Bachelor Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Multitask Learning for NLP Classifiers

Danil Semin

Supervisor: Ing. Jan Drchal, Ph.D.
Study program: Open Informatics
Specialisation: Artificial Intelligence and Computer Science
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Semin Danil**	Personal ID number:	**503160**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Multitask Learning for NLP Classifiers

Bachelor’s thesis title in Czech:

Víceúlohové učení pro NLP klasifikátory

Guidelines:

The aim of this project is to improve performance of existing neural text classifiers by fine-tuning them on a set of diverse yet related tasks choosing available datasets based even on diverse languages.
1) Research state-of-the-art methods of multi task learning as well as modern neural architectures for text classification.
2) Compile a multilingual selection of datasets for tasks related to Natural Language Inference (NLI).
3) Decide a methodology to perform the multitask learning and evaluation of the fine-tuned models.
4) Perform experiments comparing contribution of different tasks and languages.
5) Evaluate the most promising approach on CTKFactsNLI dataset provided by the supervisor.

Bibliography / sources:

[1] Worsham, Joseph, and Jugal Kalita. "Multi-task learning for natural language processing in the 2020s: where are we going?." Pattern Recognition Letters 136 (2020): 120-126.
[2] Zhang, Zhihan, et al. "A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods." arXiv preprint arXiv:2204.03508 (2022).
[3] Minaee, Shervin, et al. "Deep learning--based text classification: a comprehensive review." ACM Computing Surveys (CSUR) 54.3 (2021): 1-40.
[4] Ullrich, Herbert, et al. "CsFEVER and CTKFacts: Acquiring Czech data for fact verification." arXiv e-prints (2022): arXiv-2201.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Drchal, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **31.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Jan Drchal, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to thank my supervisor, Ing. Jan Drchal, Ph.D., for an opportunity to explore exciting concepts in this project and for providing valuable guidance during moments when I required it the most.

I am grateful to my family for their patience, care and support at all times.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 26 May 2023

Abstract

Recent advancements in the field of natural language processing (NLP) have led to more generalized neural language models that effectively tackle multiple tasks across diverse natural languages simultaneously. These advancements are driven by the development of novel neural architectures, the utilization of multilingual pre-training methods, and the deployment of multitask learning (MTL). Building upon these achievements, this thesis focuses on improving the performance of existing multilingual neural text classifiers by leveraging MTL methods, resulting in the development of models that competitively handle 6 tasks in 17 languages and achieve state-of-the-art results on the Czech fact-checking task, CTKFactsNLI.

Keywords: NLP, MTL, text classification, NLI, task embeddings, LoRA, multilingual

Supervisor: Ing. Jan Drchal, Ph.D.

Abstrakt

Nedávné inovace v oblasti zpracování přirozeného jazyka (anglicky natural language processing, NLP) umožnily vytvoření obecnějších neuronových jazykových modelů, které efektivně řeší více úloh v různých přirozených jazycích najednou. Tyto pokroky jsou výsledkem vývoje nových neuronových architektur, využívání metod vícejazyčného předtrénování a uplatnění víceúlohového učení (anglicky multitask learning, MTL). V návaznosti na tyto úspěchy se tato práce zaměřuje na zlepšení výkonu stávajících vícejazyčných neuronových klasifikátorů textů s využitím metod MTL, což vede k vývoji modelů, které kompetitivně řeší 6 úloh v 17 jazycích a dosahují tzv. state-of-the-art výsledků v úloze ověřování faktů (anglicky fact checking) v češtině, ČTKFactsNLI.

Klíčová slova: NLP, MTL, text classification, NLI, task embeddings, LoRA, multilingual

Překlad názvu: Víceúlohové učení pro NLP klasifikátory

Contents

Chapter 1

Introduction

Humans can naturally learn multiple tasks simultaneously and leverage the knowledge learned in one task to help the learning of another task. For example, it is common for humans to learn multiple natural languages throughout their lives. By doing that, they acquire skills for communication in new languages and refine their knowledge of the languages they already speak. Inspired by this analogy, multitask learning (MTL) [1], a machine learning (ML) paradigm, aims to learn multiple tasks in parallel in order to improve the generalization performance. In the natural language processing (NLP) field, neural language models' generalization capabilities have rapidly grown in recent years. These models are now able to solve multiple tasks in many natural languages. Such advances can be attributed to the development of new neural architectures, extreme scaling of models, methods for pre-training the models on large cross-lingual corpora and MTL deployment. Multitasking and multilinguality are invaluable properties of modern language models used for real-world problems. Pragmatically, running one instance of a universal model is cheaper and easier to manage than running a different model for each combination of tasks and languages.

This work aims to research possibilities for improving the performance of existing multilingual neural classifiers with MTL methods. Some resulting models achieve state-of-the-art (SOTA) results on the Czech fact-checking task CTKFactsNLI [2]. The concepts and methods are introduced in Chapter 2. They are applied to create an MTL setup (see Chapter 3). In Chapter 4, the setup is jointly trained and evaluated on a selection of tasks from the XGLUE [3] multilingual benchmark, then the performance of the resulting models is compared on the target task (CTKFactsNLI). The results are discussed in Chapter 5.

The source code of the setup[1] and the experimental runs[2] are available online. The trained weights will be gradually published[3].

[1]https://github.com/semindan/mtl_thesis
[2]https://wandb.ai/semindan/thesis
[3]https://huggingface.co/semindan

Chapter 2

Theory

2.1 Text classification

Definition 2.1. Text classification. Given a text unit x and a set of labels R, assign a label $r \in R$ to x.

Text classification (TC) or text categorization tasks are popular benchmarks in natural language understanding (NLU). They have numerous real-world applications, such as spam classification, news categorization and sentiment analysis. A review of models and datasets for TC is presented in [4]. For automatic TC, rule-based approaches, which require a deep understanding of the task, were replaced by ML methods. Classical ML algorithms for classification, such as naive Bayes, support vector machines, hidden Markov models, random trees and forests, traditionally have relied on hand-crafted features extracted from textual units. To obtain good performance, data required feature engineering and analysis. The design of these features mostly depends on task-specific knowledge, which often limits the generalization capabilities to new tasks. Embedding methods, which encode text information as low-dimensional feature vectors, address this problem. Modern neural models almost exclusively use learned features. While generally achieving better performance on tasks including TC, they are criticized for lacking robustness and interpretability. Still, neural networks, particularly pre-trained models based on the transformer architecture [5], are the current SOTA in TC.

In benchmarks, it is common to test the model on tasks related to natural language inference (NLI), a pair-input classification task aiming to predict whether a hypothesis can be inferred from a premise. There are typically three possible labels: entailment, neutral and contradiction. A general version of NLI involves deciding if one sentence is a paraphrase of the other in a sentence pair. Fact verification is one of the formulations of NLI, which is particularly interesting. The goal of the fact-checking system, given the claim and evidence, is to determine whether this evidence supports or refutes the claim or if there is not enough information to decide.

2.2 Transformer

The transformer architecture [5] has changed the landscape of NLP. It improves upon the previous SOTA architecture, recurrent neural networks (RNN). Primarily, it alleviates RNN's long dependency issues by processing the input sequence as a whole. The critical component in transformer is the attention mechanism. It has become a standard unit in NLP models since its application to long short-term memory (LSTM) RNNs for machine translation [6].

Figure 2.1 shows the visualization of the architecture. It consists of the encoder and decoder, stacks of multiple identical layers. Initially, the model replaces each token in the input sequence with an embedding vector. The sequence of embeddings is summed with positional encodings and passed into the encoder. The encoder aims to produce a "useful" representation, sometimes denoted as "context." Each encoder layer is formed by two sub-layers: a multi-head self-attention layer and a position-wise fully connected feed-forward network. An encoder layer yields a sequence of hidden state vectors, which are then processed by the next encoder layer. The second block is the decoder that generates the final output sequence. A decoder layer is similar to an encoder layer, yet its multi-head self-attention is masked to ensure that already decoded positions cannot attend to the following positions. A decoder layer adds a third sub-layer – multi-head attention over the encoder's final output. The first input to the decoder is the `[SOS]` (start-of-sequence) token. That way, the decoder always receives the right-shifted sequence. The linear layer and the softmax layer are used for converting the resulting hidden states of the decoder's stack into probability vectors, each having the length of the model's "vocabulary." At this point, the output tokens can be constructed (e.g., using greedy decoding) and passed back to the decoder to generate the next token.

A residual connection and layer normalization wraps the output of each sub-layer in the model. Dropout [7] is applied to the output before performing these operations. Additionally, dropout is applied to the input of both encoder and decoder stacks, i.e., to the sums of the embeddings and the positional encodings.

This overview of transformer would be incomplete without discussing its inner workings.

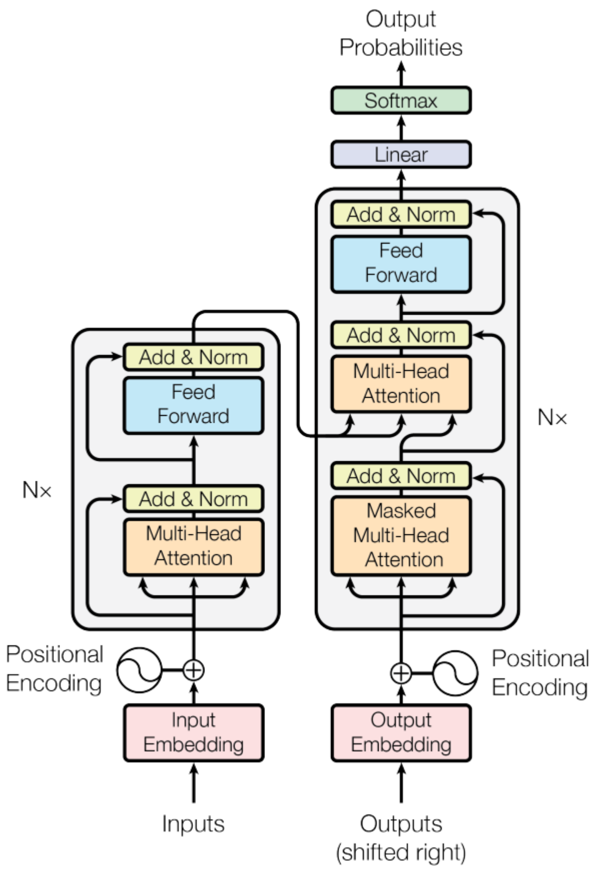


Figure 2.1: Visualization of transformer, the image is from [5]

Self-attention

The variant of attention used in transformer is called self-attention. It is usually presented as a function that takes a query and a set of key-value pairs as input and outputs a weighted sum of the values. In practice, inputs are embedding vectors packed into matrices Q, K and V. All three matrices come from an input sequence. Self-attention is applied to each element separately and identically. It captures the relation of every element to all other elements in a given set.

Scaled Dot-Product Attention

The output of the self-attention layer is computed with scaled dot-product attention. The dot products of larger vectors tend to have larger magnitudes. Due to this, the softmax function produces values close to 1 or 0, resulting in small gradients. To counteract this, the authors introduce the scaling factor $\frac{1}{\sqrt{d_k}}$. This operation is formalized as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-head attention

Transformer utilizes multiple parallel attention layers called heads. Each head learns its own weight matrices to project Q, K and V to low-dimensional linear subspaces before performing the attention function. Output matrices of heads are concatenated and, finally, linearly projected by an additional weight matrix to the subspace of the model dimension. Mathematically multi-head attention is expressed like this:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

While highlighting relevant information, single-head attention may eliminate other valuable aspects of it. Deployment of multiple heads mitigates this disadvantage. They increase the model's capacity and allow it to view the information from different perspectives.

Positional encodings

Multi-head self-attention is a set operation – it ignores the order of elements in an input sequence. In most cases, it is crucial to preserve the order of words in a text sequence. RNNs do this by design as they utilize hidden states derived from the previous words to process the subsequent ones. Transformer, however, does not involve any recurrence or convolution. Other techniques must be deployed to capture the structure of the input. Authors solve this by injecting positional information into the sequence of embeddings right before passing it to a transformer's block. While the positional encodings used in the original transformer are fixed, learned positional encodings, or positional embeddings, produce almost identical results. The fixed version has an advantage: it is not limited by the sequence lengths encountered during training. The proposed encodings are computed with sine and cosine functions of varying frequencies:

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$
$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

where pos is the position and i is the dimension, d_{model} is the size of each embedding.

These positional encodings match the shape of the input embedding sequence, and the two are summed. Recent models, starting with BERT [8], use learned encodings over static ones.

Position-wise Feed-Forward Network

This layer has two linear transformations and a rectified linear unit (ReLU) activation between them. They are applied to each position of the multi-head attention layer outputs separately and identically.

$$FFN(x) = max(0, W_1x + b_1)W_2 + b_2$$

2.3 Multilinguality

Ruder[1] argues that many reasons exist for working on languages beyond English. In short, more than 7000 languages are spoken worldwide, yet NLP research is English-centric. Only 1% of known languages [9] enjoy a large amount of labeled and unlabelled data (groups 4, 5 in Figure 2.2), while other languages are mostly neglected. From the ML perspective, contemporary multilingual neural models are far from being language agnostic – many of their inductive biases are specific to English and other high-resource languages.

It takes work to create multilingual datasets for each target language. Typically, human annotators develop validation and test datasets for each target language, and the training data is usually annotated only in English. The training corpus can be translated into other languages as means of data augmentation. Modern multilingual models achieve competitive results on multilingual datasets by only learning a task in one source language. This emergent ability (and the learning scheme itself) is referred to as zero-shot cross-lingual transfer. In NLP, multilinguality is a complex field, the theoretical details of which are out of the scope of this work.

