (Arivazhagan et al., 2019). A common solution is to extend model capacity with language-specific modules (Blackwood et al., 2018; Sachan and Neubig, 2018; Vázquez et al., 2019; Wang et al., 2019; Lin et al., 2021; Zhang et al., 2020; Fan et al., 2021).

Adapters In this work, we incorporate language-specific parameters using adapter modules, as they are generic and widely adopted by the community for multilingual or multi-task problems. We follow the formulation of Bapna and Firat (2019); Philip et al. (2020), and inject one adapter block after each Transformer layer, followed by a residual connection. Let $\boldsymbol{z}_i \in \mathbb{R}^{d_z}$ be the output of the i-th encoder or decoder layer, where d_z is the embedding dimension of the Transformer model. First, we feed $\boldsymbol{z}_i$ to a LayerNorm sublayer $\bar{\boldsymbol{z}}_i = \text{LN}_i(\boldsymbol{z}_i | \boldsymbol{\beta}, \boldsymbol{\gamma})$. Next, we transform $\bar{\boldsymbol{z}}_i$ by applying a down-projection $\boldsymbol{D_i} \in \mathbb{R}^{d_z \times d_b}$, followed by a non-linearity ϕ, an up-projection $\boldsymbol{U_i} \in \mathbb{R}^{d_b \times d_z}$, and a residual connection, where d_b is the bottleneck dimensionality of the adapter. Formally, each adapter is defined as:

$$\text{adapter}_i(\boldsymbol{z}_i) = \boldsymbol{U_i}(\phi(\boldsymbol{D_i}\, \text{LN}_i(\boldsymbol{z}_i))) + \boldsymbol{z}_i$$

In this work, we use ReLU as the non-linearity ϕ.

Adapter Variants In MNMT, prior work has used adapters for language(-pair) adaptation, via finetuning. In our work, we consider two variants but train the adapters jointly with the main network. Preliminary experiments also showed that jointly training adapters with the main networks yields better results than finetuning adapters. The first variant is *language-pair adapters* (Bapna and Firat, 2019), which uses a different adapter module per language pair in each encoder and decoder layer. This approach is effective, but it quickly becomes prohibitively expensive in a multi-parallel setting[1], as the number of adapter layers scales quadratically with the number of languages. Next, we consider (monolingual) *language adapters* (Philip et al., 2020), which use one adapter per language. Specifically, during xx→yy translation, we activate the adapters for the xx (source) language in the encoder and the yy (target) language in the decoder. Thus, they require fewer adapter layers while also they generalize to unseen translation directions.

3 Hyper-Adapters

We propose to use a hyper-network (Ha et al., 2017), a network that generates the weights of another network, to produce the weights of all adapter modules, dubbed *hyper-adapters*. As shown in Figure 2, we use a single hyper-network to generate adapters for all languages and layers by conditioning on $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{l})$ tuples, where $\boldsymbol{s}$ and $\boldsymbol{t}$ denote the source and target language and $\boldsymbol{l}$ denotes the encoder or decoder layer-id (e.g., enc3). Unlike regular adapters, our approach enables information sharing across languages and layers, and the hyper-network can learn to optimally allocate its capacity across them. Our hyper-network has 3 components:

Input We first embed $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{l})$. We use a shared matrix for the source and target language embeddings, and a separate matrix for the layer-id embeddings for all encoder and decoder layers.

Encoder The language and layer embeddings are given as input to the hyper-network encoder. First, we concatenate the embeddings and project them with $\boldsymbol{W}_{\text{in}}$, followed by a non-linearity ϕ[2], to obtain the hyper-network hidden representation $\boldsymbol{h} \in \mathbb{R}^{d_h}$:

$$\boldsymbol{h} = \phi(\boldsymbol{W}_{\text{in}} [\boldsymbol{s} \| \boldsymbol{t} \| \boldsymbol{l}]) \quad (1)$$

[1]Multi-parallel refers to a fully many-to-many setting, unlike the English-centric setting that is {en→X ∪ X→en}.

[2]In this work we use ReLU.

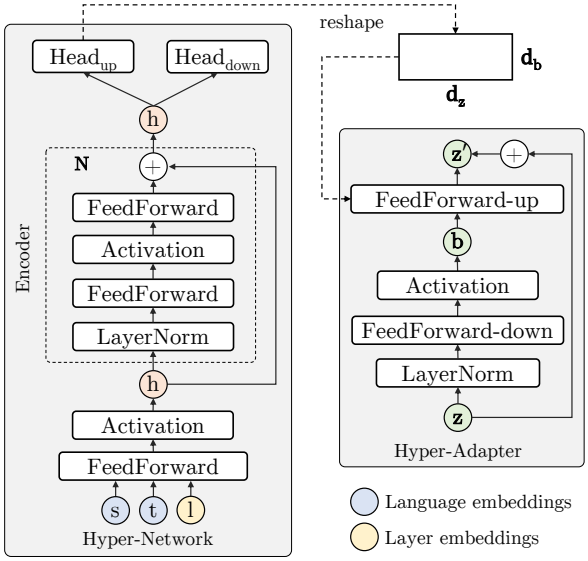

Figure 2: We feed source language, target language and layer-id embeddings into a shared hyper-network, to generate adapters weights for all languages and layers.

where $\|$ denotes the concatenation operation. We then pass $\boldsymbol{h}$ through N residual blocks, to encode high-level interactions between the input features:

$$\text{enc}(\boldsymbol{h_{i+1}}) = \boldsymbol{W_2}(\phi(\boldsymbol{W_1}\,\text{LN}(\boldsymbol{h_i}))) + \boldsymbol{h_i} \quad (2)$$

where $\boldsymbol{W_1} \in \mathbb{R}^{d_h \times d_h}$ and $\boldsymbol{W_2} \in \mathbb{R}^{d_h \times d_h}$ are the trainable weights of each residual block.

Projections We feed the final representation $\boldsymbol{h}$ to separate projection heads to obtain (by reshaping their outputs) each weight matrix of a hyper-adapter. Specifically, we use $\boldsymbol{H}_{\text{up}} \in \mathbb{R}^{d_h \times (d_b d_z)}$ to generate the weights for each up-projection $\boldsymbol{U} \in \mathbb{R}^{d_b \times d_z}$, $\boldsymbol{H}_{\text{down}} \in \mathbb{R}^{d_h \times (d_z d_b)}$ to generate the weights for each down-projection $\boldsymbol{D} \in \mathbb{R}^{d_m \times d_b}$. We also generate the LayerNorm parameters $\boldsymbol{\gamma} \in \mathbb{R}^{d_z}$ and $\boldsymbol{\beta} \in \mathbb{R}^{d_z}$, with the projection heads $\boldsymbol{H}_{\boldsymbol{\gamma}} \in \mathbb{R}^{d_h \times d_z}$ and $\boldsymbol{H}_{\boldsymbol{\beta}} \in \mathbb{R}^{d_h \times d_z}$, respectively.

