Multilinguality and MT

Jörg Tiedemann
Raúl Vázquez
Timothee Mickus
Outline

- **Why?**
  - The blessings of multilinguality
  - The curse of multilinguality

- **How?**
  - The MAMMOTH framework
  - Parameter sharing & modularity
  - Scaling up & parallelization

- **Get involved (at the MT marathon and beyond)**
Combinatorial explosion
Pivot-based approach
Neural “interlingua”
Multilingual NMT

One model with completely shared parameters
Multilingual NMT

Modular model with partially-shared parameters and language-specific components
The Blessings of Multilinguality
The language continuum and language embeddings

Back in 2016:

1303 Bible translations into 990 languages

Continuous multilinguality with language vectors

Robert Östling
Department of Linguistics*
Stockholm University
robert@ling.su.se

Jörg Tiedemann
Department of Modern Languages
University of Helsinki
jorg.tiedemann@helsinki.fi

Abstract

Most existing models for multilingual natural language processing (NLP) treat language as a discrete category, and make predictions for either one language or the other. In contrast, we propose using continuous vector representations of language. We show that these can be learned separately for each language. This presupposes large quantities of monolingual data in each of the languages that needs to be covered and each model with its parameters is completely independent of any of the other models.

We propose instead to use a single model with real-valued vectors to indicate the language used, and to train this model with a large number of languages. We thus get a language model whose
Continuous multilinguality with language vectors

Interpolating between the English language vector and the German language vector

(cross-entropy for English)
Continuous multilinguiality with language vectors

Interpolating between Modern English and Middle English

(cross-entropy for English from the 17th century)
Continuous multilinguality with language vectors

<table>
<thead>
<tr>
<th>%</th>
<th>Random sample</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(temperature parameter $\tau = 0.5$)</td>
</tr>
<tr>
<td>30</td>
<td>and thei schulen go in to alle these thingis, and schalt endure bothe in the weie</td>
</tr>
<tr>
<td>40</td>
<td>and there was a certaine other person who was called in a dreame that he went into a mountaine.</td>
</tr>
<tr>
<td>44</td>
<td>and the second sacrifice, and the father, and the prophet, shall be given to it.</td>
</tr>
<tr>
<td>48</td>
<td>and god sayd, i am the light of the world, and the powers of the enemies of the most high god may find first for many.</td>
</tr>
<tr>
<td>50</td>
<td>but if there be some of the seruants, and to all the people, and the angels of god, and the prophets</td>
</tr>
<tr>
<td>52</td>
<td>then he came to the gate of the city, and the bread was to be brought</td>
</tr>
<tr>
<td>56</td>
<td>therefore, behold, i will lose the sound of my soul, and i will not fight it into the land of egypt</td>
</tr>
<tr>
<td>60</td>
<td>and the man whom the son of man is born of god, so have i therefore already sent to the good news of christ.</td>
</tr>
</tbody>
</table>
Continuous multilinguality with language vectors

Control text generation with language embeddings:

**turn on Swedish:**

> och jehova sade till honom : ”jehova har sagt , och jag skall ..."

**turn on German:**

> und er sprach zu ihnen : siehe , ich bin der herr

**mix Swedish and German:**

> vocken änner vocken ånnen söhenöckenföcken ...

**average of Scandinavian languages:**

> og han sa til herrens : " han skal vitnaðus til herrens hjárt
At the same time: Johnson et al. for multilingual NMT

Same idea about learning language embeddings and it works!