Multilinguality in transformers

BERT [8] encoder-only architecture introduced the ”pre-train, then fine-tune” framework for transformers. The model learns general language representations during the pre-training by training on two self-supervised tasks. The first is masked language modeling (MLM). With a chosen probability, tokens in an input sequence are replaced with a `[MASK]` token. The model with a language modeling head predicts what tokens were replaced. The second is next sentence prediction (NSP): the model receives a two-sentence segment and, using a binary classification head, decides if the second sentence is the actual sentence that follows the first sentence. The two described tasks are performed on the same input, and the respective losses are summed. The authors of BERT released a multilingual version of the model, trained on the Wikipedia corpora from 104 languages. This multilingual version, denoted mBERT, can perform cross-lingual generalization, even successfully transferring between languages with no lexical overlap [10]. The next considerable improvement of the pre-training strategy was RoBERTa [11]. The most significant changes involve the removal of the NSP objective. In MLM, instead

[1] `https://ruder.io/nlp-beyond-english/`

of copying the same segments and masking different tokens, they are randomly masked right when they are fed to the model. XLM-R [12] leverages this strategy and trains on 100 language corpora from the cleaned CommonCrawl[2]. T5 [13] and its multilingual counterpart mT5 [14], which are text-to-text models in contrast to BERT-based models, extend the BERT-style pre-training process. They replace spans of tokens with sentinel tokens and the model's objective is to produce only the replaced tokens (see Figure 2.3).

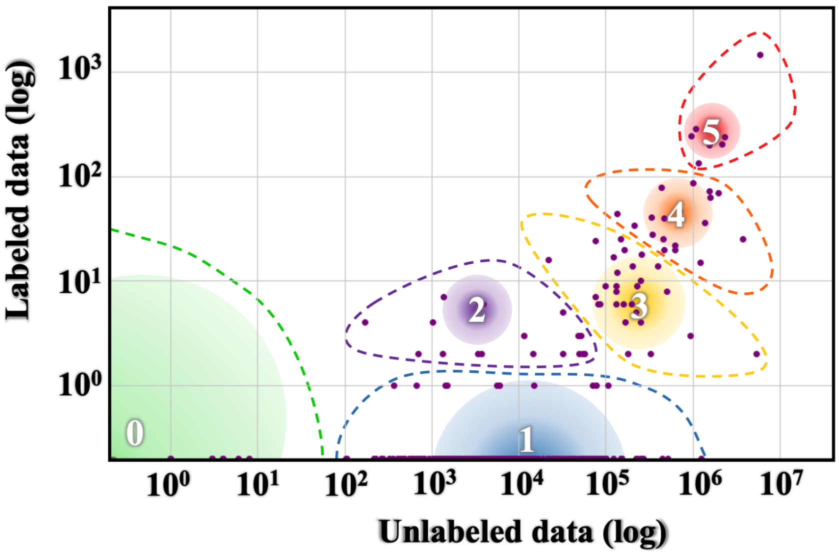


Figure 2.2: Language data distribution, the image is from [9]

The input word sequence is tokenized before being passed into the model, and when the model is expected to generate tokens as output, the output is detokenized. The multilingual model should be able to handle many diverse languages, therefore, handle different scripts and separation modes (e.g., Japanese and Chinese do not separate words with spaces). On the other hand, a reasonable vocabulary size is desired, i.e., there should be a way to choose what tokens to include. Modern multilingual transformers use SentencePiece [15] that solves these problems. In short, it is a subword tokenizer that treats the input text as a sequence of characters (space is not treated in any specific way). SentencePiece implements the unigram segmentation [16] and byte pair encoding [17] to build a vocabulary (of a pre-defined size) based on subwords. In short, the most frequent words end up in the vocabulary. The other words are constructed with subwords.

[2]`https://commoncrawl.org/`

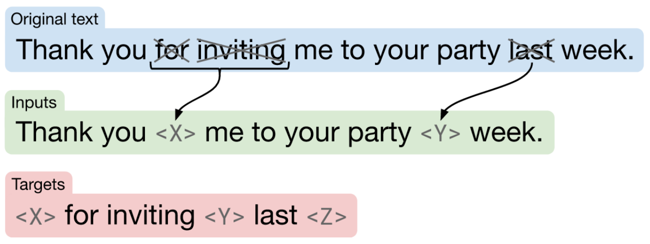

Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 2.3: T5's pre-training objective, the image is from the original paper [13]

2.4 Multitask learning

The Motivation for developing a neural network able to perform multiple tasks is pragmatic – instead of running and switching between n instances for n tasks, a single model handles n tasks. The memory is allocated only for one network, and there is no additional overhead during inference.

In their seminal paper [1], Caruana lays the foundation for MTL and, in various ways, demonstrates that artificial neural networks and even specific traditional ML algorithms benefit from learning multiple related tasks simultaneously. Caruana defines MTL as "an inductive transfer mechanism whose principle goal is to improve generalization performance." MTL learns related tasks in parallel while using a shared representation. By doing this, MTL leverages the domain-specific information contained in their training signals. Although the concept is straightforward, the MTL framework and its goals differ greatly depending on the application. Terms like task relatedness and domain are ambiguous. The paper specifies the latter as a complex task involving many subtasks (e.g., road-following domain, medical decision-making domain). A domain may be equally interpreted as a topical difference in one modality, for example, NLP for medical data or legal documents. Moreover, in NLP, different languages are sometimes treated as domains [18], even for the same tasks (e.g., multilingual text classification). The following paragraphs use the term domain to distinguish tasks based on their formulation, such as text generation or question-answering domains.

Parameter sharing

In MTL, parameter sharing is necessary for creating a general representation suitable for multiple tasks. In backpropagation models, the training process is centered around minimizing a loss function by updating model parameters. When the parameters are shared, training signals from different tasks contribute to the update of the parameters. Caruana observes that ”MTL tasks prefer hidden layer representations that other tasks prefer” [1]. Neural networks for MTL are typically designed using hard parameter sharing or soft parameter sharing [19]. The first denotes a model that shares most of its hidden layers between all tasks and separates task-specific output layers. Canonically, MTL encoder-only models deploy this sharing strategy [20]. The encoder is shared, and task-specific layers (e.g., classification heads) are attached to it. Hard parameter sharing is widely used – a majority of modern large networks, which are often encoder-decoder models [13] or decoder-only models [21, 22], share all parameters and do not have any task-specific layers. On the other hand, in soft parameter sharing, each task has its own separate model. The sharing comes from regularizing the distance between the lower layer parameters, which encourages the parameters to be similar.

Datasets for MTL

There are generally two MTL settings in NLP [23]. First, one dataset is associated with multiple outputs (e.g., named entity recognition and relation extraction on the same input). In computer vision, MTL is usually performed for tasks with the same inputs (e.g., detecting cars and pedestrians in the same image). This setting allows the model to compute and manipulate losses for these tasks in a single forward pass. However, creating such multifaceted datasets for NLP tasks is highly labor-intensive. The second setting, which is a default situation for NLP, involves multiple datasets with their respective labels, where samples are drawn from different tasks and jointly learned in parallel. Currently, the most popular benchmarks in this setting are XGLUE [3], XTREME [24] and MMLU [25].

Task relatedness

Task relatedness is yet to be formalized, considering its counterintuitive nature: in practice, seemingly related tasks may have underlying competing dynamics [26]. On the other hand, unrelated tasks may still be beneficial for the training process as a source of ”noise” [1]. Therefore, manual task selection is often a fruitless approach. Nevertheless, if the learned tasks come from the same domain, say TC, there is a particular intuitive assumption that these tasks are more related to each other than any task from TC to any task from text generation, question-answering or any other domain. The straightforward method to measure task relatedness is to train models on the given tasks and then measure the performance gain with respect to single-task training. This method is computationally expensive, even for a small set of tasks. However, such massive empirical studies bring an opportunity to evaluate methods for approximating task relatedness. In computer vision, Task2Vec [27] is a method to encode a task into a vector representation to predict task similarities. In NLP, this paper [28], based on Task2Vec, introduces two methods for finding the best intermediate task for transferring knowledge to the target task. The first, TextEmb, collects the average of the final layer token-level representations h_x for each sample x of a dataset D from a pre-trained model's encoder without fine-tuning. The final task embedding is an average of these pooled vectors over the entire dataset:

$$\sum_{x \in D} \frac{h_x}{|D|}$$

TextEmb is supposed to capture the linguistic properties of the input text and does not depend on the training labels. The second method, TaskEmb, describes the task by highlighting the parameters most beneficial to the task. This is achieved by computing the empirical Fisher information matrix (FIM) with respect to parameter θ derived from the fine-tuned model's weights:

$$F_\theta = \frac{1}{n} \sum_{i=1}^{n} [\nabla_\theta \log P_\theta(y^i|x^i) \nabla_\theta \log P_\theta(y^i|x^i)^T]$$

Only the diagonal entries are considered in order to reduce computational complexity. FIM is computed with respect to different components of the model and additionally with respect to the outputs of the components. The latter is averaged over the input tokens and the entire dataset. The resulting embeddings of the source tasks are ranked by cosine similarity with the target task's embeddings by component. The final score is computed according to the reciprocal rank fusion (RRF) algorithm [29]:

$$RRF(s) = \sum_{i=1}^{c} \frac{1}{60 + r_i}$$

where s is a source task, c is a component index, r_i is a rank score assigned to s by the component i of the model. TaskEmb and TextEmb can be further combined using RRF to produce the aggregated score.

Although TextEmb and TaskEmb originally are not intended for MTL, the current MTL research [30] deploys conditional adapters [31] to derive task embeddings and uses TextEmb and TaskEmb as baselines.

Another notable category of approaches is to learn or manage task relations dynamically during training. PCGrad [32] is a decent representation of this group of methods. The core idea behind the algorithm is to mitigate gradient interference by performing ”gradient surgery.” The conflicting gradients are each projected onto the normal plane of the other. This method prevents the interfering components of the gradient from being applied to the network.

MTL at different stages of training

We may want to achieve better performance on one target task by training it together with auxiliary tasks. Even though it is a reasonable goal for MTL, some researchers [33] claim this objective is more suitable for the adjacent framework called transfer learning. They state that MTL, in contrast to transfer learning, aims to improve performance on all of the learned tasks. Transfer learning, in short, happens when the model is pre-trained on a task or a set of tasks and only then is fine-tuned on a target task, therefore transferring the knowledge gained during pre-training. The two frameworks are often combined at different stages. For pre-training, BERT [8] jointly learns masked language modeling and next-sentence prediction tasks. Authors of the T5 paper [13] claim that in NLP, unsupervised pre-training and then single-task fine-tuning consistently outperforms MTL setups. They experiment with setups to ”close the gap.” They demonstrate that a classic unsupervised pre-training is roughly equivalent to a multitask ”mix” of unsupervised and supervised tasks during pre-training. In both pre-training scenarios, subsequent single-task fine-tuning produces almost identical results. Currently, the most popular approach is to add a selection of supervised tasks as an intermediate step after unsupervised pre-training. Initially introduced in [20], this idea developed into a new approach known as multitask pre-finetuning [34]. Given a model pre-trained in an unsupervised fashion, the current SOTA MTL approach is to gather a massive selection of tasks to learn jointly before individual fine-tuning. Apparently, past a certain number of tasks, the massively pre-finetuned model provides a better basis for task-specific fine-tuning.

Catastrophic forgetting

Given a pre-trained MTL model, it is common to fine-tune it on the target task. A problem arises: updating the model’s parameters on a new task degrades the model’s performance on the ”old” tasks. This problem is known as catastrophic forgetting or catastrophic interference [35]. Freezing the shared layers of the model and updating exclusively task-specific layers is not competitive to the full fine-tuning. To alleviate the issue, the recent approaches freeze the model, inject new parameters and update only them (e.g., adapter layers [36]). One of the prominent methods is LoRA [37]. It constrains an update of a pre-trained matrix W_0 by representing the former with a low-rank matrix decomposition. During training, W_0 is frozen, and two matrices, A and B, are updated. The forward pass is then formalized in the following way:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

The authors of LoRA report that the method can approach the performance of full fine-tuning. The advantage is that LoRA adds a small number of parameters to the model (typically 1-2% of the original model size). These added weights are lightweight and can be quickly replaced with other LoRA weights on demand during inference. As the number of trained parameters is low, the training process is significantly less demanding computationally, allowing to train large models on consumer hardware. In contrast to adapter layers, there is no additional inference latency as the new weights can be precomputed.

When fine-tuning a pre-trained model, the pre-trained representations are likewise subject to catastrophic forgetting, e.g., a model that forgets how to perform its MLM task on certain languages practically forgets these languages. The R3F and R4F methods [38] prevent the degradation of generalizable representations of pre-trained models during fine-tuning by reweighting the loss. The new loss with applied R3F/R4F is:

$$\begin{aligned}
&L_{R3}(f,g,\theta) = L(\theta) + \lambda KL_S(g \cdot f(x) \,||\, g \cdot f(x+z)) && \textbf{R3F} \\
&\text{subject to } z \sim N(0,\sigma^2 I) \text{ or } z \sim U(-\sigma,\sigma) && \\
&\text{subject to } Lip\{g\} \leq 1 && \text{optional } \textbf{R4F}
\end{aligned}$$

where θ denotes a parameter, λ is a weighting factor, KL_S is symmetric Kullback-Leibler divergence (KL), $\cdot$ is function composition, f is function performed by the model's shared layers, g is a function of a classification head, z is a sample from a parametric distribution – N normal or U uniform. R4F "adds a constraint on the smoothness of g by making it at most 1-Lipschitz." In practice, spectral normalization is applied to the last layer of a classification head.