3.1 Unlocking Large Hyper-networks

Prior work (Karimi Mahabadi et al., 2021; Ansell et al., 2021) in natural language understanding (NLU) has used the equivalent of small values of $\boldsymbol{d_h}$ and only considered finetuning. Sung et al. (2021, Fig. 4), recently found that scaling up hyper-networks (Karimi Mahabadi et al., 2021) leads to poor results, which they attributed to unknown optimization issues. In preliminary experiments, we found similar issues when using larger $\boldsymbol{d_h}$ values (i.e., increasing the hyper-network size). We found that the issues were more pronounced when training hyper-adapters jointly with the main network

from scratch, and speculate that this is a harder optimization problem than training them with a pre-trained and frozen main network. Next, we identify the cause of this problem and propose a simple fix that allows us to effectively scale hyper-adapters.

Figure 3 shows the training loss curve as we vary d_h. We find that increasing the hyper-network size by increasing d_h leads to worse instead of better performance and also makes training very unstable. In Figure 4, we plot the average standard deviation (SD) of the Transformer layer activations during training, and find that for small d_h, the activations stay within a healthy range, but as we increase d_h, the activations start to grow fast. After a certain point, the network fails to recover and the activations grow to extreme values.

To solve this issue, we scale down the generated adapter weights by $\frac{1}{\sqrt{d_h}}$, and generate the adapter weights as $\tilde{W} = \text{reshape}(\frac{\boldsymbol{H}\,\boldsymbol{h}}{\sqrt{d_h}})$. Note that, each component of the generated adapter matrix $\tilde{W}$ is the dot-product of $\boldsymbol{h}$ and the corresponding column of a given projection head $\boldsymbol{H}$. Thus, the generated weights' SD is proportional to d_h. The motivation is similar to the scaled dot-product in Transformer's self-attention. Once we apply the rescaling fix, the activations stay within a healthy range (Figure 4), and increasing d_h improves convergence as expected (Figure 3). Note that, in this work we consider variants with $d_h > 512$, and the rescaling fix is crucial to unlocking these variants.

3.2 Parameter Efficiency and FLOPS

Given N languages, language adapters introduce N new modules, whereas language-pair adapters introduce N^2 new modules in a multi-parallel setting or $2N$ modules in an English-centric many-to-many setting. By contrast, the number of extra parameters in hyper-adapters is invariant to both the number of languages and layers. Most of the parameters are in the projection heads. Intuitively, each row of a head's weight matrix is equivalent to a (flattened) adapter weight matrix. The number of rows in each head is equal to the hidden size d_h, thus d_h controls its capacity. Therefore, to reduce the memory needs compared to language adapters we must use $d_h < N$, and $d_h < 2N$ for *English-centric* language-pair adapters (details in Appendix B.3).

In terms of computational cost, all adapter and hyper-adapter variants yield models with the same FLOPS. This is because, at test time, we activate only the main network and the corresponding

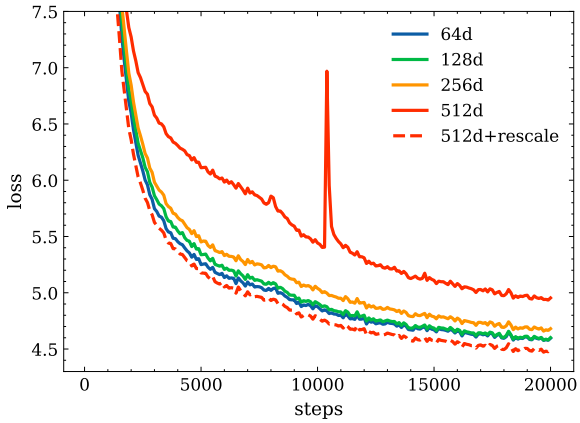

Figure 3: Effect of increasing d_h on training. Without rescaling the weights, as we use bigger hyper-networks, training becomes unstable and the loss increases.

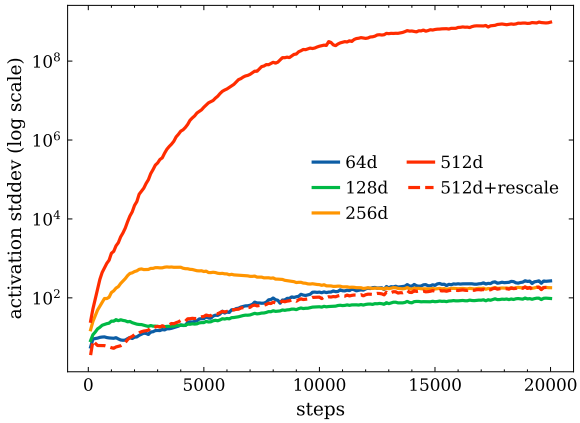

Figure 4: Transformer layer activations as we vary d_h.

adapters, with both regular and hyper-adapters having identical architecture and size. During training, hyper-adapters incur an additional cost for generating the adapter parameters. However, this cost is negligible in practice, as it is run only once per batch for each language pair. At test time, the generated weights can be precomputed and cached.

4 Experimental Setup

Data We present results on ML50 (Tang et al., 2020), a multilingual translation dataset with 230M sentences between English and 50 other typologically diverse languages with data from different domains, and is larger than comparable publicly available MNMT datasets (e.g., 4x larger than OPUS100; Zhang et al. 2020). We concatenate the En→X and X→En directions, and group languages based on the amount of their training data into HIGH (≥1M, 14 languages), MED (≥100K, 17 languages) and LOW (<100K, 19 languages). We use SentencePiece[3] (Kudo and Richardson, 2018)

Model	Params		En→X				X→En				Mean
	Total	Extra	All	High	Med	Low	All	High	Med	Low	
Transformer-base	90M	-	16.8	20.2	14.3	16.6	23.9	25.3	23.0	23.6	20.3
+lang-adapters	171M	81M	18.1	21.0	15.5	18.3	25.2	26.6	24.8	24.4	21.6
+pair-adapters	250M	159M	18.3	21.5	16.0	18.1	24.7	25.9	24.5	23.8	21.5
+hyper-adapters (17%)	104M	14M	17.9	21.5	15.4	17.5	24.9	26.3	24.1	24.5	21.4
+hyper-adapters (33%)	118M	27M	18.5	22.0	16.0	18.2	25.3	26.6	24.5	**25.0**	21.9
+hyper-adapters (100%)	173M	83M	**19.0**	**22.2**	**16.6**	**18.9**	**25.7**	**26.9**	**25.3**	**25.0**	**22.3**

Table 1: Results with *Transformer-base* models and (hyper-)adapter bottleneck size of 128.

to obtain a joint vocabulary of 90k symbols. We explored smaller and larger vocabularies but empirically found that 90k strikes a good balance between the number of parameters and translation quality. Finally, we filter out pairs with more than 250 tokens or with a length ratio over 2.5.