Add a special token that tells the system to translate into a particular language. (early type of “prompting”)

<2uk> Hello world!
Transfer learning with skewed training data
Examples of successful transfer learning

<table>
<thead>
<tr>
<th>Model / test set</th>
<th>Belarusian $\rightarrow$ English</th>
<th>English $\rightarrow$ Belarusian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belarusian - English</td>
<td>10.0</td>
<td>8.2</td>
</tr>
<tr>
<td>East Slavic languages - English</td>
<td>38.7</td>
<td>20.8</td>
</tr>
<tr>
<td>Slavic languages - English</td>
<td>42.7</td>
<td>22.9</td>
</tr>
</tbody>
</table>

BLEU scores (in %)
Examples of successful transfer learning

multilingual model: Celtic languages and English
Zero shot translation directions
Zero-shot translation in massively multilingual models

Test case:
Ukrainian - Russian

Better generalization in highly multilingual models

Treating paraphrasing as zero-shot translation

Learning curves during training:

- en-fr validation perplexity
- en-en paraphrase perplexity

- English-French
- All languages

learn to recognize paraphrased sentences
Handle mixed and non-standard language input
Multilingual NMT for text normalisation

Valsch geschreibt is nich gut!
Das Pferd hat gelaufen.
Ich haben fertig.
Wir sein kommen.
wat morkelst du denn da rum?
Icke geb dir dann och noch wat zu trinken.
Dat is nix für meinereiner!
Mein Fuß ist brechen! Ich muss nach die dokter.

The Curse of Multilinguality
(1) Limits of training data

Size of bitexts in OPUS

Languages aligned to English

(slightly outdated counts)
Very skewed towards English-centric data and tasks

| language | files | tokens | sentences | bg | ca | cs | da | de | el | en | es | et | eu | fi | fr | gl | hr | hu | is | it | km | nl | no | pl | pt | ro | ru | sk | sl | so | sv | tl |
|----------|-------|--------|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|          |       |        |           | 2.39 | 240.4M | 12.9M | 39.0M | 53.5M | 41.9M | 261.1M | 34.8M | 2.3M | 696.5M | 6.8M | 11.1M | 12.7M | 5.7M | 120.1M | 65.1K | 4.0M | 8.0M | 8.2M | 1.6M | 31.4K | 92.1K | 98.5M | 59.1M | 2.7M |

WMT news translation tasks

Both directions
- Chinese to/from English
- German to/from English: document-level
- Hebrew to/from English: low-resource
- Japanese to/from English
- Russian to/from English
- Ukrainian to/from English

Single direction
- Czech to Ukrainian: non-English
- English to Czech

ParaCrawl
Realistic MT data sets with **large language coverage** (currently: 557 languages)

- No artificial low-resource scenarios
- Straightforward to use (train/dev/test splits)
- Consistent language labels + writing script information

**Benchmarks**

- Tatoeba collection of user-contributed translations
- Continuously updated
(2) Limits of generalisations & transfer learning in NMT

<table>
<thead>
<tr>
<th>Model / test set</th>
<th>Belarusian → English</th>
<th>English → Belarusian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belarusian - English</td>
<td>10.0</td>
<td>8.2</td>
</tr>
<tr>
<td>East Slavic languages - English</td>
<td>38.7</td>
<td>20.8</td>
</tr>
<tr>
<td><strong>Slavic languages - English</strong></td>
<td><strong>42.7</strong></td>
<td><strong>22.9</strong></td>
</tr>
<tr>
<td>Indo-European languages - English</td>
<td>41.7</td>
<td>18.1</td>
</tr>
</tbody>
</table>

(BLEU scores (in %))

(increasing language coverage while keeping the model size constant)
Zero-shot translation in massively multilingual models

Test case:
Ukrainian - Russian

Sweet spots of multilinguality?