Chapter 3

Setup

3.1 Models

Transformer-based models are the current SOTA in a majority of NLP domains, including TC. It is well-known that SOTA performance is often achieved by larger models. For multilingual transformers, assuming the model is open-source and its pre-trained weights are publicly available, XLM-R, mT5 and their derivatives perform competitively across all scales. Additionally, I use mBERT for reference in initial experiments.

Regarding model sizes, I use *base* versions: mBERT has 110 million parameters, XLM-R$_{\text{base}}$ has 270 million and mT5$_{\text{base}}$ has 582 million. Although it may be arguably incorrect to compare these models, in practice, it is a fair comparison. The discrepancy between mBERT and XLM-R is given by the vocabulary size (119 thousand token embeddings vs. 250 thousand). The computational cost is similar for the same input size. mT5 uses a decoder, which has the same structure (except for cross-attention) as an encoder and has a linear output layer. Its encoder has roughly the same number of parameters as XLM-R. mT5 requires twice as much memory as XLM-R. However, for TC tasks, the decoder acts as a classification head over the vocabulary and only generates a few additional tokens – usually a label and the end-of-sequence token. Therefore, the computational cost is roughly similar [14].

I am particularly interested in mT5 as it represents a unified text-to-text framework, which is highly popular in larger models, such as LLaMA [21].

mT5 can be viewed as a predecessor to the latest frameworks as initially it has no task-specific layers, in contrast to MTL XLM-R, and distinguishes tasks based on simple instructions: a task prefix and task-specific feature names in the input sequence. Despite having more parameters than equivalent versions of encoder-only models, mT5 was pre-trained using only $\frac{1}{6}$ as much data as XLM-R [14].

3.2 Datasets

I have chosen TC tasks from XGLUE [3] as a benchmark for joint MTL experiments. XGLUE is a set of 11 cross-lingual tasks, 6 of which are classification tasks. The classification tasks in this benchmark seem closely related. The held-out dataset for testing the benefits of MTL pre-finetuning is CTK-FactsNLI [2]. All used datasets are curated, meaning they are ready to use.

Natural Language Inference (XNLI)

XNLI [39] is a cross-lingual natural language inference corpus. English validation and test sets are translated into 14 languages: French, Spanish, German, Greek, Bulgarian, Russian, Turkish, Arabic, Vietnamese, Thai, Chinese, Hindi, Swahili and Urdu. Moreover, the authors provide machine-translated training datasets. XNLI is the largest dataset in my selection, with more than 392 thousand training examples.

Paraphrase Identification (PAWS-X)

PAWS-X [40] is a cross-lingual paraphrase identification task. XGLUE version incorporates English, French, German and Spanish. The original dataset additionally has Chinese, Japanese, and Korean. The dataset contains 49 thousand of training examples.

News Categorization (NC)

NC is a single-input task aiming to predict the category of a news article, focusing on five different languages, including English, Spanish, French, German, and Russian. Each instance in the dataset consists of a news title, body, and category (overall, there are ten categories). The data is collected from a commercial news website, and the metric is multi-class accuracy.

Query-Ad Matching (QADSM)

QADSM is a task of predicting whether an advertisement is relevant to an input query. It covers three languages, including English, French and German. The dataset is based on a commercial search engine.

Web Page Ranking (WPR)

In WPR, the task is to decide how much a given web page is relevant to an input query on a scale ranging from ”Bad” to ”Perfect” (overall, there are five categories). It covers seven languages: English, German, French, Spanish, Italian, Portuguese and Chinese. This dataset is likewise based on a commercial search engine.

Question-Answer Matching (QAM)

QAM is a set of question-answer pairs, and the task is to decide if the passage is the answer to the question or not. It covers three languages: English, French and German. The dataset is constructed on data from a commercial search engine.

CTKFactsNLI

CTKFactsNLI is a fact-checking task in the Czech language based on the Czech News Agency[1] news reports. It is a low-resource task comprising 3626 labeled claim-evidence pairs for training, 482 for validation and 558 for testing.

Pre-processing

I reformulate the tasks to text-to-text format for mT5. The labels are additionally stored in the text format and the features are transformed into a single string. For MTL, I prepend the input string with a task prefix. An example of the pre-processed entry is in Figure 3.1. XLM-R processes a sentence or a sentence pair separated by `[SEQ]` token. Tokenizer automatically inserts this token when the features are passed separately. In order to conform to this format, I concatenate the advertisement title and body in QADSM, web page title and snippet in WPR. NC is a single-input task, so I concatenate the news title and body. Other XGLUE datasets do not require any pre-processing.

I pre-process CTKFactsNLI in two modes: a standalone task and a reformulation of XNLI (for mT5). In both cases, I reorder the mapping of label names to label indices: `SUPPORTS` to `0`, `NOT ENOUGH INFO` to `1` and `REFUTES` to `2`. If the task is reformulated, e.g., for a zero-shot evaluation, the features are renamed: `evidence` becomes `premise`, `claim` becomes `hypothesis`. The same pre-processed example in two modes for mT5 is showcased in Figure 3.2.

3.3 Metrics

Choosing a metric for a task is crucial for a correct performance assessment. All described datasets, except for WPR and CTKFactsNLI, use accuracy as a metric. It is formulated as follows:

$$Accuracy = \frac{number\ of\ correct\ predictions}{number\ of\ examples}$$

[1] `https://www.ctk.eu`

I should note that accuracy is usually not a metric of choice for unbalanced data. Although, for instance, NC uses accuracy and has a highly unbalanced label distribution in the validation sets, this is compensated by a similar imbalance in the training set (`foodanddrink` and `sports` categories outweigh the rest, see Figure 3.3). This way, the matching unbalanced sets do not interfere with the ”true” label distribution as it is assumed to be unbalanced.

Performance on WPR is calculated with normalized discounted cumulative gain (nDCG). For this task, I additionally insert so-called ”guids” into the dataset – simply, entries have the same guid if and only if they belong to the same query. For practical purposes, I assume the predictions and ground truth vectors have the same size, and the latter are sorted by relevancy. Mathematically, nDCG over the whole input of size n is expressed as:

$$nDCG_n = \frac{DCG_n}{IDCG_n}$$

where $\mathrm{DCG_n}$ is

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)}$$

rel_i is a graded relevance of the result at position i, and $\mathrm{IDCG_n}$ is the ideal discounted cumulative gain, computed the same way as $\mathrm{DCG_n}$ from the ground truth list.

CTKFactsNLI is a small and unbalanced dataset (see Figure 3.4), which uses F1 macro score. Each label receives its F1 score, and the scores are averaged. The formula for one label is:

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

where TP, FP and FN are standard acronyms for true positive, false positive and false negative.

```
{'label': 0,
 'input': 'nc: news_title: How to whisk egg whites
   news_body: Great British Chefs demonstrates
   how to whisk egg whites',
 'target': 'foodanddrink'}
```

Figure 3.1: mT5 data format

```
{'label': 0,
'input': 'ctkfacts_nli: evidence: Střelec z denverského kina
```

```
  předstoupil před soud claim: Denverský střelec
  byl u soudu.',
 'target': 'SUPPORTS'}

{'label': 0,
 'input': 'xnli: premise: Střelec z denverského kina předstoupil
  před soud hypothesis: Denverský střelec byl u soudu.',
 'target': 'entailment'}
```

Figure 3.2: CTKFactsNLI pre-processed example in the text format. The second entry showcases its reformulation as XNLI

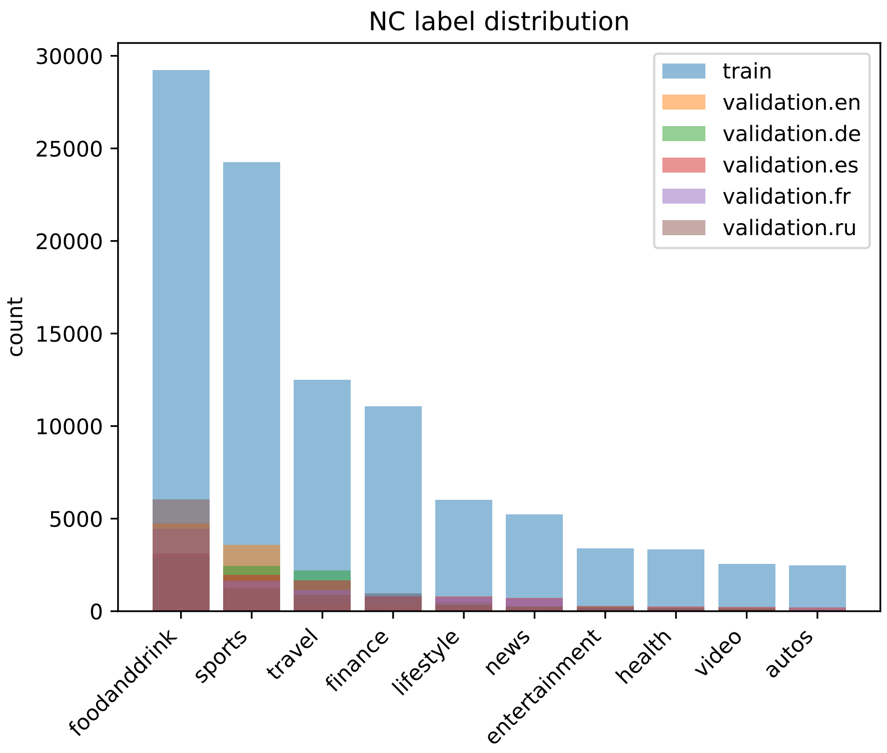


Figure 3.3: The highly unbalanced label distribution of NC

3.4 Methods

Models are trained with the recommended optimizers: XLM-R, mBERT and LoRA are trained with Adam [41], mT5 is trained with Adafactor [42]. Initially, I use unsupervisedly pre-trained multilingual transformers and evaluate them on multilingual tasks. As XNLI and PAWS-X provide translated training sets, it may be beneficial to incorporate the training data in different languages. Additionally to the cross-lingual zero-shot transfer setting (a model is trained in one language and evaluated in all target languages),

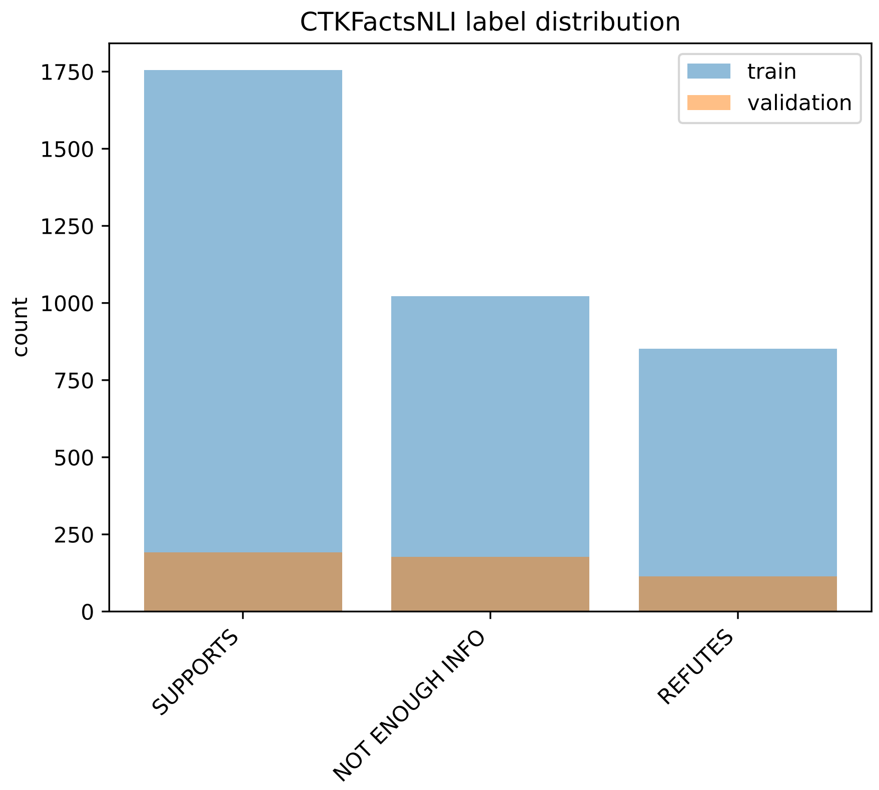


Figure 3.4: The label distribution of CTKFactsNLI

I evaluate other training schemes: translate-train (a model is trained and evaluated only in one language) for XLM-R and mBERT and, for the most resourceful task XNLI, a modified version of translate-train-all (a model is trained on a concatenation of all languages). Instead of concatenating data from all languages, I reconstruct XNLI using disjoint partitions in different languages and denote it XNLI$_{mix}$.
I use hard parameter sharing for MTL experiments, where XLM-R has task-specific classification heads and mT5 has a shared decoder. MTL has several methods for improving performance and mitigating the negative knowledge transfer between tasks. Methods that involve gradient manipulation, such as PCGrad, are expensive memory-wise. The tangible methods in most setups are loss reweighting and data sampling. Following [34], I apply the R3F method and loss scaling. After reweighting with R3F, the loss is scaled:

$$L_i^{scaled}(x_i, y_i; \theta) = \frac{L_i(x_i, y_i; \theta)}{\log n(i)}$$

where L_i is a loss for data point i and n is a function that returns the number of predictions (e.g., 2 for binary classification, vocabulary size for generation). The authors have found this scaling method to be more effective than other forms of loss scaling they have tried (the "other" techniques are not specified in the paper). I do not apply R4F due to the inconsistencies stated in 3.5.