Sampling To obtain a more balanced data distribution we use temperature-based sampling (Arivazhagan et al., 2019). Assuming that p_L is the probability that a sentence belongs to language L, we sample sentences for L with a probability proportional to $p_L^{1/T}$, where T is a temperature parameter. Larger values of T lead to more even sampling across languages. During preprocessing, we train SentencePiece with T=5. During training, we set T=2 as we observed that with larger values, models overfit on low-resource languages.

Model Configuration We use the Transformer-Base architecture (Vaswani et al., 2017) in most of our experiments, which has 6 encoder and decoder layers, embedding size of 512, feed-forward filter size of 2048, 8 attention heads, and 0.1 dropout. To verify the effectiveness of our approach with more large-scale models, we also consider an experiment with the Transformer-Big configuration, which uses embedding size of 1024, feed-forward filter size of 4096, 16 attention heads, and 0.3 dropout. In all models, we tie the encoder-decoder embeddings and the decoder output projections (Press and Wolf, 2017; Inan et al., 2017). All models are implemented in Fairseq (Ott et al., 2019).

Optimization We use Adam (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-6}$ and regularize models with label smoothing (Szegedy et al., 2016) with $\alpha = 0.1$. We train Transformer-Base models with a learning rate of 0.004 for 360k steps, and Transformer-Big models with a learning rate of 0.001 for 220k, using a linear warm-up of 8k steps, followed by inverted squared decay. All models are optimized with large batches of 256k tokens (8k $\times$ 32 V100 GPUs). The training time for all models is similar, ranging from 4 to 5 days, with adapter variants being slightly slower than their dense counterparts.

Evaluation During training, we evaluate models every 20k steps and select the checkpoint with the best validation loss, aggregated across languages. At test time, we use beam search of size 5. We evaluate all models using BLEU (Papineni et al., 2002) computed with SacreBLEU[4] (Post, 2018).

Baselines We compare with strong baselines that incorporate language-specific parameters into MNMT. We consider two adapter variants that yield significant improvements over dense MNMT models, namely (monolingual) *language adapters* and *language-pair adapters* and set their bottleneck size to 128. Given that ML50 contains 51 languages in total, language adapters require 612 adapter modules (51 $\times$ 12), whereas language-pair adapters require 1224 (i.e., twice as many).

Hyper-adapter Settings We use our proposed hyper-network to generate hyper-adapters with *identical* architecture as their regular adapter counterparts. We consider three hyper-network variants in our experiments: *base* ($d_h = 612$), *small* ($d_h = 204$) and *tiny* ($d_h = 102$). They contain roughly 100%, 33%, and 17% of the parameters of language adapters[5], respectively. We set the size of the language and layer embeddings to 50 and use 2 layers in the hyper-network encoder.

5 Results

Main Results Table 1 shows our main results. All the reported results are from single runs, as MNMT training is computationally expensive.

[3]We use the `unigram` model with coverage 0.99995.

[4]BLEU+case.mixed+lang.S-T+numrefs.1+smooth.exp+tok.13a+v1.5.1

[5]Or 50%, 17% and 8% w.r.t. language-pair adapters

Model	Params		En→X				X→En				Mean
	Total	Extra	All	High	Med	Low	All	High	Med	Low	
Transformer-Big	269M	-	18.5	21.2	15.7	19.1	25.7	26.6	25.0	25.7	22.1
+lang-adapters	591M	323M	19.6	22.3	17.0	20.0	27.3	28.1	27.0	27.1	23.5
+pair-adapters	902M	633M	20.0	**22.9**	17.5	20.3	27.0	28.2	27.0	26.3	23.5
+hyper-adapters (tiny)	323M	54M	19.7	22.4	16.8	20.4	27.4	28.0	26.7	**27.6**	23.5
+hyper-adapters (small)	377M	108M	20.0	22.8	17.2	20.5	**27.5**	**28.3**	26.9	27.4	23.7
+hyper-adapters (base)	594M	325M	**20.3**	**22.9**	**17.6**	**20.9**	27.4	**28.3**	**27.1**	27.2	**23.9**

Table 2: BLEU (↑) scores of the *Transformer-big* models with (hyper-)adapter bottleneck size of 256.

However, the results are averaged across 50 languages and 100 translation directions, which makes them robust to noise. For completeness, we include the non-aggregate results in the appendix (§ C).

Consistent with prior work on fine-tuning adapters for MNMT, we find that adding language(-pair) adapters brings substantial improvements across the board (Bapna and Firat, 2019; Philip et al., 2020). However, the dense (Transformer-Base) baseline has fewer parameters and FLOPS than all adapter variants.

Hyper-adapters-base consistently outperforms both regular adapter variants in all directions, while having the same parameter count as lang-adapters and half the parameter count of pair-adapters. We also find that our smaller variants yield very competitive results to regular adapters while being more parameter efficient. Hyper-adapters-small outperforms both regular adapter variants with fewer parameters, and hyper-adapters-tiny yields comparable results with only 1/6th and 1/12th of the capacity of lang-adapters and pair-adapters, respectively.

In the En→X directions, hyper-adapters-base outperforms lang-adapters by 0.9 BLEU and pair-adapters by 0.7 BLEU. Interestingly, we see gains even in high-resource settings up to +1.2 BLEU, although regular adapters have dedicated capacity for these language(-pairs). In X→En, hyper-adapter-base has smaller improvements on medium- and high-resource languages, but we observe improvements of +1.2 BLEU on low-resource languages. We hypothesize that the lower improvements on X→En compared to En→X are partly due to language specific capacity being more valuable when decoding into many different languages.

Regular Adapters We discover interesting trade-offs between the regular adapter variants. Pair-adapters are better in En→X, which suggests that it is beneficial to have dedicated capacity for encoding the source-side of each En→X pair. By contrast, language-adapters are stronger in X→En. We believe this is because the (single) decoder-side English adapter benefits from observing all the target-side English data, unlike the separate X-En adapters that see only the target-side English data of each pair. However, hyper-adapters enjoy the best of both approaches, while being more efficient.

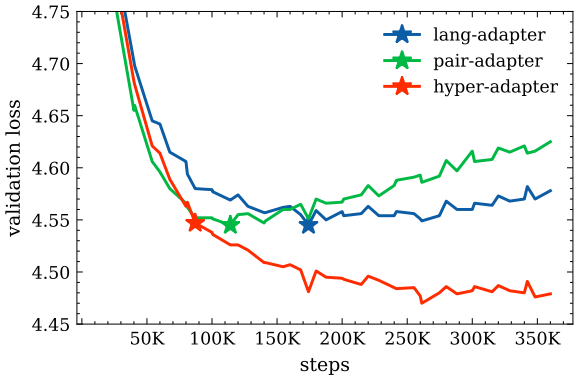

Figure 5: Validation losses of adapter variants. We mark when each variant reaches the best loss of lang-adapters.