(3) Limits of the model capacity

Testing the model capacity when adding more languages
(similar patterns for adding languages in random order or according to typological relationship)

(from “Continuous multilinguality with language vectors”)
(3) Limits of the model capacity

Comparing the impact of
- adding languages
- decreasing model size

(from “Continuous multilinguality with language vectors”)
Growing model size for multilingual models

Nr of parameters (in millions)

- Transformer base
- Transformer big
- NLLB
Growing model size for multilingual models

Bar chart showing the number of parameters (in millions) for different models: transformer base, transformer big, and NLLB. NLLB has the highest number of parameters, followed by transformer big, and then transformer base. The chart notes that NLLB is a mixture of experts.
Back to Modularity
Multilingual NMT

Modular model with partially-shared parameters and language-specific components
At the MT marathon 5 years ago …

NN Decoders:

EN  FR  CS  DE

minimal shared component

Attention bridge

shared meaning representation matrix

NN Encoders:

EN  FR  CS  DE

handy, small-sized language-specific modules
At the MT marathon 5 years ago …

NN Decoders:
- language
- independent
- meaning
- representation

Attention bridge
- shared meaning representation matrix

NN Encoders:
- FR

CS
- plug & play with small and efficient modular models
What happened since then?

Modular NLP is increasingly popular

- Partial sharing and task-specific components
- Adapters and hyper networks
- Gated routing and mixture of experts (see NLLB and GPT-4)

Many open questions

- What to share and how much?
- Hierarchical models and language clusters?
- Efficient training with optimal routing and communication
What happened since then?

Modular NLP is increasingly popular

- Partial sharing and task-specific components
- Adapters and hyper networks
- Gated routing and mixture of experts (see NLLB and GPT-4)

Many open questions

- What to share
- Hierarchical models and language clusters
- Efficient training with optimal routing and communication
What we want from a scalable mNMT system

Must-have Feats of a Scalable mNMT System:

● Allow for versatile parameter sharing
● Efficient GPU allocation
● Supports addition of new language pairs
● Provide tools for mNMT data management
● Efficient inference
What to do?

Use our toolkit(*)(**) to train modular systems:

**MAMMOTH: MAssively Multilingual Modular Open Translation @ Helsinki**

https://github.com/Helsinki-NLP/mammoth

* still a beta version
** we built on top of OpenNMT-py but the code has changed so much that it cannot be named after the parent codebase
The lore

The inner-attention bridge model

Language specific decoders

EN ES DE FR

attention bridge

m, m, m, ... m

RelU

W

K

MTM 2018

Thirteenth Machine Translation Marathon, Prague, September 30th-6th, 2018

Latest Development in the FoTran Project – Scaling Up Language Coverage in Neural Machine Translation Using Distributed Training with Language-Specific Components

Raúl Víquez*, Michele Boggia*, Alessandro Rogantini*, Niki A. Lopatá, Sigurður Grétarsson, Jorge Tedeschi*

* University of Helsinki, Finland & NVIDIA

† NVIDIA & Sílo-AI

{raul.viquez@helsinki.fi, niki.lopata@nvidia.com, alessandro.rogantini@nvidia.com, sigurdur.gretarsson@nvidia.com, jorge.tedeschi@nvidia.com}
Scalable mNMT Systems

Features

- Parameter sharing
- GPU allocation & communication
- Add new languages
- mNMT data
- Efficient inference

The ABCD of mNMT

A) Anatomy of parameter sharing
B) Bridges and structures for sharing
C) Communication chaos
D) Dearth of data
Scalable mNMT Systems

Features

- Parameter sharing
- GPU allocation & communication
- Add new languages
- mNMT data
- Efficient inference

The ABCD of mNMT

A) Anatomy of parameter sharing
B) Bridges and structures for sharing
C) Communication chaos
D) Dearth of data

We showcase mNMT using MAMMOTH
A) Anatomy of parameter sharing

- Parameter sharing is tied to transfer learning
- The trichotomy of this choice:
  - full sharing,
  - no sharing, and
  - everything in between
A) Anatomy of parameter sharing

Fully shared (Johnson et al., 2017)

- Simple and effective
- Use of language tags (prompt-based learning predecessor; Liu et al., 2023)

<2tgt> The rise of the radical right across Europe is a symptom of a failing capitalism.
A) Anatomy of parameter sharing

No shared parameters (Escolano et al., 2021)