I additionally explore mixing in mT5's MLM task with a fixed probability, as it was reported to be beneficial for performance [14]. For the MLM task, n in the scaling factor returns the vocabulary size. The languages used in MLM are the target languages of the selected tasks.
Regarding data sampling, I follow [34] and preserve the original data size distribution. For that, I use proportional sampling, meaning the probability of sampling from a task i is proportional to the size of its dataset D_i:

$$p_i \propto |D_i|$$

Besides, I experiment with using task-heterogeneous effective batches, meaning the effective batch is formed by the batches drawn from different tasks. The motivation for this is to include as many tasks as possible in each effective batch. Each optimizer step is then performed on the aggregated gradients computed from different tasks.
Finally, I showcase the performance of models on the target task CTKFact-sNLI. I compare MTL models with models pre-trained on the best intermediate task for transfer, following [28]. I additionally compare LoRA performance on the target task with the fine-tuned mT5.
In all experiments, I pick the best checkpoint based on the average of the validation set score over all tasks and languages.

3.5 Implementation details

All my experiments are conducted on RCI cluster[2]. I train the models on 4 Tesla A100 40GB GPUs on a single node. The code for the experiments is written in Python and utilizes popular libraries for working with neural networks.

Hugging Face transformers and datasets

Hugging Face [43] library `transformers` offers ready-to-use transformer models and utilities for training. Hugging Face has a hub where users share pre-trained and fine-tuned models. On top of that, transformers have specialized model wrappers for prefix tuning, prompt tuning and training the adapter modules. The other convenient library by Hugging Face is `datasets`. The datasets can be shared on the same hub. The library provides methods for mapping functions over all splits of the dataset. It is compatible with PyTorch [44] `Dataset` class and uses fast Arrow tables for storing the datasets. Hugging Face transformers library offers a `Trainer` class for managing the training process. However, it is not as flexible as the framework I have decided to use for this purpose.

[2]`https://login.rci.cvut.cz`

PyTorch Lightning

Pytorch Lightning is a framework for training neural networks. It includes callbacks for logging results, checkpointing models and early stopping of the training process, distributed training strategies, classes for data processing and loading and model training. Every stage of the process (training, validation, testing, prediction) is under the user's control, which makes Lightning's `Trainer` more flexible than Hugging Face's `Trainer`. I wrap the models into a `LightningModule` class with model-specific code and wrap the model module with an outer `LightningModule`, where I log the metrics. The data is managed in `LightningDataModule`.

Distributed training

Multiple GPUs accelerate the training process, yet they need to communicate with each other to achieve this. Due to the global interpreter lock in Python, multithreading is not a viable option (except for input-output operations). Therefore, multiprocessing is utilized. Lightning relies on multiprocessing in the distributed data-parallel (DDP) strategy. Each GPU is assigned to a separate process. Every process runs a model instance that is identical across all processes. During training, the forward pass is local and, during the backward pass, the computed gradients are accumulated in parameter buckets. The prepared buckets are then averaged across processes. This way, the `grad` field of the parameters is the same across all DDP processes. The optimizer step is then applied locally. The tradeoff of DDP is that it requires extra memory to store the parameter buckets.

Model-specific code

For XLM-R, I combine two parts from the `transformers` library: the first is the XLM-R encoder itself and the second is a generic classification head. I pass a list of tuples consisting of task names and their number of labels. I initialize the required classification heads from this list and collect them into a dictionary (keys are the task names) using `nn.ModuleList` from PyTorch. mT5 is from `transformers` and used as is.

R3F and R4F

The R3F and R4F methods are adapted from the Meta's `fairseq` library[3]. At the time of writing, their implementation of R3F is numerically unstable on mT5 – the training loss becomes `nan` after a few steps. To compute Kullback-Leibler divergence, the predictions are passed to log softmax beforehand, and the targets are passed to softmax. Then, the empirical formula is formalized as:

$$KL(y_{pred}, y_{true}) = y_{true} \cdot (\log(y_{true}) - y_{pred})$$

[3] `https://github.com/facebookresearch/fairseq/tree/main/examples/rxf`

where · denotes pointwise multiplication.
The numerical instability of the procedure seems to be caused by the difference between computing the log softmax from the original logits using the `torch.nn.LogSoftmax` function and computing the log of the output of `torch.nn.Softmax` inside the Kullback-Leibler divergence function `torch.nn.functional.kl_div`. The solution is to pass the targets to `torch.nn.LogSoftmax` instead of `torch.nn.Softmax` and set the `log_target` parameter of KL to `true`. The flag was introduced to combat the instability[4]. Assuming both of the inputs are outputs of `torch.nn.LogSoftmax`, the formula translates to:

$$KL(y_{pred}, y_{true}) = \exp(y_{true}) \cdot (y_{true} - y_{pred})$$

R4F is implemented by wrapping the last `Linear` layer of the classification head with `torch.nn.utils.parametrizations.spectral_norm`. However, I have observed that with this method, the model produces decent results in evaluation during training and significantly worse results on the same data in the validation phase (in Lightning, the model is loaded from the best checkpoint after training). The use of DDP may cause the problem, though DDP promises to keep the instances identical after every optimizer step. Right now, the cause is still unclear to me.

LoRA

Hugging Face now supports parameter-efficient fine-tuning (PEFT) with their library `peft`. LoRA is one of the offered methods. The `peft` initialization function receives the model and returns the model with injected LoRA parameters. The original model's weights are frozen as a result of that operation. The training process proceeds as usual. I have encountered a problem: when loading the model's weights from a checkpoint, Lightning first initializes the model and only then loads the state dictionary. However, if I wrap the model with LoRA during initialization, the keys are mismatched as LoRA pushes the keys of a base model deeper into the state dictionary. I have partially solved this by wrapping the model with LoRA during optimizer initialization, i.e., after the weights are already loaded. I additionally save the LoRA weights separately.

Handling data

I adapt some tricks from the developers of `jiant` [45]. I use an MTL data loader that accepts task data loaders as an argument, picks a task to sample and yields a task batch from the respective data loader. I add arguments for a list of task probabilities and a seed for more control over the task sampling. XLM-R chooses the head using a task name, thus the task name must be somehow stored in a batch. The second trick is to iterate through the data loader and add the desired information to each batch. However, PyTorch

[4]`https://github.com/pytorch/pytorch/issues/32520`

tensors can only contain numerical values, it does not allow the strings on the CUDA device. Thus I use another trick that wraps the string with a `str` subclass that only contains a `to` method, which is just the identity function. When `to` is called on the task name, the string is not moved to the target device.
In order to create the multilingual training set XNLI$_{mix}$ leveraging the translated training sets of XNLI, I shuffle the translated sets in different languages (including the original English training set) with the same seed and then add them to a new list, gradually taking a block of equal size from each translated set, resulting in non-overlapping partitions. To merge the resulting list of data blocks I pass it to `datasets.interleave_datasets`. The result is XNLI$_{mix}$, which consists of disjoint partitions from the XNLI's training set. Pseudocode of the procedure is in Figure 3.5.
I adapt the code from the Hugging Face repository[5] to reproduce the correct MLM preparation for mT5's training process. The dataset is mC4[6], as in the original paper [14]. The partitions of high-resource languages have massive sizes, reaching multiple terabytes. Therefore I use the `streaming` option when loading the data with `datasets.load_dataset`. In order to acquire multilingual batches, I mix the monolingual sets with `datasets.interleave_datasets`.
Aside from the standard feature pre-processing and tokenization, the data should be prepared for DDP. Each model instance should have a unique batch in the forward pass. Otherwise, DDP has no computational benefits. I use PyTorch's sampler called `DistributedSampler`. In DDP, every process has a unique rank. The distributed sampler receives this rank and the number of all training processes as an argument and evenly distributes dataset indices between them. I pass the sampler to a task-specific data loader, wrap the latter with a data loader providing the task name and then pass the list of data loaders to the described MTL data loader.

Task embeddings

To reproduce the results of [28], I adapt the code from the official repository[7]. Tracking of gradients of layer outputs for TaskEmb and TextEmb is implemented by retaining the gradients of the target tensors using the `retain_grad` method. For TaskEmb, only diagonal entries of FIM are considered. Thus it suffices to raise the collected gradients to the power of 2 before normalization. `igraph` library[8] is utilized for visualizations.

[5]`https://github.com/huggingface/transformers/blob/main/examples/flax/language-modeling/run_t5_mlm_flax.py`

[6]`https://huggingface.co/datasets/mc4`

[7]`https://github.com/tuvuumass/task-transferability`

[8]`https://python.igraph.org/en/stable/`

Logging

I use Weights & Biases[9] for logging. The library is flexible and intuitive and provides many ready-to-use solutions for tracking gradients (helps with detecting divergence during training). Moreover, it allows the user to set up custom notifications, which has helped me track longer experiments more conveniently.

Notes on tools for MTL

Validation and test phases over multiple datasets are supported in most frameworks, yet there is little support for simple mixing and serving datasets in the training phase. One of the promising libraries for that is `seqio`, a prototype of which was used for T5 training. It is specifically designed to handle multiple tasks and mix them. Unfortunately, it is not yet mature for different dataset formats and it utilizes `TensorFlow`, which is a significant downside in the current era. Maintenance of an MTL library `jiant`, explicitly built for training models in an MTL setting, combining datasets and managing multiple classification heads of models, was discontinued in 2021.

```
language_sets = list()
for language in XNLI_LANGS:
    language_set = load_xnli_train(language).shuffle(seed)
    language_sets.append(language_set)

train_size = length(language_sets[0])
language_count = length(XNLI_LANGS)
partition_size = floor(train_size / language_count)

partitions = list()
for index, language_set in enumerate(language_sets):
    start = index * partition_size
    end = (index + 1) * partition_size
    partition = language_set.select(range(start, end))
    partitions.append(partition)

xnli_mix = interleave_datasets(blocks, seed)
```

Figure 3.5: The creation of XNLI$_{mix}$ from the translated training sets of XNLI, `XNLI_LANGS` denotes a list of the 15 languages used in XNLI, `seed` is a parameter

[9] `https://wandb.ai/site`

Chapter 4

Experiments and results

I perform a series of experiments to explore MTL's benefits in the chosen setup. Due to the temporal and computational constraints, I do not test many possible combinations of hyperparameters and methods. The results are reported on validation sets unless stated otherwise.
For reproducibility, I list the used hyperparameters. The effective batch size is invariantly 128. The batch size per GPU is 32 for XLM-R. For mT5, the batch size per device is 16, and, to achieve the desired effective batch size, the gradients are accumulated over two batches. In MTL and LoRA runs, gradient accumulation and batch size per device are changed depending on the memory requirements. The input sequence length is 512 as a compromise between mT5 (pre-trained on the sequence length 1024) and XLM-R (256). The learning rate is constant, 7.5e-6 for XLM-R (Adam) and mBERT (Adam), 1e-3 for mT5 (Adafactor) and 5e-4 for LoRA (Adam). XGLUE tasks and MTL setups train until they reach ten epochs or 20 thousand optimizer steps (in larger experiments, it is extended to 23 thousand steps), and CTKFactsNLI is trained for 100 epochs. Models are evaluated every 200 optimizer steps. Other hyperparameters and their modifications are mentioned when relevant to the experiment.
Hyperparameters are consistent across all experiments except the first one. I conducted it for my semestral project using a different setup. The results are reported on the test sets, provided only for comparison between mBERT and XLM-R, and do not influence subsequent experiments. The models from the old setup are explicitly denoted with a subscript *old*. The effective batch size is 128. The learning rates vary in some cases due to unstable training.
Finally, to increase the speed and lower the memory footprint of training, the precision of matrix multiplication is set to "medium" in PyTorch, meaning `float32` matrix multiplications use the `bfloat16` datatype for internal computations. The results should still be similar if the chosen hardware does not support this feature.

4.1 Multilingual experiments

In order to investigate the discrepancy between training approaches and design choices used for multilingual pre-training, I train and evaluate XLM-

R and mBERT individually on every target language of XNLI and original PAWS-X. This training scheme is called translate-train. Then, models trained only on English data are evaluated in all target languages (cross-lingual zero-shot transfer). The latter scheme will be later used in all MTL experiments unless stated otherwise. With this, I assess the difference between translate-train (the results are in Table 4.1) and cross-lingual zero-shot transfer (see Table 4.2). XLM-R outperforms mBERT in both scenarios. mBERT fails at cross-lingual zero-shot transfer on XNLI. There is a significant average performance drop of **12**. The decline in XLM-R's performance on XNLI is only **3.7**. On PAWS-X, both models underperform in Japanese (ja) and Korean (ko).

model	mBERT_{old}	XLM-R_{old}	mBERT_{old}	XLM-R_{old}
dataset	PAWS-X	PAWS-X	XNLI	XNLI
ar	-	-	69.9	75.5
bg	-	-	74.6	79.6
de	87.0	88.3	76.3	79.9
el	-	-	73.4	79.9
en	93.5	93.9	79.3	84.4
es	89.2	90.4	76.9	80.5
fr	89.5	91.0	76.6	79.8
hi	-	-	66.9	71.9
ru	-	-	74.1	78.7
sw	-	-	65.8	70.7
th	-	-	66.1	77.3
tr	-	-	71.5	76.5
ur	-	-	61.3	67.1
vi	-	-	75.4	78.6
zh	82.8	84.0	76.1	78.1
ja	80.8	81.1	-	-
ko	80.5	81.4	-	-
AVG	86.2	**87.2**	72.3	**77.2**

Table 4.1: Translate-train scheme using the old setup, test set results (AVG stands for average)

In contrast to the translate-train-all scheme, which was reported to boost the performance on XNLI [12], XNLI_{mix}, a dataset constructed from the translated training sets of XNLI, maintains the size of the original XNLI training set. I hypothesize XNLI_{mix} can still be beneficial for overall performance on XNLI when compared to the cross-lingual zero-shot transfer setting. Moreover, this can be used later in an MTL setup as XNLI has the biggest language overlap with the chosen XGLUE datasets. I compare the performance on XNLI_{mix} to XNLI. The results are in Table 4.3. In all models, there is a drop in English performance, which is compensated by the positive gain in all other languages except for mT5's Urdu (ur) score. For mBERT and XLM-R, Spanish (es) has the lowest positive gain. The

model	mBERT_{old}	XLM-R_{old}	mBERT_{old}	XLM-R_{old}
dataset	PAWS-X	PAWS-X	XNLI	XNLI
ar	-	-	58.7	72.5
bg	-	-	61.6	77.1
de	86.2	87.5	65.2	75.8
el	-	-	59.5	75.0
en	93.5	93.9	79.3	84.4
es	87.8	88.0	70.0	78.0
fr	87.6	89.0	67.4	77.8
hi	-	-	54.0	69.3
ru	-	-	63.3	73.7
sw	-	-	45.7	63.9
th	-	-	42.2	71.1
tr	-	-	54.1	72.4
ur	-	-	51.9	65.4
vi	-	-	67.5	73.1
zh	79.0	80.1	63.4	73.7
ja	75.6	75.0	-	-
ko	74.6	72.8	-	-
AVG	83.5	**83.8**	60.3	**73.5**

Table 4.2: Cross-lingual zero-shot transfer using the old setup, test set results

mT5's smallest gain is in Vietnamese (vi) and the biggest is in Chinese (zh). Overall, XNLI_{mix} improves performance in all three cases.