Convergence In Figure 5, we compare the validation loss curves of each adapter variant with our hyper-adapters-base variant, which has the same size as lang-adapters. We mark the point at which each variant reaches the best loss of lang-adapters. First, we observe that hyper-adapters converge to the best lang-adapters loss at half the number of steps (87K-vs-174K). This shows that assuming a fixed parameter budget, hyper-adapters can significantly reduce training time. We also find that regular adapters suffer from overfitting, in particular pair-adapters. We suspect this is because using the same capacity for all languages is suboptimal. Bapna and Firat (2019) proposed to use bottleneck sizes proportional to the available training data of a given language pair, which requires tuning. By contrast, hyper-adapters automatically learn to allocate their available capacity as needed.

Model	Org	Sim	Dist	Acc ↑
+lang-adapters	34.1	19.0	6.5	0.56
+pair-adapters	33.7	18.0	5.6	0.53
+hyper-adapters (base)	34.8	**21.7**	4.9	**0.62**

Table 3: BLEU (↑) scores of models on the X→En adapter relatedness probe. *Org*, *Sim*, *Dist*, refer to using the original, similar, and distant source languages, respectively, while *Acc* denotes the ratio *Sim*/*Org*.

Model	Original		Artificial	
	Param	BLEU	Param	BLEU (Δ)
Transformer-Base	90M	23.9	90M	23.7 (-0.2)
+lang-adapters	114M	24.7	167M	23.8 (-0.9)
+pair-adapters	135M	24.8	240M	23.9 (-0.9)
+hyper-adapters	114M	24.9	114M	24.9 (-0.0)

Table 4: BLEU (↑) scores on the original-vs-artificial ML15 splits.

Large-Scale Models We also evaluate models using the Transformer-Big architecture. In these experiments, we set the bottleneck size in all adapter and hyper-adapter variants to 256. We report results in Table 2. Overall, we observe similar trends across models as with the Transformer-Base architecture, although the gains of hyper-adapters are smaller. We believe this is because we only scale up the main network, while keeping constant the amount of training data. Therefore, this mitigates the negative interference by reducing the capacity bottleneck, and leaves less room for improvement for language-specific modules, like hyper-adapters.

To our surprise, we find that hyper-adapter-base with the Transformer-Base architecture (Table 2) achieves comparable results with the Transformer-Big baseline, while having significantly fewer parameters (173M-vs-269M) and a smaller computation cost. This suggests that hyper-adapters are more effective for addressing negative interference than naively scaling up dense networks.

6 Analysis

This section investigates why hyper-adapters outperform regular adapters. Specifically, we focus on how well each adapter variant encodes language relatedness and how it is affected by the redundancy in the training data (i.e., similarities between the data of different languages). We also explore how modifications in the hyper-network architecture affect the final model performance.

6.1 (Hyper-)Adapter Language Relatedness

We design a probe (Table 3), that explicitly compares the ability of regular-vs-hyper adapters to encode language relatedness. At test time, instead of using the adapters of the original source language, we activate the adapters of another similar, or distant, language.[6] We focus on X→En, as we found that changing the target language produced very low BLEU scores, making comparisons unreliable. We select 4 low-resource languages which have a similar high-resource neighbour in our dataset, namely {af→nl, pt→es, gl→pt, uk→ru}. Also, we consider replacement with "zh", which is high-resource but distant to all 4 source languages.

When using related languages, hyper-adapters suffer less than regular-adapters, as they recover more (62%) of their original BLEU. Pair-adapters yield worse results than lang-adapters, presumably due to having weaker target-side (X-En) adapters. When using an unrelated language, hyper-adapters suffer the most. These findings further support that our hyper-networks encode language relatedness.

6.2 The Role of Data Redundancy

We have hypothesized that our hyper-network exploits similarities (i.e., redundancies) in the data, to produce similar adapters for similar languages and avoid encoding redundant features. This implies that hyper-adapters would "degenerate" into regular adapters if the training data contained only distant languages. To test this hypothesis, we create two different splits out of ML50, with and without similar languages. First, we select 14 (+English) relatively unrelated languages and create ML15[7]. Then, we create another version of ML15, that emulates a dataset with similar languages. We split the data of each language into smaller parts (e.g., $\text{fr}_1, \text{fr}_2, \ldots, \text{fr}_N$) which we treat as different languages, which results in 47 artificial languages.

Table 4 shows the results. We observe that in the original ML15 version, regular- and hyper-adapters achieve similar results. In contrast, in the fragmented ML15 version, regular adapters suffer significantly as they cannot share information, unlike hyper-adapters that are unaffected. These findings show that the gains of hyper-adapters are proportional to the redundancies in the training data. Thus, we expect that the gap between regular- and hyper-adapter will grow as the number of related languages (or their data) grows. Note that, as the artificial ML15 has more languages, regular adapters

[6] For hyper-adapters, we change the source language-id $\boldsymbol{s}$.

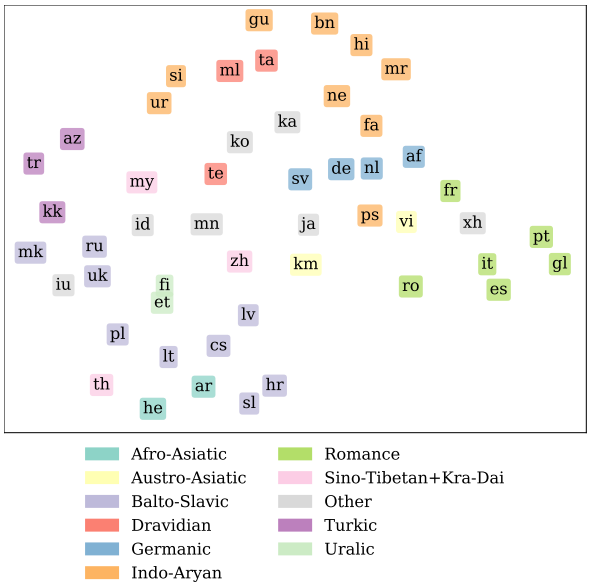

Figure 6: Plot of hyper-network language embeddings.

Model	En→X	X→En	Mean
Linear	17.3	24.4	20.8
Non-Linear	18.2	25.0	21.6
Non-Linear + 2 ResBlocks	18.5	25.2	21.8

Table 5: Comparison of encoding methods with Transformer-base models trained for 160K steps.

require more layers and thus more parameters.

6.3 Hyper-Network Embeddings

In Figure 6, we visualize the language embeddings of our hyper-adapters-tiny variant using UMAP (McInnes et al., 2018). We observe that the hyper-network embeds languages that belong to the same family close to each other. This is another piece of evidence that hyper-adapters encode language relatedness.