- Exploits the encoder-decoder NMT architecture
- Increases the data (& its distribution) used to train encoder/decoder modules
- Easy to add modules (no need to re-train)
A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

<table>
<thead>
<tr>
<th>Transversal</th>
<th>Longitudinal</th>
</tr>
</thead>
</table>

  | Embeddings  | or vocab hacks |
A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

  Transversal

  Longitudinal

  Embeddings
  or vocab hacks
A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

  Transversal | Longitudinal
  (e.g., Purason & Tätter, 2022) | Embeddings or vocab hacks

- Embeddings or vocab hacks
A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

  Transversal
  (e.g., Purason & Tätter, 2022)

  Longitudinal
  (e.g., LaSS Lin et al., 2021)

Embeddings or vocab hacks
A) Anatomy of parameter sharing

Partial sharing schemes (or “everything in between”):

- Myriad of research works
- Roughly, we classify into:

  Transversal
  (e.g., Purason & Tättar, 2022)

  Longitudinal
  (e.g., Lin et al., 2021)

Embeddings or vocab hacks

Share all vocabs:
- naively (Johnson et al., 2017)
- Temp.-based sampling (Aharoni, 2018)
Dynamic vocab adaptation (Lakew, 2018)
Vocab substitution in incremental training (Chronopoulou, 2020; Garcia, 2021; Huang, 2022)

- naively (Johnson et al., 2017)
- Temp.-based sampling (Aharoni, 2018)
Dynamic vocab adaptation (Lakew, 2018)
Vocab substitution in incremental training (Chronopoulou, 2020; Garcia, 2021; Huang, 2022)
A) Anatomy of parameter sharing

We break down mNMT training into a series of smaller “tasks”

- A task requires specific modules
- A task is done on a specific device
- A task corresponds to a specific (parallel) corpus

A centralized manager handles tasks synchronization
### A) Anatomy of parameter sharing

- **full sharing**

```python
data:

<table>
<thead>
<tr>
<th>dataset</th>
<th>src_tgt</th>
<th>enc_sharing_group</th>
<th>dec_sharing_group</th>
<th>path_src</th>
<th>path_tgt</th>
<th>transforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>train_ar-ar</td>
<td>ar-ar</td>
<td>&quot;all&quot;</td>
<td>&quot;all&quot;</td>
<td>/path/to/train.ar</td>
<td>/path/to/train.ar</td>
<td>[filtertoolong, bart]</td>
</tr>
<tr>
<td>train_ar-en</td>
<td>ar-en</td>
<td>&quot;all&quot;</td>
<td>&quot;all&quot;</td>
<td>/path/to/train.ar</td>
<td>/path/to/train.en</td>
<td>[filtertoolong]</td>
</tr>
<tr>
<td>train_en-ar</td>
<td>en-ar</td>
<td>&quot;all&quot;</td>
<td>&quot;all&quot;</td>
<td>/path/to/train.en</td>
<td>/path/to/train.ar</td>
<td>[filtertoolong]</td>
</tr>
<tr>
<td>train_en-en</td>
<td>en-en</td>
<td>&quot;all&quot;</td>
<td>&quot;all&quot;</td>
<td>/path/to/train.en</td>
<td>/path/to/train.en</td>
<td>[filtertoolong, bart]</td>
</tr>
</tbody>
</table>
```
A) Anatomy of parameter sharing

- full sharing

- no sharing

```python
data:
train_ar-ar:
  src_tgt: ar-ar
  enc_sharing_group: ["ar"]
  dec_sharing_group: ["ar"]
  path_src: /path/to/train.ar
  path_tgt: /path/to/train.ar
  transforms: [filtertoolong, bart]

train_ar-en:
  src_tgt: ar-en
  enc_sharing_group: ["ar"]
  dec_sharing_group: ["en"]
  path_src: /path/to/train.ar
  path_tgt: /path/to/train.en
  transforms: [filtertoolong]

train_en-ar:
  src_tgt: en-ar
  enc_sharing_group: ["en"]
  dec_sharing_group: ["ar"]
  path_src: /path/to/train.en
  path_tgt: /path/to/train.ar
  transforms: [filtertoolong]

train_en-en:
  src_tgt: en-en
  enc_sharing_group: ["en"]
  dec_sharing_group: ["en"]
  path_src: /path/to/train.en
  path_tgt: /path/to/train.en
  transforms: [filtertoolong, bart]
```
A) Anatomy of parameter sharing