4.2 mT5 experiments

For a quick overview of the results of this series of experiments, refer to Figure 4.1.

Single-task baseline

I set the single-task baseline on XGLUE tasks. Starting with this experiment, the XGLUE version of PAWS-X is used. The results (see Table 4.4) are used as a reference in later experiments.

MTL experiments

Using the initial setup and training on the XGLUE tasks jointy, I present the baseline for mT5 MTL experiments in Table 4.5. The overall score is already close to the single-task baseline. The average English (en) score is higher. I observe negative transfer on NC, PAWS-X and XNLI. On the other hand, QADSM and QAM have a significant positive gain. When computing the overall average, 15 terms out of 37 come from XNLI. Thus, to optimize

model	mBERT	mBERT	XLM-R	XLM-R	mT5	mT5
dataset	XNLI$_{mix}$	XNLI	XNLI$_{mix}$	XNLI	XNLI$_{mix}$	XNLI
ar	67.7	64.8	73.5	72.7	73.5	72.9
bg	73.2	69.5	77.7	76.9	77.6	75.8
de	74.5	71.7	78.8	76.6	77.8	76.3
el	70.3	67.1	78.1	76.1	77.6	76.7
en	78.4	81.1	83.1	85.2	80.8	82.6
es	75.8	75.4	79.8	79.7	78.9	78.4
fr	75.7	74.2	78.5	77.8	78.7	77.7
hi	66.4	62.4	73.5	70.2	72.8	70.8
ru	69.8	68.5	77.6	75.3	75.7	73.9
sw	**61.2**	51.5	**68.5**	65.2	68.5	67.7
th	**64.6**	54.5	76.2	73.2	**74.6**	71.2
tr	69.9	64.6	75.8	72.6	73.6	72.0
ur	64.2	59.1	68.7	66.5	68.1	68.3
vi	72.4	70.1	76.6	74.7	72.9	72.8
zh	73.5	70.6	**77.9**	74.0	**77.1**	73.1
AVG	**70.5**	67.0	**76.3**	74.4	**75.2**	74.0

Table 4.3: A comparison of XNLI$_{mix}$ results to English-only XNLI, results in **bold** highlight some of the significant gains when using XNLI$_{mix}$

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.9	72.9
bg	-	-	-	-	-	75.8	75.8
de	85.4	87.6	63.4	69.0	75.7	76.3	76.2
el	-	-	-	-	-	76.7	76.7
en	93.2	93.3	69.5	68.1	77.2	82.6	**80.6**
es	84.5	90.0	-	-	74.4	78.4	81.8
fr	78.2	91.0	67.4	67.6	73.6	77.7	75.9
hi	-	-	-	-	-	70.8	70.8
it	-	-	-	-	66.6	-	66.6
pt	-	-	-	-	76.5	-	76.5
ru	79.2	-	-	-	-	73.9	76.5
sw	-	-	-	-	-	67.7	67.7
th	-	-	-	-	-	71.2	71.2
tr	-	-	-	-	-	72.0	72.0
ur	-	-	-	-	-	68.3	68.3
vi	-	-	-	-	-	72.8	72.8
zh	-	-	-	-	60.5	73.1	66.8
AVG	84.1	90.5	66.8	68.2	72.1	74.0	**75.7**

Table 4.4: mT5 single-task baseline, results in **bold** highlight the average result in English and the overall average

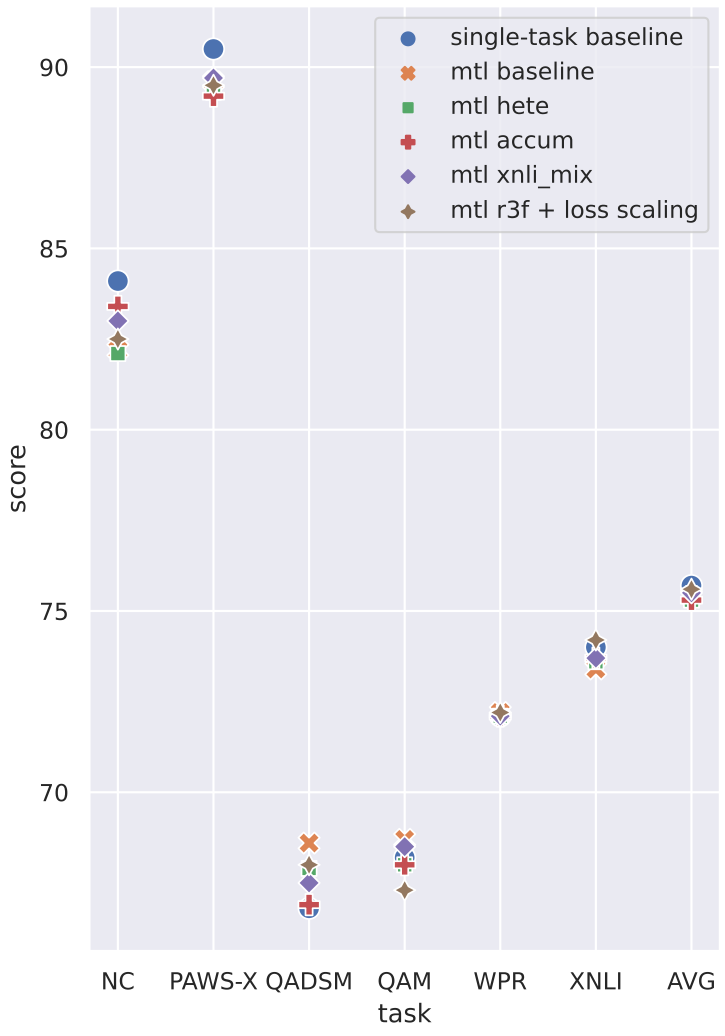


Figure 4.1: Results of mT5 MTL experiments with averages across tasks, where every keyword in ”mtl” entry stands for the modification it adds, ”hete” denotes heterogeneous batches, ”accum” denotes accumulation over more batches

the overall average score, it is crucial to include XNLI in as many optimizer steps as possible.

At each global step, the MTL data loader selects a task from which it produces a batch. Suppose all MTL data loaders in DDP processes are initialized with the same seed. At every step, each data loader chooses the same task. As I accumulate gradients over two batches in the baseline, the effective batch consists of batches from 2 tasks. The first modification introduces more heterogeneity in the effective batch as every data loader receives a unique seed. This way, the task choice in each process does not depend on the rest. Therefore, the effective batch at each step potentially consists of 8 samples from different tasks, which covers my set of 6 XGLUE tasks. I hypothesize this modification may be beneficial for performance as more tasks are included at

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.3	72.3
bg	-	-	-	-	-	76.0	76.0
de	80.4	85.9	65.9	69.5	75.6	75.8	75.5
el	-	-	-	-	-	76.5	76.5
en	92.2	93.8	71.3	69.1	77.5	83.2	**81.2**
es	82.9	88.1	-	-	74.8	78.1	80.9
fr	76.5	90.5	68.7	67.6	73.2	77.6	75.7
hi	-	-	-	-	-	68.9	68.9
it	-	-	-	-	66.6	-	66.6
pt	-	-	-	-	76.9	-	76.9
ru	79.1	-	-	-	-	73.8	76.5
sw	-	-	-	-	-	65.8	65.8
th	-	-	-	-	-	71.5	71.5
tr	-	-	-	-	-	71.0	71.0
ur	-	-	-	-	-	65.8	65.8
vi	-	-	-	-	-	72.4	72.4
zh	-	-	-	-	60.8	72.9	66.9
AVG	82.2	89.6	68.6	68.7	72.2	73.4	**75.4**

Table 4.5: mT5 MTL baseline, results in **bold** highlight the average gain in English and the overall average

each optimizer step. The results are in Table 4.6. The average performance on all tasks, except XNLI, decreases, yet the change in the overall average indicates that XNLI compensates for the losses on other tasks. This modification is necessary for creating a task-heterogeneous effective batch when gradient accumulation is not utilized. Besides, the authors of [34] advocate for this modification. I keep it as I expect additional gains when introducing the other methods. In the later MTL experiments, all models use the heterogeneous batches modification.
I then add the mT5's MLM task to the training process with a chance of sampling once in every 100 batches. As the loss of the MLM task is usually larger, and the task itself is unrelated to TC tasks, I also test if R3F with loss scaling may be helpful in this case. I provide only the average scores (see Table 4.7). MLM brings no benefit to my setup. Even with loss scaling over the vocabulary size, the score is lower. I continue without the MLM task.
Building on the heterogeneous batches, the second modification changes the number of batches per GPU to 8 and gradient accumulation batches to 4. With this change, the training step requires roughly half GPU memory. Besides, there is a potential of sampling smaller batches from more tasks, though the chosen task set is already covered without this change. The initial setup without this modification in conjunction with the R3F method runs out of memory, thus it is necessary to test whether this modification harms performance. The average results are in Table 4.8. The only significant changes are a positive gain on NC and a drop on QADSM. The setup

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.7	72.7
bg	-	-	-	-	-	77.0	77.0
de	80.9	86.3	66.6	68.8	75.5	75.6	75.6
el	-	-	-	-	-	76.3	76.3
en	92.0	92.3	69.8	68.0	77.7	82.4	80.4
es	82.2	88.9	-	-	74.6	77.9	80.9
fr	76.1	89.9	67.4	67.3	73.5	77.4	75.3
hi	-	-	-	-	-	70.0	70.0
it	-	-	-	-	65.9	-	65.9
pt	-	-	-	-	76.8	-	76.8
ru	79.2	-	-	-	-	74.3	76.8
sw	-	-	-	-	-	65.5	65.5
th	-	-	-	-	-	71.0	71.0
tr	-	-	-	-	-	72.0	72.0
ur	-	-	-	-	-	66.9	66.9
vi	-	-	-	-	-	72.2	72.2
zh	-	-	-	-	60.6	73.1	66.9
AVG	82.1	89.3	67.9	68.0	72.1	73.6	**75.3**

Table 4.6: mT5 MTL more heterogeneous batches, the result in **bold** highlights the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
MLM	83.4	89.4	66.8	67.7	72.0	72.6	74.9
+ *R3F*							
+ *loss scaling*	83.8	86.9	68.2	67.5	72.4	73.4	75.2

Table 4.7: mT5 MTL average results with MLM mixed in, the second row of results is MLM with R3F and loss scaling

is sustainable, and the overall average is the same.

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
AVG	83.4	89.2	66.9	68.0	72.1	73.7	**75.3**

Table 4.8: mT5 MTL average results with more gradient accumulation, the result in **bold** highlights the overall average

The next step involves replacing the XNLI training set with XNLI$_{mix}$. I anticipate positive gains across tasks that share target languages with XNLI and some improvement on XNLI-exclusive low-resource languages. The results are presented in Table 4.9. My first hypothesis fails due to the drop on QADSM (**68.6** baseline vs. **67.5**) and QAM (**68.7** baseline vs. **68.5**). However, XNLI$_{mix}$ elevates several languages, not only low-resource ones. Overall, the result is positive, I continue with XNLI$_{mix}$.
The final modification introduces the R3F method with loss scaling. It was reported in [34] that these methods (including heterogeneous batches) are

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	70.6	70.6
bg	-	-	-	-	-	76.4	76.4
de	82.4	85.5	64.3	69.2	76.2	74.8	75.4
el	-	-	-	-	-	75.9	75.9
en	93.3	94.1	71.2	68.8	77.5	81.2	81.0
es	82.6	89.0	-	-	74.6	77.5	80.9
fr	77.1	90.0	67.0	67.6	73.2	76.9	75.3
hi	-	-	-	-	-	**70.3**	70.3
it	-	-	-	-	65.7	-	65.7
pt	-	-	-	-	77.1	-	77.1
ru	79.7	-	-	-	-	74.1	76.9
sw	-	-	-	-	-	**68.0**	68.0
th	-	-	-	-	-	**72.7**	72.7
tr	-	-	-	-	-	**71.5**	71.5
ur	-	-	-	-	-	**68.4**	68.4
vi	-	-	-	-	-	**72.5**	72.5
zh	-	-	-	-	60.7	**74.2**	67.5
AVG	83.0	89.7	67.5	68.5	72.1	73.7	**75.5**

Table 4.9: mT5 MTL results when XNLI training set is replaced with XNLI$_{mix}$, the results in **bold** highlight the significatnt gains on XNLI and the overall average

effective on larger sets of tasks. I treat XNLI$_{mix}$ as a collection of separate low-resource tasks. For R3F, the noise is sampled from a uniform distribution with σ set to 1e-5, and the weight λ is set to 1e-2. The results are listed in Table 4.10. Surprisingly, average English (en) performance degrades, especially on QADSM (**71.2** previously vs. **66.5**). On the other hand, the overall performance is higher. If I replace the XNLI results in the single-task baseline with the single-task XNLI$_{mix}$ result, the final score of the single-task setup becomes **76.2**. My final mT5 MTL setup loses against the single-task setup by **0.6**.