6.4 Hyper-Network Encoder

In Table 5, we compare different methods for encoding the hyper-network inputs $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{l})$ for obtaining the hyper-network output representations $\boldsymbol{h}$ (i.e., before generating the hyper-adapter weights). We find that using only one linear layer is suboptimal, and stacking multiple non-linear layers is important. Specifically, adding a non-linearity to Eq. 1 (we used ReLU) improves performance, and stacking more layers helps even further (Eq. 2). We speculate this allows the input features to better interact with each other. In preliminary experiments, we found that stacking more than 2 layers did not produce consistent improvements.

[7]The languages of ML15 are {en, fr, zh, hi, lt, iu, et, ro, nl, it, ar, tr, km, vi, uk}. We include more details in Appendix A.

Model	Supervised		Zero-Shot	
	En→X	X→En	Direct	Pivot
Transformer-Base	16.2	23.1	12.3	16.5
+lang-adapters	17.9	24.9	13.1	18.8
+ *hyper-adapters (base)*				
enc=$(\boldsymbol{s}, \boldsymbol{t})$ dec=$(\boldsymbol{s}, \boldsymbol{t})$	18.5	25.1	1.7	19.4
enc=$(\boldsymbol{s}, \boldsymbol{t})$ dec=$(\boldsymbol{t})$	**18.6**	**25.2**	8.7	**19.6**
+ dropout=0.1	18.3	25.0	11.4	19.3
+ dropout=0.2	18.1	25.0	11.7	18.8
enc=$(\boldsymbol{s})$ dec=$(\boldsymbol{t})$	17.8	**25.2**	**13.8**	19.1
+ dropout=0.1	17.7	25.1	13.7	18.7
+ dropout=0.2	17.5	24.8	12.9	18.8

Table 6: Effect of different hyper-network input combinations on zero-shot translation. The layer embedding $\boldsymbol{l}$ is always used and is omitted for brevity.

6.5 Zero-Shot Translation

In this analysis (Figure 6), we investigate the zero-shot capabilities of different hyper-adapter variants. Specifically, we mask either the source or target language in the hyper-network's input $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{l})$ when generating the encoder or decoder hyper-adapters. We train models for 160k steps to reduce training time. This means that hyper-adapter haven't fully converged (Figure 5), unlike regular adapters. However, we are interested in comparing different hyper-adapter variants to each other and include lang-adapters only for context.

Test Data To compute the zero-shot results we use the 15 translation combinations between Arabic, Chinese, Dutch, French, German, and Russian, following the setup of Zhang et al. (2020). We use the devtest splits from the FLORES-200 multi-parallel evaluation benchmark (Goyal et al., 2022; NLLB-Team et al., 2022). Each test set contains 3001 sentences from Wikipedia articles. We evaluate models both in direct zero-shot (i.e., X→Y) and pivot zero-shot through English (i.e., X→En→Y). Note that pair-adapters cannot do direct zero-shot translation by definition.

Results Hyper-adapters fail at direct zero-shot translation when using both $\boldsymbol{s}$ and $\boldsymbol{t}$ in the hyper-network for both the encoder and decoder hyper-adapters. Masking $\boldsymbol{s}$ in decoder hyper-adapters yields a significant boost, which is further increased by masking $\boldsymbol{t}$ in encoder hyper-adapters. This reveals a trade-off between supervised and zero-

shot translation. Removing the target language information from encoder hyper-adapters harms En→X translation, which is reflected in the pivot-based zero-shot translation. However, removing the source language information from decoder hyper-adapters has no effect on supervised translation, although it improves zero-shot. These results suggest that the "enc=$(\boldsymbol{s}, \boldsymbol{t})$ dec=$(\boldsymbol{s}, \boldsymbol{t})$" variant behaves similar to language-pair adapters, which cannot do zero-shot, whereas the "enc=$(\boldsymbol{s})$ dec=$(\boldsymbol{t})$" variant behaves similar to language-adapters. In our experiments, we use the "enc=$(\boldsymbol{s}, \boldsymbol{t})$ dec=$(\boldsymbol{t})$" variant, which strikes a good balance.

We also explore adding dropout inside the hyper-network layers, to produce more robust representations $\boldsymbol{h}$, but not in the generated hyper-adapters. We observe small negative effects in the supervised setting, but mixed results in the zero-shot setting. In particular, in the "enc=$(\boldsymbol{s}, \boldsymbol{t})$ dec=$(\boldsymbol{t})$" variant, dropout significantly improves zero-shot. These results suggest that there is room for improvement in this direction, but we leave this for future work.

7 Related Work

Platanios et al. (2018) explored an idea similar to hyper-networks in MNMT with the so-called "contextual parameter generation" to promote information sharing across languages, by generating the weights of an RNN-based (Bahdanau et al., 2015) MNMT model from language embeddings. By contrast, we consider a hybrid approach that generates only a few (language-specific) modules, instead of generating all the layers of a Transformer model, which introduces a large computational overhead.

Another approach is combining hyper-networks with pretrained models. In NLU, Karimi Mahabadi et al. (2021) generate task-specific adapters from task embeddings. Tay et al. (2021) use a hyper-network to learn grid-wise projections for different tasks. Ye and Ren (2021) extend text-to-text Transformers (Raffel et al., 2020) to unseen tasks by generating adapters from task descriptions.

In multilingual dependency parsing, Üstün et al. (2020) generate adapters for the biaffine attention from language representations in linguistic databases. Ansell et al. (2021) also use linguistic databases for cross-lingual NLU, and extend Pfeiffer et al. (2020) by generating language adapters for unseen languages. In concurrent work, (Üstün et al., 2022) consider a conceptually similar approach to our work for multi-task multilingual transfer in NLU tasks.

Unlike prior work, (1) we identify and solve optimization issues overlooked by other hyper-network-based methods, (2) we train regular- and hyper-adapters jointly with the main network instead of using them for finetuning, and (3) we focus on NMT, which is a more complex generation task instead of the relatively simpler NLU tasks.

8 Conclusion

In this work, we extend the capacity of MNMT models with hyper-adapters, which are language-specific adapter modules generated from a hyper-network. By resolving optimization issues not addressed by prior work, we successfully train large hyper-networks from scratch jointly with the rest of the main network on MNMT (§3.1).

We show that hyper-adapters consistently outperform other regular adapter variants across translation directions and model sizes (§5), while improving parameter efficiency. We also observe computational efficiency gains, as a smaller Transformer-Base model with hyper-adapters gives similar results to the dense Transformer-Big model, which is computationally more expensive and requires more parameters. Besides improvements in translation quality, hyper-adapters achieve faster training convergence as shown in §5. Our analysis shows that, unlike regular adapters, hyper-networks enable positive transfer across the hyper-adapters of similar languages, by encoding language relatedness (§6.1,6.3) and exploiting redundancies (i.e., language similarities) in the training data (§6.2). Finally, by manipulating the input of the hyper-network we discover that there is a trade-off between the zero-shot and supervised translation performance of hyper-adapters (§6.5).