- full sharing
- no sharing
- and everything in between
A) Anatomy of parameter sharing

This relies on an implementation of *coders as stacks of layer stacks

def forward(self, src, lengths=None, **kwargs):
    # wrapper embeds src and creates mask
    emb = self.embeddings(src)
    emb = emb.transpose(0, 1).contiguous()
    mask = ~sequence_mask(lengths).unsqueeze(1)
    output = emb
    for active_id, stacks in zip(self._active, self.encoders):
        encoder = stacks[active_id]
        # Throw away emb, lengths, mask
        _, output, _, _ = encoder.forward(
            output,
            lengths=lengths,
            skip_embedding=True,
            mask=mask,
        )
    # only at the end transpose back to timestep-first
    output = output.transpose(0, 1).contiguous()
    return emb, output, lengths, mask
A) Anatomy of parameter sharing

We can define stacks with arbitrary number of layers

```python
rnn_size: 512
word_vec_size: 512
transformer_ff: 2048
heads: 8
enc_layers: [5, 1]
dropout: 0.1

stacks[module_id] = AdaptedTransformerEncoder(
  n_layers,
  model_opt.enc_rnn_size,
  model_opt.heads,
  model_opt.transformer_ff,
  model_opt.dropout[6] if type(model_opt.dropout) is list else model_opt.dropout,
  (
    model_opt.attention_dropout[0]
    if type(model_opt.attention_dropout) is list
    else model_opt.attention_dropout
  ),
  None, # embeddings,
  model_opt.max_relative_positions,
  pos_ffn_activation_fn=model_opt.pos_ffn_activation_fn,
)```
A) Anatomy of parameter sharing

Our research:

Effects of **Sharing more parameters VS Adding lang. pairs** on (Boggia et al., 2023):

- Task fitness OR *how adequate they are for machine translation?*
- Lang. independence OR *how independent of the source language they are?*
- Semantic content OR *what semantic information they convey?*
A) Anatomy of parameter sharing

Sharing more parameters

Adding lang. pairs

∀ language $L$ in a dataset, we consider 3 translation directions:

1. $L$-to-English,
2. English-to-$L$, and
3. $L$-to-$L$ denoising auto-enc.
A) Anatomy of parameter sharing

Conclusions **Sharing more parameters VS Adding lang. pairs** (Boggia et al., 2023)

- MAMMOTH made this study possible

Broadly: Sharing parameters and multiplying languages affect differently multilingual NLP systems.

Setting right the number of shared parameters brings higher performances and more reliable representations, but the optimal number of shard layers depends on the task.

- Go read the paper for more ;)

A) Anatomy of parameter sharing

Conclusions **Sharing more parameters VS Adding languages**

- MAMMOTH made this study possible

Broadly: Sharing parameters and multiplying languages affect differently multilingual NLP systems.

Setting right the number of shared parameters brings higher performances and more reliable representations, but the optimal number of shard layers depends on the task.

- Go read the paper for more ;)

Pikku SPOILER

**DO:** Spend effort on tuning the level of parameter sharing for the task at hand.

**DON'T:** Spend too much effort on acquiring data for additional languages
B) Bridges and structures for sharing

What structure we want to explore for the shared parameters???

- fully shared layers
  - Transformer-based
  - FFWD
  - Attention bridge (remember the lore?)
  - RNN, CNN?
- Adapters (Bapna & Firat, 2019)
The attention bridge model

Architecture proposed by Cifka and Bojar (2018). Our implementation in OpenNMT-py (MTM2018)
B) Bridges and structures for sharing

We saw stacks, we also have:

- **bridges**, shared across all tasks