4.3 XLM-R experiments

Single-task baseline

The single-task baseline is presented in Table 4.11. Instantly, XLM-R has better performance than mT5 in a single-task setup.

MTL experiments

Although I do not accumulate batches in XLM-R's initial setup, I introduce heterogeneous batches in the first run. The results are in Table 4.12. The overall score is similar to what was observed on mT5 with more heterogeneous

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	71.8	71.8
bg	-	-	-	-	-	76.9	76.9
de	82.3	85.9	68.7	68.5	75.6	76.1	76.2
el	-	-	-	-	-	76.0	76.0
en	92.7	92.9	66.5	66.1	77.5	80.7	79.4
es	82.2	89.3	-	-	74.6	77.7	81.0
fr	76.5	89.8	68.8	67.2	73.6	77.3	75.5
hi	-	-	-	-	-	70.8	70.8
it	-	-	-	-	66.3	-	66.3
pt	-	-	-	-	77.4	-	77.4
ru	78.9	-	-	-	-	**75.1**	77.0
sw	-	-	-	-	-	69.9	69.9
th	-	-	-	-	-	**73.7**	73.7
tr	-	-	-	-	-	71.8	71.8
ur	-	-	-	-	-	66.9	66.9
vi	-	-	-	-	-	**73.3**	73.3
zh	-	-	-	-	60.7	**75.5**	68.1
AVG	82.5	89.5	68.0	67.3	72.2	**74.2**	**75.6**

Table 4.10: mT5 MTL results with XNLI$_{mix}$, R3F and loss scaling, the results in **bold** highlight the significant gains on XNLI and the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.7	72.7
bg	-	-	-	-	-	76.9	76.9
de	86.0	87.0	64.4	67.7	76.1	76.6	76.3
el	-	-	-	-	-	76.1	76.1
en	92.5	93.0	71.1	70.2	77.5	**85.2**	81.6
es	84.6	88.3	-	-	74.8	79.7	81.9
fr	78.5	89.0	68.2	66.0	73.8	77.8	75.6
hi	-	-	-	-	-	70.2	70.2
it	-	-	-	-	66.3	-	66.3
pt	-	-	-	-	77.3	-	77.3
ru	79.4	-	-	-	-	75.3	77.3
sw	-	-	-	-	-	65.2	65.2
th	-	-	-	-	-	73.2	73.2
tr	-	-	-	-	-	72.6	72.6
ur	-	-	-	-	-	66.5	66.5
vi	-	-	-	-	-	74.7	74.7
zh	-	-	-	-	61.0	74.0	67.5
AVG	84.2	89.3	67.9	68.0	72.4	74.4	**75.9**

Table 4.11: XLM-R single-task baseline, the results in **bold** highlight the best result on English XNLI data across experiments and the overall average

batches.

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	71.7	71.7
bg	-	-	-	-	-	76.6	76.6
de	86.3	83.7	63.6	68.2	75.6	76.9	75.7
el	-	-	-	-	-	75.7	75.7
en	92.2	92.2	70.2	68.4	77.2	83.7	80.7
es	84.9	86.9	-	-	74.3	78.7	81.2
fr	78.0	86.2	65.9	65.9	73.8	77.1	74.5
hi	-	-	-	-	-	69.3	69.3
it	-	-	-	-	66.3	-	66.3
pt	-	-	-	-	76.9	-	76.9
ru	79.6	-	-	-	-	74.6	77.1
sw	-	-	-	-	-	64.2	64.2
th	-	-	-	-	-	72.9	72.9
tr	-	-	-	-	-	73.4	73.4
ur	-	-	-	-	-	66.1	66.1
vi	-	-	-	-	-	73.7	73.7
zh	-	-	-	-	60.6	74.6	67.6
AVG	84.2	87.3	66.5	67.5	72.1	73.9	**75.3**

Table 4.12: XLM-R MTL with heterogeneous batches, the result in **bold** highlights the overall average

Then, I combine the steps I applied to mT5: increasing the number of gradient accumulation batches, replacing XNLI with XNLI$_{mix}$, applying R3F and loss scaling. The average performance should be similar to the final mT5 MTL performance and have better XNLI results than the previous experiment. The results are in Table 4.13. There is a significant drop on NC and decent positive gains on PAWS-X and QADSM. Performance on XNLI is significantly higher, contributing to my setup's highest final MTL score. However, if the single-task XNLI$_{mix}$ result is taken into consideration, the overall single-task average for XLM-R is **76.7**, which outperforms the final MTL setup by **1** point.

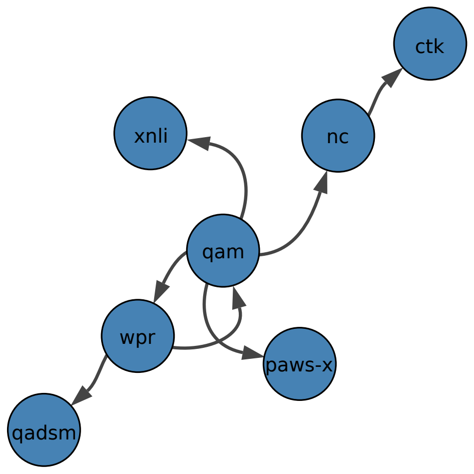
4.4 CTKFactsNLI experiments

In this series of experiments on the CTKFactsNLI task, I evaluate the benefit of MTL pre-finetuning by comparing the pre-finetuned models with several training schemes. First, I fine-tune XLM-R and mT5 only on the target task. Second, I perform transfer learning from the best source task according to TaskEmb and TextEmb. TaskEmb of each task is computed using a model that is fine-tuned for three epochs. The results are presented as oriented graphs, where every oriented edge points to the target task from the best intermediate task, "ctk" denotes CTKFactsNLI: TextEmb results are in Figure 4.2, TaskEmb results are in Figure 4.3 and the aggregated version,

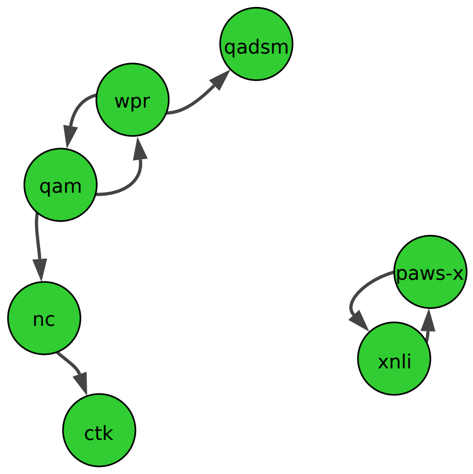
dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	73.5	73.5
bg	-	-	-	-	-	77.0	77.0
de	82.8	85.3	66.8	68.2	75.2	77.9	76.0
el	-	-	-	-	-	76.8	76.8
en	85.2	94.2	69.3	69.2	77.3	81.6	79.4
es	80.3	88.8	-	-	74.4	78.2	80.4
fr	76.1	89.9	68.0	66.1	73.3	77.7	75.2
hi	-	-	-	-	-	73.3	73.3
it	-	-	-	-	66.2	-	66.2
pt	-	-	-	-	77.3	-	77.3
ru	78.8	-	-	-	-	75.6	77.2
sw	-	-	-	-	-	66.9	66.9
th	-	-	-	-	-	75.4	75.4
tr	-	-	-	-	-	73.3	73.3
ur	-	-	-	-	-	69.6	69.6
vi	-	-	-	-	-	75.7	75.7
zh	-	-	-	-	60.4	76.1	68.3
AVG	80.6	89.5	68.0	67.8	72.0	**75.3**	**75.7**

Table 4.13: XLM-R MTL results with XNLI$_{mix}$, R3F and loss scaling, the results in **bold** highlight the significant gain on XNLI and the overall average

TaskEmb + TextEmb, is presented in Figure 4.4. TextEmb in both models indicates that NC is the closest task to CTKFactsNLI based on linguistic properties. However, TaskEmb is more relevant as it promises to capture the semantic properties of the tasks. In XLM-R, the best intermediate task for CTKFactsNLI is XNLI. The trained classification head is dropped, and the CTKFactsNLI classification head is initialized. In mT5, it is QAM. For CTKFactsNLI, the aggregated version does not change any of the TaskEmb's results. I take the required models from the single-task baselines and fine-tune them on CTKFactsNLI for 100 epochs. Third, instead of fine-tuning mT5, I utilize LoRA with inner dimension and scaling factor both set to 32 and dropout probability of 0.1. The results are in Table 4.14. XLM-R demonstrates that, in this setup, the transfer from the best source task is as effective as pre-finetuning. More importantly, both schemes are beneficial for the resulting performance. On the other hand, transferring from QAM degrades the resulting performance of mT5. LoRA successfully adapts when the task is presented in the XNLI format. LoRA is not able to learn the target task with no pre-finetuning. For the two XLM-R runs with the highest scores, the best mT5 run and the best LoRA run, I provide the results on the test set (see Table 4.15). The highest reported result on CTKFactsNLI in the original paper [2] is **76.9**. It was achieved with XLM-R$_{\text{large}}$ trained on the question-answering task SQuAD2.0 [46] before final fine-tuning. At the time of writing, the highest reported result on the test set is **80.7** [47]. The setup used the RobeCzech model [48] in conjunction with active learning



(a) : Based on TextEmb embeddings extracted from the mT5's encoder



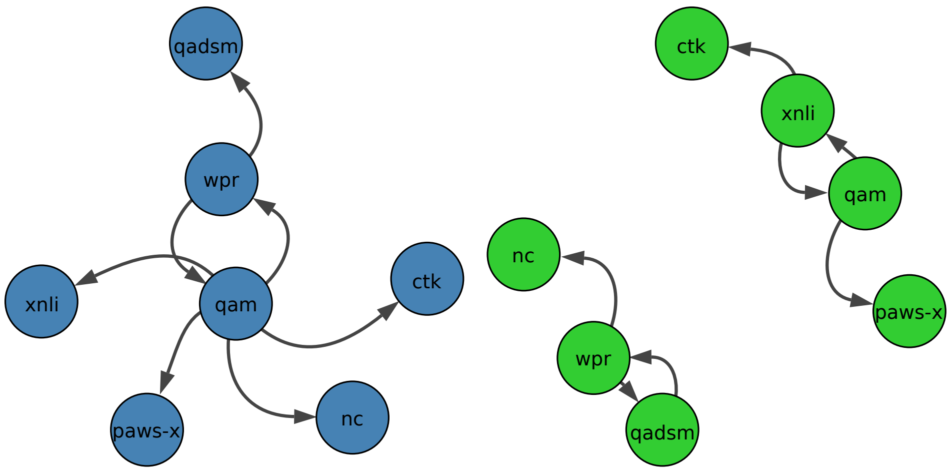
(b) : Based on TextEmb embeddings extracted from the XLM-R's encoder. The graph has two components as XNLI is the best intermediate task for PAWS-X and vice versa

Figure 4.2: TextEmb results, the edges point to the target task from the best intermediate task, "ctk" denotes CTKFactsNLI

methods. Judging by the test results of my experiments, my pre-finetuned mT5 sets a new record on CTKFactsNLI with **81.9**.

4.5 Test set results

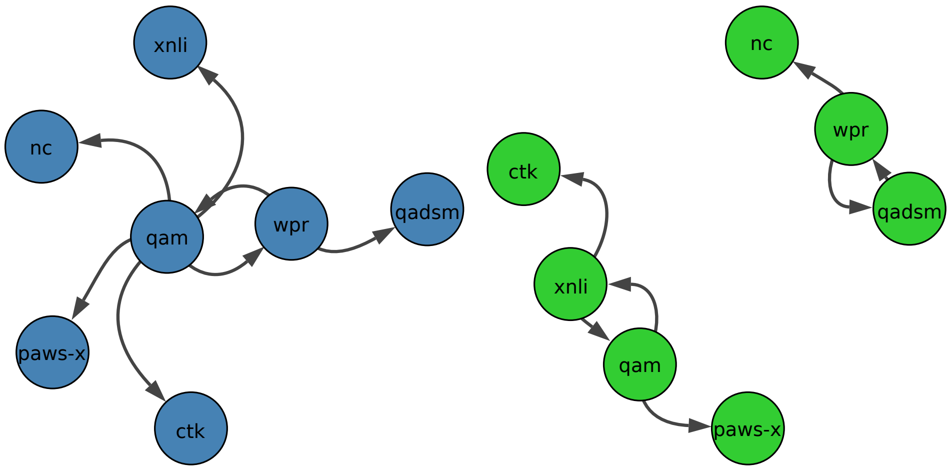
I additionally evaluate the single-task and MTL baselines and the final MTL models on the test sets. The tables are presented in Appendix A. mT5's MTL baseline outperforms the final MTL version. The latter sees a significant drop on QAM (**69.8** vs. **67.2**). XLM-R with R3F and loss scaling performs better compared to the heterogeneous batch MTL baseline (**75.8** vs. **75.3**).
On average, single-task baselines still outperform the MTL models. For mT5, as can be seen in Figure 4.5, QADSM is the only task that benefits from MTL with a gain of **5.1** in German (de) and **1.3** average gain on that task. In XLM-R's final MTL version, PAWS-X and XNLI benefit from MTL, while NC has a large drop (**7**) in English (en) performance and **3.4** drop on that task (see Figure 4.6).
Lastly, I compare the final MTL versions of XLM-R and mT5 (see Figure 4.7). The overall result is similar, mT5 performs better on NC and WPR, while XLM-R produces stronger results on the rest of the tasks. XLM-R outperforms mT5 in this setting by **0.1**.