Limitations

Modeling As mentioned in Section 3.2, one limitation of hyper-adapters, compared to regular adapters, is that they introduce a small computational overhead during training. Specifically, in each batch, we need to do one pass through the hyper-network to generate the hyper-adapter parameters. This cost is proportional to the size of the hyper-network and the total number of transformer layers (and thus the hyper-adapter layers to generate). In this work, we found that this cost was negligible, as the number of total transformer layers is small (12) and 2-layer deep hyper-network was suf-

ficient to obtain good results. Besides, the parameter generation cost only affects training time. Once the training is completed we can pre-generate and cache all the hyper-adapter modules, thus obtaining identical inference cost with regular adapters.

Data In our experiments, we use only the ML50 dataset, which is relatively larger than those used in prior works. However, ML50 contains only English-centric parallel data and real-world multilingual datasets can be much larger, noisy, and diverse (NLLB-Team et al., 2022; Bapna et al., 2022).

Acknowledgments

We thank Angela Fan, Myle Ott, Vedanuj Goswami, and Naman Goyal for all their help and advice during this project.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. 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 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Alan Ansell, Edoardo Maria Ponti, Jonas Pfeiffer, Sebastian Ruder, Goran Glavaš, Ivan Vulić, and Anna Korhonen. 2021. MAD-G: Multilingual adapter generation for efficient cross-lingual transfer. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4762–4781, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, San Diego, CA, USA.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant, Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, et al. 2022. Building machine translation systems for the next thousand languages. *arXiv preprint arXiv:2205.03983*.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. 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)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Graeme Blackwood, Miguel Ballesteros, and Todd Ward. 2018. Multilingual neural machine translation with task-specific attention. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3112–3122, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Oscar Chang, Lampros Flokas, and Hod Lipson. 2020. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Michael Auli, and Armand Joulin. 2021. Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, 22(107):1–48.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder. In *International Workshop on Spoken Language Translation (IWSLT)*, Seattle, USA.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

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 *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proceedings of the International Conference on Learning Representations*.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576, Online. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, San Diego, CA, USA.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Zehui Lin, Liwei Wu, Mingxuan Wang, and Lei Li. 2021. Learning language specific sub-network for multilingual machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 293–305, Online. Association for Computational Linguistics.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. 2018. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861.

NLLB-Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. No language left behind: Scaling human-centered machine translation.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jerin Philip, Alexandre Berard, Matthias Gallé, and Laurent Besacier. 2020. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. Association for Computational Linguistics.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. 2018. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435, Brussels, Belgium. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou,

Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Devendra Sachan and Graham Neubig. 2018. Parameter sharing methods for multilingual self-attentional translation models. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 261–271, Brussels, Belgium. Association for Computational Linguistics.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2021. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. *arXiv preprint arXiv:2112.06825*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Xu Tan, Jiale Chen, Di He, Yingce Xia, Tao Qin, and Tie-Yan Liu. 2019. Multilingual neural machine translation with language clustering. 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)*, pages 963–973, Hong Kong, China. Association for Computational Linguistics.

Y. Tang, C. Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, and Angela Fan. 2020. Multilingual translation with extensible multilingual pretraining and finetuning. *ArXiv*, abs/2008.00401.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2021. Hypergrid transformers: Towards a single model for multiple tasks. In *International Conference on Learning Representations*.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, Gertjan van Noord, and Sebastian Ruder. 2022. Hyper-x: A unified hypernetwork for multi-task multilingual transfer. *arXiv preprint arXiv:2205.12148*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Raúl Vázquez, Alessandro Raganato, Jörg Tiedemann, and Mathias Creutz. 2019. Multilingual NMT with a language-independent attention bridge. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 33–39, Florence, Italy. Association for Computational Linguistics.

Xinyi Wang, Hieu Pham, Philip Arthur, and Graham Neubig. 2019. Multilingual neural machine translation with soft decoupled encoding. In *International Conference on Learning Representations*.

Qinyuan Ye and Xiang Ren. 2021. Learning to generate task-specific adapters from task description. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 646–653, Online. Association for Computational Linguistics.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Counter-interference adapter for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2812–2823, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Language	Sentences	# Splits
fr_XX	38,507,539	5
zh_CN	11,173,646	5
hi_IN	1,450,114	5
lt_LT	1,402,892	5
iu_CA	1,109,076	5
et_EE	1,064,974	5
ro_RO	600,019	5
nl_XX	232,038	2
it_IT	226,385	2
ar_AR	225,678	2
tr_TR	203,702	2
km_KH	183,934	2
vi_VN	127,117	1
uk_UA	104,021	1

Table 7: Statistics of the Ml15 dataset, include the number of splits per language in the artificial version.

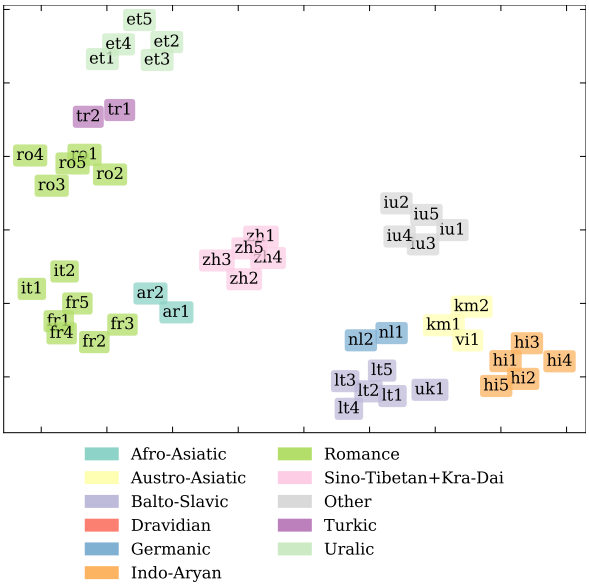

Figure 7: Plot of hyper-network language embeddings trained on the fragmented version of ML15 with the artificial languages.

A ML15 Dataset

In Table 7, we show the statistics of the ML15 dataset. It includes 14 (+English) medium- to high-resource languages from ML50, that are relatively distant from each other. We also create another version of this dataset, in which we split each language into smaller parts and consider each one of them as a different language. We aim to have approximately 100k sentences per split with at most 5 splits per language, to prevent the number of artificial languages from becoming too large.

In Figure 7, we visualize the hyper-network language embeddings of the model trained on the fragmented version of ML15 with the artificial languages. The plot clearly demonstrates that the hyper-network is able to capture the fact that all the artificial splits of a given language are similar to each other. Based on that, it is able to avoid relearning the same features, while also exploiting the all the available (related) data to learn to generate more powerful hyper-adapters.

B Hyper-Network Architecture

B.1 Initialization

Classical weight initialization methods (Glorot and Bengio, 2010; He et al., 2015), when used to initialize a hyper-network, fail to produce weights for the main network on the correct scale. We explore a simple and generic solution to properly initialize each hyper-network head.

First, we initialize a given projection head $\boldsymbol{H}$ with a regular initialization method. Then, we also randomly initialize another (temporary) weight matrix, with the same dimensions as the adapter matrix we want to generate from $\boldsymbol{H}$, and compute its standard deviation $\boldsymbol{\sigma_a}$. Recall that, we want $\boldsymbol{H}$ to generate adapter weights in the target scale of $\boldsymbol{\sigma_a}$. Next, we feed a random input $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{l})$ into the hyper-network, generate the hyper-adapter weight matrix, and compute its standard deviation $\boldsymbol{\sigma_h}$. Finally, we re-scale the original weights of $\boldsymbol{H}$ as

$$\boldsymbol{H'} = \boldsymbol{H} \odot \frac{\boldsymbol{\sigma_a}}{\boldsymbol{\sigma_h}}$$

which ensures that the next time the projection head $\boldsymbol{H}$ will generate a weight matrix, it will have values within the desired scale. In our experiments, we found that this hyper-network aware initialization was helpful only when not using our proposed re-scaling (§ 3.1). However, once we employ the re-scaling, all models converge to the same results regardless of initialization.

Chang et al. (2020) first pointed out the importance of properly initialing a hyper-network. However, while their proposed initialization is principled, it requires to be computed analytically for each source→target layer mapping. By contrast, our method simply initializes a target layer and numerically adjusts the weights of the hyper-network, which works for arbitrary layer architectures.

B.2 LayerNorm Generation

In regular LayerNorm, we initialize the scaling parameters γ with 1 and the shifting parameters β with 0. However, when we generate $\tilde{\gamma}$, $\tilde{\beta}$ from

Model	Params		En→X				X→En				Mean
	Total	Extra	All	High	Med	Low	All	High	Med	Low	
Transformer-Base	90M	-	41.7	45.3	40.6	40.1	47.8	53.0	46.8	44.7	44.7
+lang-adapters	171M	81M	43.1	46.2	42.0	41.8	49.2	**54.4**	48.7	45.8	46.1
+pair-adapters	250M	159M	43.6	46.8	42.9	42.0	48.9	54.2	48.7	45.3	46.3
+hyper-adapters (tiny)	104M	14M	42.7	46.5	41.5	40.9	48.6	53.9	47.7	45.6	45.7
+hyper-adapters (small)	118M	27M	43.5	46.9	42.4	41.9	48.9	54.2	48.0	45.8	46.2
+hyper-adapters (base)	173M	83M	**44.2**	**47.2**	**43.3**	**42.8**	**49.4**	54.3	**48.8**	**46.3**	**46.8**

Table 8: ChrF (↑) scores of the *Transformer-base* models with (hyper-)adapter bottleneck size of 128.

the hyper-network, their values (initially) will be zero-mean as they are the activations of a randomly initialized projection. This can cause convergence issues, because if the values of $\tilde{\gamma}$ are close to zero, then the inputs z would be scaled down close to zero, thus slowing down convergence. To address this issue, we increment the generated weights for $\tilde{\gamma}$ by +1, to ensure that they have the desired scale:

$$\text{LN}(\boldsymbol{z}_i|\tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{\beta}}) = \frac{\boldsymbol{z}_i - \boldsymbol{\mu}_{\boldsymbol{z}_i}}{\boldsymbol{\sigma}_{\boldsymbol{z}_i}} \odot (\tilde{\boldsymbol{\gamma}} + \mathbb{1}) + \tilde{\boldsymbol{\beta}}$$

where $\mathbb{1}$ denotes a vector of ones.

B.3 Parameter Efficiency

In this section, we discuss the parameter efficiency of each (hyper-)adapter variant in greater detail. For brevity, we ignore the (negligible) LayerNorm parameters.

Regular Adapters Each adapter block has an up- and a down-projection with equal parameters and total capacity $C_{\text{block}} = d_z d_b + d_b d_z = 2 d_z d_b$. Language adapters add $C_{\text{lang}} = N \cdot L \cdot C_{\text{block}}$ new parameters into an MNMT model, where $\boldsymbol{N}$ is the number of languages and L the number of (encoder+decoder) layers. Language-pair adapters, introduce $C_{\text{pair}} = N^2 \cdot L \cdot C_{\text{block}}$ new parameters in a multi-parallel many-to-many setting, or $C_{\text{pair}} = 2N \cdot L \cdot C_{\text{block}} = 2 \cdot C_{\text{lang}}$ new parameters in an English-centric[8] many-to-many setting.

Hyper-Adapters A benefit of hyper-adapters, is that their number of parameters is invariant to both the number of languages N and layers L. Most of the parameters are in the projection heads of the hyper-network. Intuitively, each row of a head's weight matrix is equivalent to a (flattened) adapter weight matrix. The number of rows in a head is equal to hidden size $\boldsymbol{d_h}$ of the hyper-network. Therefore, $\boldsymbol{d_h}$ controls the hyper-network capacity:

$$C_{\text{hyper}} = \underbrace{\boldsymbol{d_h}(d_z d_b)}_{\text{Head-down}} + \underbrace{\boldsymbol{d_h}(d_b d_z)}_{\text{Head-up}} = \boldsymbol{d_h} \cdot C_{\text{block}}$$

For example, in a dataset with $N = 50$ languages with a Transformer model with total $L = 12$ layers, language adapters introduce 600 adapter blocks. If we set $\boldsymbol{d_h} = 600$, then hyper-adapters introduce the same number of parameters, whereas using $\boldsymbol{d_h} < N \cdot L$ yields parameter savings. The parameter savings with respect to language adapters are $\frac{d_h}{N \cdot L}$, and to *English-centric* pair-adapters are $\frac{d_h}{2N \cdot L}$. The hyper-network embeddings and encoder contain a comparatively negligible amount of parameters.

C Additional Results

This section contains additional results for the main experiments in Section 5 with the Transformer-Base models. Table 8 shows results measured with ChrF (Popović, 2015). Overall, we observe that the results are consistent with the BLEU scores reported in Table 1 in the main paper. In Tables 9 and 10, we report the non-aggregated BLEU scores for the en→X and X→en pairs, respectively.

[8] Concatenation of English→X and X→English directions.

Language	Transformer-Base	+adapters				
		lang	pair	hyper (base)	hyper (base)	hyper (base)
en→af	17.1	16.1	15.7	16.2	15.6	15.4
en→ar	11.5	13.4	14.0	12.5	13.3	14.0
en→az	6.8	7.1	7.3	7.9	8.1	7.5
en→bn	11.2	13.0	12.5	11.0	11.9	12.5
en→cs	19.6	20.6	20.7	20.8	20.8	21.2
en→de	34.3	35.2	36.0	36.1	36.6	36.8
en→es	26.6	28.9	28.5	28.6	29.1	29.3
en→et	15.8	16.8	17.5	17.0	17.1	17.8
en→fa	14.3	15.2	15.8	14.9	15.2	16.2
en→fi	16.8	17.7	18.7	17.8	18.9	18.9
en→fr	34.1	34.4	35.1	35.0	35.4	35.2
en→gl	25.0	26.0	23.3	25.8	26.8	26.3
en→gu	0.4	0.4	0.3	0.2	0.1	0.2
en→he	22.6	25.2	26.5	24.1	25.4	27.2
en→hi	14.9	15.8	16.9	16.5	16.7	17.1
en→hr	25.0	28.1	28.5	27.0	28.4	29.2
en→id	29.6	32.4	32.3	31.5	32.5	33.2
en→it	30.1	32.0	32.7	31.8	32.9	34.3
en→iu	14.3	14.5	14.9	14.5	14.6	15.1
en→ja	14.4	14.5	14.5	15.0	14.5	15.6
en→ka	11.0	12.5	11.5	11.5	12.1	12.9
en→kk	4.5	4.7	4.9	3.8	5.0	5.2
en→km	0.0	0.1	0.1	0.0	0.1	0.1
en→ko	5.2	6.0	6.0	5.6	5.9	6.3
en→lt	11.2	12.2	11.9	12.0	12.6	12.6
en→lv	13.9	14.7	15.5	15.2	15.6	16.1
en→mk	23.9	25.8	24.8	24.1	25.6	26.7
en→ml	4.7	4.9	5.8	4.9	4.9	5.7
en→mn	6.7	8.0	7.3	7.8	7.7	8.2
en→mr	9.0	12.0	11.4	10.2	11.4	11.8
en→my	21.5	22.4	21.9	21.9	22.5	23.0
en→ne	6.1	5.9	5.4	6.2	6.4	5.8
en→nl	26.5	29.4	29.8	28.5	29.6	30.6
en→pl	19.8	20.6	20.6	20.8	21.4	21.4
en→ps	6.1	6.5	7.0	6.6	6.6	7.6
en→pt	34.4	37.8	37.6	36.7	38.2	39.0
en→ro	22.1	23.7	24.2	23.5	24.2	24.5
en→ru	22.4	23.1	23.7	24.0	24.4	24.1
en→si	0.8	1.7	2.0	1.0	1.5	2.4
en→sl	19.2	20.8	20.5	20.5	21.4	21.8
en→sv	30.5	34.9	35.6	33.3	34.8	35.9
en→ta	6.3	6.6	6.8	6.4	6.9	7.0
en→te	21.5	24.1	25.9	21.9	22.8	23.9
en→th	16.8	18.6	19.2	17.4	18.3	19.7
en→tr	14.1	15.5	16.5	15.5	16.2	16.8
en→uk	20.3	21.6	21.2	21.5	21.9	22.3
en→ur	13.9	17.9	18.7	14.6	16.5	18.9
en→vi	27.1	28.6	29.2	28.3	28.6	30.0
en→xh	12.4	12.7	12.2	12.8	12.8	12.9
en→zh	24.1	25.6	25.9	25.7	26.4	26.5

Table 9: BLUE (↑) scores of the *Transformer-base* models on the **en**→**X** pairs of ML50.

Language	Transformer-Base	+adapters				
		lang	pair	hyper (base)	hyper (base)	hyper (base)
a→en	27.4	26.0	26.0	30.3	29.1	27.5
ar→en	29.8	32.7	32.6	31.1	31.9	33.0
az→en	15.1	15.1	14.7	15.8	16.1	15.3
bn→en	17.6	17.9	16.5	19.6	18.8	19.9
cs→en	26.5	27.3	26.4	27.5	27.5	27.5
de→en	35.6	37.1	36.6	36.4	36.7	37.4
es→en	29.1	30.5	27.5	29.1	29.6	29.0
et→en	22.8	24.2	23.1	24.0	24.3	24.8
fa→en	27.3	30.5	29.3	28.2	28.6	30.6
fi→en	22.7	24.9	23.9	24.2	24.8	25.1
fr→en	33.9	34.6	34.0	34.8	34.9	35.0
gl→en	34.7	33.7	33.5	35.9	35.7	34.5
gu→en	2.3	1.1	0.9	2.4	2.3	2.2
he→en	35.4	38.6	37.9	36.1	36.8	38.4
hi→en	20.5	20.9	20.4	21.4	21.2	21.8
hr→en	37.0	40.1	39.6	38.5	38.9	39.9
id→en	31.5	34.9	34.5	33.1	33.7	34.7
it→en	36.9	39.5	39.2	38.2	39.0	39.9
iu→en	24.5	26.7	26.4	25.1	25.7	27.3
ja→en	14.4	15.6	15.2	15.6	15.6	15.8
ka→en	23.2	23.3	21.5	23.0	23.6	23.6
kk→en	13.3	12.8	12.1	12.5	13.4	13.3
km→en	6.3	6.7	5.7	6.3	6.0	7.3
ko→en	15.6	17.2	16.9	16.5	16.3	17.4
lt→en	25.7	28.0	27.6	26.9	27.1	27.8
lv→en	17.9	19.1	19.0	19.0	19.2	19.2
mk→en	35.6	37.0	36.1	36.9	36.8	37.0
ml→en	13.5	16.0	15.7	14.0	14.5	15.5
mn→en	10.5	11.0	11.3	10.8	11.9	11.6
mr→en	12.8	13.1	12.3	13.7	13.7	14.0
my→en	24.6	25.5	24.2	25.7	25.6	26.2
ne→en	15.7	13.9	13.6	15.8	15.3	14.2
nl→en	32.9	35.1	34.8	33.9	34.3	35.4
pl→en	25.9	26.4	26.1	26.7	27.2	26.7
ps→en	11.0	10.8	13.2	12.2	12.4	12.8
pt→en	42.4	44.8	44.1	44.0	44.3	45.2
ro→en	31.5	34.1	33.5	32.9	33.7	34.8
ru→en	34.2	35.2	34.8	34.5	35.1	35.3
si→en	8.4	10.4	10.0	8.8	9.3	10.3
sl→en	28.1	28.4	28.7	29.6	28.9	29.8
sv→en	39.9	43.2	42.6	41.2	42.3	43.1
ta→en	15.1	16.0	15.6	16.2	16.2	15.9
te→en	30.3	33.8	32.8	30.9	31.7	33.8
th→en	23.8	25.8	24.8	24.2	24.7	25.5
tr→en	18.4	20.4	20.2	19.5	19.6	20.3
uk→en	30.0	31.6	31.2	30.7	31.3	32.0
ur→en	22.4	24.6	24.5	22.3	23.8	25.0
vi→en	26.4	28.1	27.8	27.5	27.8	28.0
xh→en	12.2	11.7	11.5	12.0	12.6	12.6

Table 10: BLUE (↑) scores of the *Transformer-base* models on the **X**→**en** pairs of ML50.