```
class PerceiverAttentionBridgeLayer(BaseAttentionBridgeLayer):
    return alphas, self_attention_output

class LinAttentionBridgeLayer(BaseAttentionBridgeLayer):
    return True

class SimpleAttentionBridgeLayer(BaseAttentionBridgeLayer):
    return
```

straightforward configuration

```
ab_layers: ['feedforward', 'lin', 'transformer']
hidden_ab_size: 512
ab_fixed_length: 50
```

Multiple architectures

```
# TODO: for now I've used the basic implementation of TransformerEncoderLayer;
# we could consider an attention-bridge-specific implementation that would allow
# us to control the norm and return alphas if necessary.
class TransformerAttentionBridgeLayer(BaseAttentionBridgeLayer, TransformerEncoderLayer):
    return
```
B) Bridges and structures for sharing

We saw stacks, we also have:

- bridges, shared across all tasks
- **adapters** for finer-grained sharing

```python
> class TransformerAdapterMixin:
>     return result

> class AdaptedTransformerEncoder(TransformerAdapterMixin, TransformerEncoder):
>     return result

> class AdaptedTransformerDecoder(TransformerAdapterMixin, TransformerDecoder):
>     return result
```
B) Bridges and structures for sharing

Define language groups and shared components through a language distance matrix

<table>
<thead>
<tr>
<th>lang</th>
<th>en</th>
<th>de</th>
<th>fr</th>
<th>zh</th>
</tr>
</thead>
<tbody>
<tr>
<td>en</td>
<td>0</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>de</td>
<td>0.1</td>
<td>0</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>fr</td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>zh</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Group 1: {en, de, fr}
Group 2: {zh}

```python
group_idx = AgglomerativeClustering(
    n_clusters=n_groups,
    metric='precomputed',
    linkage='average',
    distance_threshold=cutoff_threshold,
).fit_predict(distance_matrix['data']).tolist()
groups = {lang: f'group(idx)' for lang, idx in zip(distance_matrix['header'], group_idx)}
```
B) Bridges and structures for sharing

What structure we want to explore for the shared parameters???

- fully shared layers
  - Transformer-based
  - FFWD
  - Attention bridge (remember the lore?)
  - RNN, CNN?
- Adapters (Bapna & Firat, 2019)

More coming soon!
This comparison is current ongoing work... we submitting to WMT!!!
C) Communication chaos

Scaling-up a mNMT to a (very) large number of languages must:

- Deal with the task2gpu allocation problem
- Allow for custom model parallelism across nodes and GPUs
C) Communication chaos

- The task2gpu allocation problem
C) Communication chaos

- The task2gpu allocation problem in MAMMOTH*

1/ define costs for factors to weigh in

```plaintext
INTER_NODE_COST = 5
INTRA_NODE_COST = 1
HOMOGENEITY_WEIGHT = 0.1
READY_TO_START_WEIGHT = 0.5
UNASSIGNED_WEIGHT = 100
UNASSIGNED_PREFER_MASTER = 10
SPLIT_LPS_WEIGHT = 50
NOT_READY_TO_START = 500
VERY_BAD = 99999999
```

2/ randomly assign language pairs to gpu slots

* currently a beta feature
C) Communication chaos