(a) : Based on TaskEmb embeddings extracted from the mT5's encoder

(b) : Based on TaskEmb embeddings extracted from the XLM-R's encoder

Figure 4.3: TaskEmb results, the edges point to the target task from the best intermediate task, "ctk" denotes CTKFactsNLI



(a) : Based on the aggregated embeddings extracted from the mT5's encoder

(b) : Based on the aggregated embeddings extracted from the XLM-R's encoder

Figure 4.4: TaskEmb + TextEmb aggregated results, the edges point to the target task from the best intermediate task, "ctk" denotes CTKFactsNLI

model	methods	F1
XLM-R	-	68.4
	transfer from XNLI	**74.5**
	pre-finetuned	**74.6**
mT5	-	73.1
	transfer from QAM	70.9
	pre-finetuned	73.8
	pre-finetuned + R3F + loss scaling	**74.1**
	pre-finetuned + XNLI format	73.1
	pre-finetuned + XNLI format + R3F + loss scaling	74.0
	LoRA	54.1
	pre-finetuned + LoRA	64.7
	pre-finetuned + XNLI format + LoRA	**73.0**

Table 4.14: results on CTKFactsNLI, the results in **bold** highlight significant scores across models and methods

model	methods	F1
XLM-R	transfer from XNLI	**81.2**
	pre-finetuned	79.1
mT5	pre-finetuned + R3F + loss scaling	**81.9**
	pre-finetuned + XNLI format + LoRA	**81.5**

Table 4.15: results on CTKFactsNLI, test set results, the results in **bold** highlight scores higher than the previous SOTA

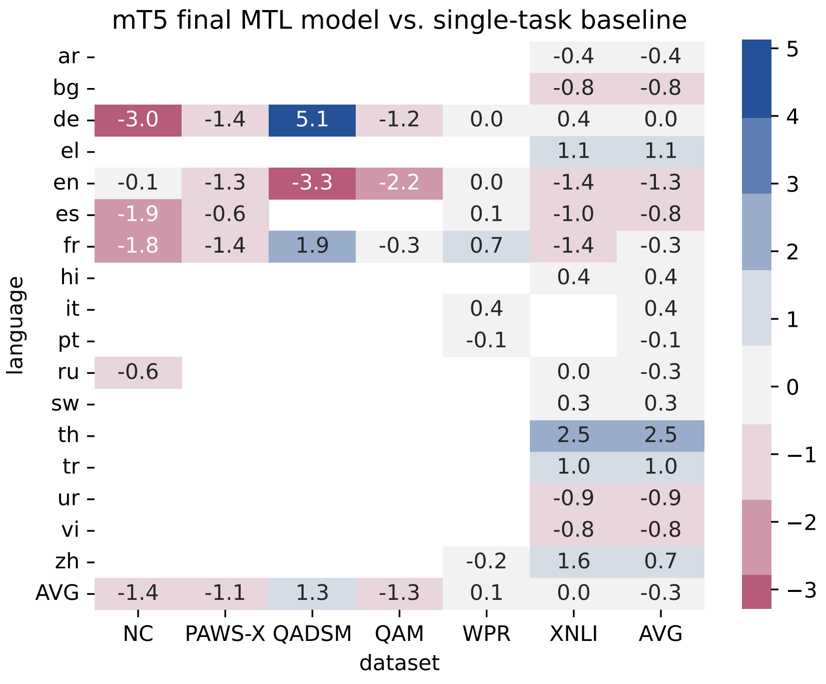


Figure 4.5: Comparison of the final mT5 MTL model with the single-task baseline, test set results

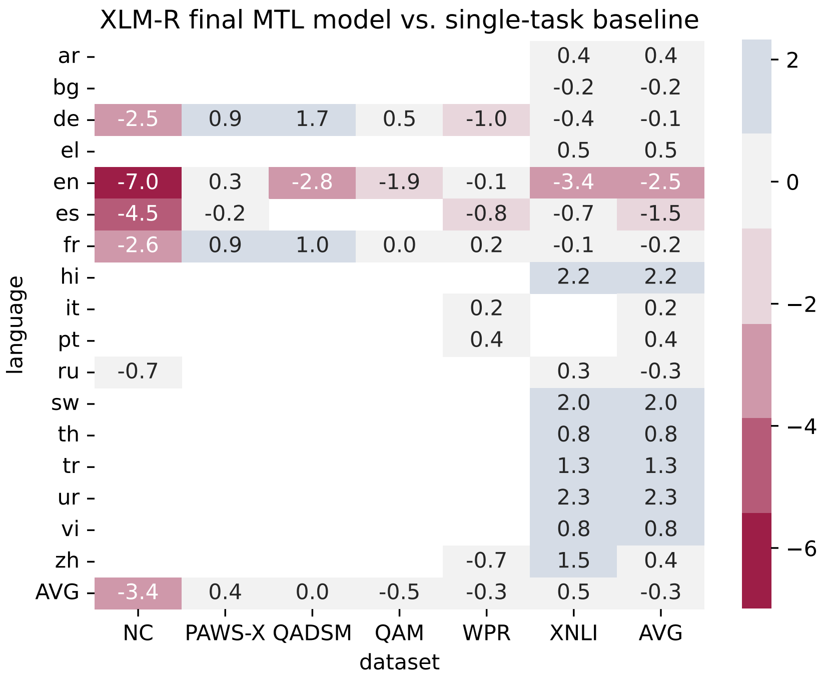


Figure 4.6: Comparison of the final XLM-R MTL model with the single-task baseline, test set results

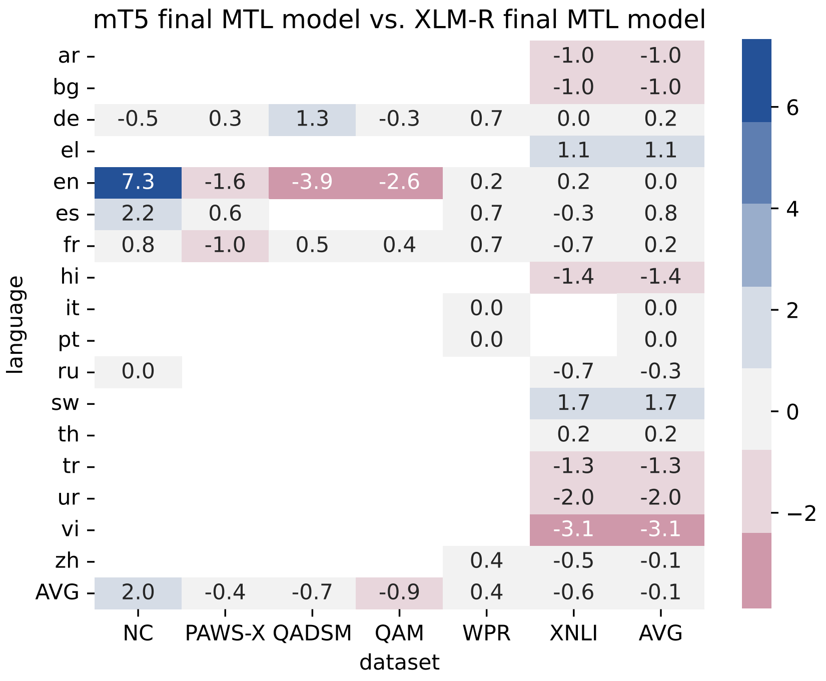


Figure 4.7: Comparison of the final mT5 MTL model with the final XLM-R MTL model, test set results

Chapter 5
Discussion

In the initial experiments, the models trained on the English training set of PAWS-X underperformed in Japanese (ja) and Korean (ko), which indicates that low lexical overlap between English and target languages degrades performance. With XNLI$_{mix}$, I saw a decent performance boost on each trained model. However, in MTL experiments, this did not benefit other tasks, only resulting in better performance on XNLI-exclusive languages. While XNLI$_{mix}$ improved mT5's overall MTL performance, the effect was mixed: some XNLI-exclusive languages saw a drop in performance, and some saw a significant boost. The mT5's MTL baseline had a better average score in English (en) than its single-task baseline. The MTL model performed more optimizer steps on English data, which might have been the cause. However, it would contradict the findings of the subsequent MTL experiments, where English performance got worse (as in the last mT5 MTL experiment). Apart from that, I did not find any significant language-specific pattern. Mixing the mT5's MLM task has been fruitless, I suppose it has no positive effect on TC performance in general. The final combination of XNLI$_{mix}$ with R3F and loss scaling resulted in the best-performing models on validation sets.
Ultimately, the performance gap between MTL models and their single-task counterparts was not closed. Pre-finetuning improved the target task performance and effectively allowed LoRA to learn at all. The similarity between XNLI and CTKFactsNLI allowed for the reformulation trick, further improving LoRA's performance. TaskEmb failed on mT5 by predicting QAM, possibly due to a low amount of training conducted for this method (each task for three epochs). Nevertheless, on XLM-R, it determined an intuitively correct intermediate task, XNLI, transferring from which provided the best result. It could mean that, for XLM-R, the chosen XGLUE tasks lead to negative transfer when learned jointly. Both of my XLM-R$_{base}$ models produced better results than had been reported on XLM-R$_{large}$ in the original paper [2], which might have been caused by an unfortunate choice of hyperparameters in the latter case. My pre-finetuned mT5 has updated the SOTA on CTKFactsNLI, yet the previous SOTA result [47] was achieved with a significantly smaller model (mT5$_{base}$ has 582 million parameters, RobeCzech has 125 million). Even though this aligns with a common notion that larger models perform better, RobeCzech is a monolingual Czech model, while the

Czech language occupies only 1.72% of mT5's pre-training volume.

Chapter 6
Conclusion

In this work, I investigated the application of MTL methods to multilingual transformers to solve multiple TC tasks. While the joint MTL experiments on the chosen set of tasks produced competitive models, the single-task fine-tuning still outperformed the MTL models. However, MTL pre-finetuning has proven to be beneficial for the performance on the target task. In case of XLM-R, transferring from an intermediate task predicted by TaskEmb embeddings outperformed the MTL model. I have demonstrated that pre-finetuning provides a strong basis to build upon when using parameter-efficient fine-tuning methods, such as LoRA. Finally, the pre-finetuned mT5 has updated the SOTA on CTKFactsNLI.

MTL is a complex yet exciting paradigm. It is one of the components that modern language models heavily rely on. MTL, combined with multilinguality, produces models that are more universal in their capabilities (compared to single-task models) and are easier to use for non-English speakers. I hope my work has provided a decent overview of what MTL has to offer, even in a small-scale multilingual setup.

It has been an excellent opportunity to learn about these topics. I have read many research papers and blog posts on various topics to gather the materials for this work. The implementation process has been chaotic, and the setup has been rewritten multiple times. On the other hand, I have gained some experience working with PyTorch Lightning and Hugging Face libraries. Adapting and implementing the code from the research papers has been a great challenge that has helped me be more critical and attentive to details. I have explored multiple methods and hyperparameter choices in this work, which has developed my intuition for the training process. Finally, I have spent many hours debugging the code, planning and tracking the experiments, which has been stressful, yet worth the time and effort.

Chapter 7

Future works

Regarding this work, I plan to rewrite the code in a cleaner format, raise several issues (including the fix for R3F and the inconsistency of R4F) and answer the questions on online forums about basic MTL setups. Besides, It would be beneficial to try other learning strategies, such as fully sharded data parallel (FSDP).

This work has combined several topics that I am interested in. First, I would like to research the possibilities of multilingual models further. There is still a gap in filtering the colossal data sources for pre-training. A more systematic approach is desired. Second, MTL is being widely adopted in modern models. The development of new MTL-related methods and frameworks happens at an overwhelmingly rapid pace. The new wave of instruction fine-tuning methods [49] offers an excellent opportunity for exploration and experimentation. Another direction I am excited about is the application of LoRA and similar parameter-efficient fine-tuning methods to existing large pre-trained models.

Appendix A

Test set results

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.9	72.9
bg	-	-	-	-	-	77.6	77.6
de	84.9	89.1	63.2	69.1	76.7	76.2	76.5
el	-	-	-	-	-	76.1	76.1
en	92.8	94.2	69.1	67.4	77.6	82.9	80.6
es	83.8	89.3	-	-	74.7	78.7	81.6
fr	78.6	90.0	68.3	68.9	73.6	78.0	76.2
hi	-	-	-	-	-	70.5	70.5
it	-	-	-	-	68.5	-	68.5
pt	-	-	-	-	75.3	-	75.3
ru	79.5	-	-	-	-	75.7	77.6
sw	-	-	-	-	-	68.8	68.8
th	-	-	-	-	-	71.5	71.5
tr	-	-	-	-	-	71.5	71.5
ur	-	-	-	-	-	67.8	67.8
vi	-	-	-	-	-	73.5	73.5
zh	-	-	-	-	61.8	73.6	67.7
AVG	83.9	90.6	66.8	68.5	72.6	74.4	**76.0**

Table A.1: single-task mT5 baseline, test set results, the result in **bold** highlights the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	73.2	73.2
bg	-	-	-	-	-	77.3	77.3
de	79.7	88.9	66.1	71.1	77.1	75.9	76.5
el	-	-	-	-	-	76.2	76.2
en	91.9	94.1	70.9	68.9	78.3	83.4	81.3
es	82.0	90.1	-	-	75.4	78.5	81.5
fr	76.3	89.5	69.9	69.3	74.3	77.3	76.1
hi	-	-	-	-	-	69.0	69.0
it	-	-	-	-	68.9	-	68.9
pt	-	-	-	-	75.4	-	75.4
ru	79.3	-	-	-	-	75.6	77.4
sw	-	-	-	-	-	66.5	66.5
th	-	-	-	-	-	72.2	72.2
tr	-	-	-	-	-	70.6	70.6
ur	-	-	-	-	-	66.1	66.1
vi	-	-	-	-	-	73.5	73.5
zh	-	-	-	-	61.8	72.9	67.3
AVG	81.8	90.6	69.0	69.8	73.0	73.9	**75.9**

Table A.2: MTL mT5 baseline, test set results, the result in **bold** highlights the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	72.5	72.5
bg	-	-	-	-	-	76.8	76.8
de	81.9	87.7	**68.3**	67.9	76.7	76.6	76.5
el	-	-	-	-	-	77.2	77.2
en	92.7	92.9	65.8	65.2	77.6	81.5	79.3
es	81.9	88.7	-	-	74.8	77.7	80.8
fr	76.8	88.6	70.2	68.6	74.3	76.6	75.9
hi	-	-	-	-	-	70.9	70.9
it	-	-	-	-	68.9	-	68.9
pt	-	-	-	-	75.2	-	75.2
ru	78.9	-	-	-	-	75.7	77.3
sw	-	-	-	-	-	69.1	69.1
th	-	-	-	-	-	74.0	74.0
tr	-	-	-	-	-	72.5	72.5
ur	-	-	-	-	-	66.9	66.9
vi	-	-	-	-	-	72.7	72.7
zh	-	-	-	-	61.6	75.2	68.4
AVG	82.5	89.5	68.1	67.2	72.7	74.4	**75.7**

Table A.3: MTL mT5 final version (more heterogeneous batches + XNLI$_{mix}$ + R3F + loss scaling), test set results, the results in **bold** highlight a significant boost in German (de) on QADSM and the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	73.1	73.1
bg	-	-	-	-	-	78.0	78.0
de	84.9	86.5	65.3	67.7	77.0	77.0	76.4
el	-	-	-	-	-	75.6	75.6
en	92.4	94.2	72.5	69.7	77.5	84.7	81.8
es	84.2	88.3	-	-	74.9	78.7	81.5
fr	78.6	88.7	68.7	68.2	73.4	77.4	75.9
hi	-	-	-	-	-	70.1	70.1
it	-	-	-	-	68.7	-	68.7
pt	-	-	-	-	74.8	-	74.8
ru	79.6	-	-	-	-	76.1	77.9
sw	-	-	-	-	-	65.4	65.4
th	-	-	-	-	-	73.0	73.0
tr	-	-	-	-	-	72.5	72.5
ur	-	-	-	-	-	66.6	66.6
vi	-	-	-	-	-	75.0	75.0
zh	-	-	-	-	61.9	74.2	68.1
AVG	83.9	89.5	68.8	68.6	72.6	74.5	**76.1**

Table A.4: single-task XLM-R baseline, test set results, the result in **bold** highlights the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	71.7	71.7
bg	-	-	-	-	-	76.6	76.6
de	86.3	83.7	63.6	68.2	75.6	76.9	75.7
el	-	-	-	-	-	75.7	75.7
en	92.2	92.2	70.2	68.4	77.2	83.7	80.7
es	84.9	86.9	-	-	74.3	78.7	81.2
fr	78.0	86.2	65.9	65.9	73.8	77.1	74.5
hi	-	-	-	-	-	69.3	69.3
it	-	-	-	-	66.3	-	66.3
pt	-	-	-	-	76.9	-	76.9
ru	79.6	-	-	-	-	74.6	77.1
sw	-	-	-	-	-	64.2	64.2
th	-	-	-	-	-	72.9	72.9
tr	-	-	-	-	-	73.4	73.4
ur	-	-	-	-	-	66.1	66.1
vi	-	-	-	-	-	73.7	73.7
zh	-	-	-	-	60.6	74.6	67.6
AVG	84.2	87.3	66.5	67.5	72.1	73.9	**75.3**

Table A.5: MTL XLM-R (heterogeneous batches), test set results, the result in **bold** highlights the overall average

dataset	NC	PAWS-X	QADSM	QAM	WPR	XNLI	AVG
ar	-	-	-	-	-	73.5	73.5
bg	-	-	-	-	-	77.8	77.8
de	82.4	87.4	67.0	68.2	76.0	76.6	76.3
el	-	-	-	-	-	76.1	76.1
en	85.4	94.5	69.7	67.8	77.4	81.3	79.3
es	79.7	88.1	-	-	74.1	78.0	80.0
fr	76.0	89.6	69.7	68.2	73.6	77.3	75.7
hi	-	-	-	-	-	72.3	72.3
it	-	-	-	-	68.9	-	68.9
pt	-	-	-	-	75.2	-	75.2
ru	78.9	-	-	-	-	76.4	77.6
sw	-	-	-	-	-	67.4	67.4
th	-	-	-	-	-	73.8	73.8
tr	-	-	-	-	-	73.8	73.8
ur	-	-	-	-	-	68.9	68.9
vi	-	-	-	-	-	75.8	75.8
zh	-	-	-	-	61.2	75.7	68.5
AVG	80.5	**89.9**	68.8	68.1	72.3	**75.0**	**75.8**

Table A.6: MTL XLM-R final version (heterogeneous batches + XNLI$_{mix}$ + R3F + loss scaling), test set results, the results in **bold** highlight the average scores of the tasks that benefit from MTL and the overall average

Appendix B

Bibliography

[1] Rich Caruana. “Multitask Learning”. In: *Machine Learning* 28.1 (July 1997), pp. 41–75. ISSN: 1573-0565. DOI: 10.1023/A:1007379606734.

[2] Herbert Ullrich et al. “CsFEVER and CTKFacts: acquiring Czech data for fact verification”. en. In: *Language Resources and Evaluation* (May 2023). ISSN: 1574-0218. DOI: 10.1007/s10579-023-09654-3.

[3] Yaobo Liang et al. *XGLUE: A New Benchmark Dataset for Cross-lingual Pre-training, Understanding and Generation.* Issue: arXiv:2004.01401 arXiv: 2004.01401 [cs]. May 2020.

[4] Shervin Minaee et al. *Deep Learning Based Text Classification: A Comprehensive Review.* Issue: arXiv:2004.03705 arXiv: 2004.03705 [cs, stat]. Jan. 2021.

[5] Ashish Vaswani et al. *Attention Is All You Need.* Issue: arXiv:1706.03762 arXiv: 1706.03762 [cs]. Dec. 2017.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* arXiv:1409.0473 [cs, stat]. May 2016.

[7] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928.

[8] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Issue: arXiv:1810.04805 arXiv: 1810.04805 [cs]. May 2019.

[9] Pratik Joshi et al. *The State and Fate of Linguistic Diversity and Inclusion in the NLP World.* arXiv:2004.09095 [cs]. Jan. 2021.

[10] Telmo Pires, Eva Schlinger, and Dan Garrette. *How multilingual is Multilingual BERT?* Issue: arXiv:1906.01502 arXiv: 1906.01502 [cs]. June 2019.

[11] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* arXiv:1907.11692 [cs]. July 2019.

[12] Alexis Conneau et al. *Unsupervised Cross-lingual Representation Learning at Scale.* arXiv:1911.02116 [cs]. Apr. 2020.

[13] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Issue: arXiv:1910.10683 arXiv: 1910.10683 [cs, stat]. July 2020.

[14] Linting Xue et al. *mT5: A massively multilingual pre-trained text-to-text transformer*. Issue: arXiv:2010.11934 arXiv: 2010.11934 [cs]. Mar. 2021.

[15] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. arXiv:1808.06226 [cs]. Aug. 2018.

[16] Taku Kudo. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 66–75. DOI: `10.18653/v1/P18-1007`.

[17] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. arXiv:1508.07909 [cs] version: 5. June 2016.

[18] Joseph Worsham and Jugal Kalita. "Multi-task learning for natural language processing in the 2020s: where are we going?" In: *Pattern Recognition Letters* 136 (Aug. 2020). arXiv: 2007.16008 [cs], pp. 120–126. ISSN: 01678655. DOI: `10.1016/j.patrec.2020.05.031`.

[19] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks*. Issue: arXiv:1706.05098 arXiv: 1706.05098 [cs, stat]. June 2017.

[20] Xiaodong Liu et al. *Multi-Task Deep Neural Networks for Natural Language Understanding*. Issue: arXiv:1901.11504 arXiv: 1901.11504 [cs]. May 2019.

[21] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. Feb. 2023.

[22] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019.

[23] Zhihan Zhang et al. *A Survey of Multi-task Learning in Natural Language Processing: Regarding Task Relatedness and Training Methods*. Issue: arXiv:2204.03508 arXiv: 2204.03508 [cs]. Apr. 2022.

[24] Junjie Hu et al. *XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization*. Issue: arXiv:2003.11080 arXiv: 2003.11080 [cs]. Sept. 2020.

[25] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. arXiv:2009.03300 [cs]. Jan. 2021.

[26] Trevor Standley et al. *Which Tasks Should Be Learned Together in Multi-task Learning?* arXiv:1905.07553 [cs]. Sept. 2020.

[27] Alessandro Achille et al. *Task2Vec: Task Embedding for Meta-Learning.* arXiv:1902.03545 [cs, stat]. Feb. 2019.

[28] Tu Vu et al. *Exploring and Predicting Transferability across NLP Tasks.* arXiv:2005.00770 [cs]. Oct. 2020.

[29] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. "Reciprocal rank fusion outperforms condorcet and individual rank learning methods". en. In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval.* Boston MA USA: ACM, July 2009, pp. 758–759. ISBN: 978-1-60558-483-6. DOI: `10.1145/1571941.1572114`.

[30] Damien Sileo and Marie-Francine Moens. *Analysis and Prediction of NLP Models Via Task Embeddings.* arXiv:2112.05647 [cs]. Dec. 2021.

[31] Jonathan Pilault, Amine Elhattami, and Christopher Pal. *Conditionally Adaptive Multi-Task Learning: Improving Transfer Learning in NLP Using Fewer Parameters & Less Data.* Issue: arXiv:2009.09139 arXiv: 2009.09139 [cs, stat]. Apr. 2022.

[32] Tianhe Yu et al. *Gradient Surgery for Multi-Task Learning.* arXiv:2001.06782 [cs, stat]. Dec. 2020.

[33] Yu Zhang and Qiang Yang. "A Survey on Multi-Task Learning". In: *IEEE Trans. Knowl. Data Eng.* 34.12 (Dec. 2022), pp. 5586–5609. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: `10.1109/TKDE.2021.3070203`.

[34] Armen Aghajanyan et al. *Muppet: Massive Multi-task Representations with Pre-Finetuning.* arXiv:2101.11038 [cs]. Jan. 2021.

[35] Michael McCloskey and Neal J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". en. In: *Psychology of Learning and Motivation.* Ed. by Gordon H. Bower. Vol. 24. Academic Press, Jan. 1989, pp. 109–165. DOI: `10.1016/S0079-7421(08)60536-8`.

[36] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP.* arXiv:1902.00751 [cs, stat]. June 2019.

[37] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv:2106.09685 [cs]. Oct. 2021.

[38] Armen Aghajanyan et al. *Better Fine-Tuning by Reducing Representational Collapse.* arXiv:2008.03156 [cs, stat]. Aug. 2020.

[39] Alexis Conneau et al. *XNLI: Evaluating Cross-lingual Sentence Representations.* Issue: arXiv:1809.05053 arXiv: 1809.05053 [cs]. Sept. 2018. DOI: `10.48550/arXiv.1809.05053`.

[40] Yinfei Yang et al. *PAWS-X: A Cross-lingual Adversarial Dataset for Paraphrase Identification.* Issue: arXiv:1908.11828 arXiv: 1908.11828 [cs]. Aug. 2019.

[41] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* arXiv:1412.6980 [cs]. Jan. 2017.

[42] Noam Shazeer and Mitchell Stern. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost*. arXiv:1804.04235 [cs, stat] version: 1. Apr. 2018.

[43] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. Issue: arXiv:1910.03771 arXiv: 1910.03771 [cs]. July 2020.

[44] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs, stat]. Dec. 2019. DOI: `10.48550/arXiv.1912.01703`.

[45] Yada Pruksachatkun et al. *jiant: A Software Toolkit for Research on General-Purpose Text Understanding Models*. Issue: arXiv:2003.02249 arXiv: 2003.02249 [cs]. May 2020.

[46] Pranav Rajpurkar, Robin Jia, and Percy Liang. *Know What You Don't Know: Unanswerable Questions for SQuAD*. arXiv:1806.03822 [cs]. June 2018.

[47] Anton Kretov. "Active Learning for NLP". In: *Czech Technical University in Prague. Computing and Information Centre.* (2023). Master thesis.

[48] Milan Straka et al. "RobeCzech: Czech RoBERTa, a monolingual contextualized language representation model". In: vol. 12848. arXiv:2105.11314 [cs]. 2021, pp. 197–209. DOI: `10.1007/978-3-030-83527-9_17`.

[49] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners*. arXiv:2109.01652 [cs]. Feb. 2022.