```python
def swap_all_slots_once(slot, assignment, current_cost):
    for i, slot_a in enumerate(slot):
        current_cost, assignment = self.swap_for(slot_a, assignment, current_cost)
        print('o', end='', flush=True)
    return current_cost, assignment

def optimize(slot, assignment, current_cost, iterations=10, patience=1):
    prev_cost = None
    stalled = 0
    print(f'initial cost: {current_cost}', flush=True)
    for i in range(iterations):
        current_cost = current_cost
        current_cost, assignment = self.swap_all_slots_once(assignment, current_cost)
        print(f'iteration {i} cost: {current_cost}', flush=True)
        if prev_cost == current_cost:
            stalled += 1
        else:
            stalled = 0
            if stalled > patience:
                print('No improvement. finishing early', flush=True)
                break
    return current_cost, assignment
```

3/ Move all pairs to their best possible option

4/ rinse and repeat until stabilization
C) Communication chaos

Custom model parallelism across nodes and GPUs

- In short, modules allocated in more than 1 GPU have to be synced at all times.
C) Communication chaos

Custom model parallelism increases param. sharing versatility

- Modules are synchronized in the GPUs where they are present:
  - AB layer synced in GPUs 1,3&4
  - Language-specific components synced as needed (e.g., EN-decoder in all GPUs)
  - Language group-specific components also synced as needed (e.g., GER in GPUs 1,2&3)
C) Communication chaos

Custom model parallelism increases param. sharing versatility & inference efficiency

- All modules are saved independently
- Light inference. E.g., DE->FR only loads
C) Communication chaos

At training time, *coder communication is based on layer stacks (and adapters)*

```python
self._gradient_accumulation_over_lang_pairs:
    batches_with_meta,
    total_stats,
    report_stats,
}

# Note that all group ids are tuples, some with length 1
for (layer_stack_index, encoder_id), (_, group) in self.my_encoder_groups.items():
    params = [
        (name, p) for (name, p)
        in self.model.encoder.get_submodule(layer_stack_index, encoder_id).named_parameters()
        if 'embeddings' not in name and 'adapter' not in name
    ]
    onmt.utils.distributed.only_ready_reduce_and_rescale_grads(params, group=group)

for (layer_stack_index, decoder_id), (_, group) in self.my_decoder_groups.items():
    params = [
        (name, p) for (name, p)
        in self.model.decoder.get_submodule(layer_stack_index, decoder_id).named_parameters()
        if 'embeddings' not in name and 'adapter' not in name
    ]
    onmt.utils.distributed.only_ready_reduce_and_rescale_grads(params, group=group)
```

* Thanks to CSC infrastructure :)

---

* We have tested comms in CUDA and ROCM!!*
C) Communication chaos

We only broadcast the gradient for modules currently in use

```python
ready_list = []
for name, p in require_grad:
    if hasattr(p, 'has_grad') and p.has_grad:
        ready_list.append(1.0)
    else:
        ready_list.append(0.0)
    if p.grad is None:
        p.grad = torch.zeros_like(p)

    # Communicate the ready bits, and reduce them using summation.
    # This gives the number of non-dummy gradients participating, for normalization
    ready_t = torch.tensor(ready_list).to(device)
    if group is None:
        torch.distributed.all_reduce(ready_t)
    else:
        torch.distributed.all_reduce(ready_t, group=group)
rescale_denoms = ready_t  # after reduction
```

* Thanks to CSC infrastructure :)
D) Dearth of data

Remember?

(1) Limits of training data

Size of bitexts in OPUS in millions of tokens

Languages aligned to English
D) Dearth of data

- Complex sharing structures in multi-node & -GPU settings make random task scheduling tricky

- Schedulers for
  - Delaying the start of specific tasks (start with overrepresented langpairs)
  - Early stopping of specific tasks (to better fine-tune underrepresented langpairs)

- Sampling schemes
  - Weight-based (sample more from high-res langpairs) / Round-robin based
  - More to come
Why use it?

MAMMOTH has been tailored for (massively) multilingual NMT:

1. Scales well to many tasks

2. Allows different param. sharing structures

3. It is modular ⇒ ☑️ big at training time
   ☑️ inference is lightweight
Sneak peek

Our MTM23 Project: Integration to HF 😊

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers
Any question?
MAMMOTH goes huggingface

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers
MAMMOTH goes huggingface

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers