*mathematics*

*Article*

# Hierarchical and Bidirectional Joint Multi-Task Classifiers for Natural Language Understanding

**Xiaoyu Ji [1,2], Wanyang Hu [3,*] and Yanyan Liang [1,4,*]**

1 School of Computer Science and Engineering, Faculty of Innovation Engineering, Macau University of Science and Technology, Macau, China; 1909853xii3001@student.must.edu.mo
2 Guangxi Key Laboratory of Machine Vision and Intelligent Control, Wuzhou 543002, China
3 Faculty of Informatics, Università della Svizzera Italiana, 6962 Lugano, Switzerland
4 CEI High-Tech Research Institute Co., Ltd., Macau, China
* Correspondence: wanyang.hu@alumni.usi.ch (W.H.); yyliang@must.edu.mo (Y.L.)

**Abstract:** The MASSIVE dataset is a spoken-language comprehension resource package for slot filling, intent classification, and virtual assistant evaluation tasks. It contains multi-language utterances from human beings communicating with a virtual assistant. In this paper, we exploited the relationship between intent classification and slot filling to improve the exact match accuracy by proposing five models with hierarchical and bidirectional architectures. There are two variants for hierarchical architectures and three variants for bidirectional architectures. These are the hierarchical concatenation model, the hierarchical attention-based model, the bidirectional max-pooling model, the bidirectional LSTM model, and the bidirectional attention-based model. The results of our models showed a significant improvement in the averaged exact match accuracy. The hierarchical attention-based model improved the accuracy by 1.01 points for the full training dataset. As for the zero-shot setup, we observed that the exact match accuracy increased from 53.43 to 53.91. In this study, we observed that, for multi-task problems, utilizing the relevance between different tasks can help in improving the model's overall performance.

**Keywords:** multi-task classifier; hierarchical structure; bidirectional joint structure; MASSIVE dataset

**MSC:** 68T07; 68T50

check for updates

**Citation:** Ji, X.; Hu, W.; Liang, Y. Hierarchical and Bidirectional Joint Multi-Task Classifiers for Natural Language Understanding. *Mathematics* **2023**, *11*, 4895. https://doi.org/10.3390/math11244895

Academic Editors: Nebojsa Bacanin and Catalin Stoean

Received: 26 October 2023
Revised: 29 November 2023
Accepted: 4 December 2023
Published: 7 December 2023

## 1. Introduction

Natural language understanding (NLU) plays an important role in information-processing systems, and the most common application of NLU is in virtual assistants like Alexa, Siri, and Google Assistant. With the help of a virtual assistant, people can achieve a desired task by just saying it instead of inputting it in devices. For example, when people say "Play music relating to the music I listened to this morning", a virtual assistant will start to play the relevant music without requiring the user to press any buttons.

Furthermore, with the support of smart home devices like lights, ovens, and televisions, virtual assistants can be much more powerful and efficient in helping the disabled and the aged. People do not need to remember how to use these devices, and, furthermore, such information is somewhat complicated for the elderly. Users can tell their virtual assistant their intentions, and the virtual assistant will operate the devices as the user wishes.

To perform this function in its entirety, in the first step, the virtual assistants have to understand what people are saying, which is the prime objective in NLU. Generally, the NLU ability is measured according to the accuracy of the machine understanding the meaning of and figuring out the corresponding entities in a given text, also known as intent classification and slot filling in many papers, including this paper.

For a specific example, given the text "order me a cheese burger from tommy's burgers", an NLU model should recognize the intent as "takeaway_order" and also fill the slot as "food_type: cheese burger" and "business_name: tommy's burgers". The intent label tells the virtual assistant to make a takeaway order, and the slot labels give the details of this order, i.e., ordering a cheeseburger from a burger shop called Tommy's Burgers.

For voice-based virtual assistants, spoken language understanding (SLU) [1,2] is the foundation for completing an entire task. In SLU, the user's audio is converted into text before applying the NLU model. We skipped this part in this paper since the MASSIVE [3] dataset offers utterances from raw audio for training and evaluating the NLU model. In Section 1.1, we discuss the details of the MASSIVE dataset and explain why we chose it for the experiment.

An NLU model consists of several components for the splitting of subtasks. Firstly, a tokenizer in the NLU model maps the words in the sentence to vocabulary IDs since the model cannot deal with the text directly. By applying an embedding layer, the vocabulary ID is represented by a vector (embedding features), and synonyms will have similar embedding features.

By transforming a sentence into a sequence of embedding features, the encoder in the NLU model computes the hidden features that constitute the semantic information of the sentence. The last components of the NLU model are the classifiers for addressing the main purposes, intent detecting, and slot filling. Since predicting only the intent or slot labels will not allow a virtual assistant to perform an operation properly, we also need to evaluate the exact match accuracy, which means that the model needs to predict all kinds of labels correctly.

### *1.1. Dataset*

The MASSIVE dataset [3] is a new dataset released last year. It consists of 1M new realistic, human-created virtual assistant utterances in text form covering 51 languages from 14 language families and in 21 distinct scripts.

Compared to other popular datasets in the NLP community, the quantity of languages in MASSIVE is the largest, while there are only nine in Multi ATIS++ [4], six in MTOP [5], and three in ATIS, with Hindi and Turkish [6] and Cross-lingual Multilingual Task-Oriented Dialog [7]. As research on such a large number of languages grows, more people using different languages can benefit from the results.

The utterances employed, spanning 18 domains, the same size as the SLURP [8] and NLU Evaluation datasets [9], can be used in most aspects of our daily lives. The 60 intent data points and 55 slot types specify the key information in the utterances and allow the virtual assistant to 'understand' the next operation. By improving intent classification accuracy and slot-filling scores, more precisely, the accuracy of exact matching (both intent- and slot-matched), virtual assistants could provide better service to their users.

The MASSIVE dataset contains 587K training, 104K validation, and 152K testing samples. There are 19,521 utterances for each language, and the similar sizes of the samples can reduce the influence of the unbalanced data size.

Every piece of data is organized into the files of JSON lines. An example JSON line includes the following information:

- ID—the original ID in the SLURP [8] collection;
- Locale—the language and country code;
- Partition—this can be either train, dev, or test, according to the original split in SLURP [8];
- Scenario—the general domain of an utterance;
- Intent—the specific intent of an utterance within a domain formatted as {scenario}_{intent};
- utt—the raw utterance text without annotations;
- annot_utt—the text from utt with slot annotations formatted as [{label}:{entity}];
- worker_id—the obfuscated worker ID from the MTurk of the worker completing the localization of the utterance;

- slot_method—(optional) measurement of whether the slot usage is correct or not when the locale is not en-US;
- Judgments—(optional) scores for the expression comment assigned by native speakers when the locale is not en-US.

In our experiment, we only used 'scenario', 'intent', 'utt', 'locale', 'annot_utt', 'id', and 'partition' to perform the training set, the validation set, and the test set splitting according to the information in 'partition'.

The results are shown in Table 1 for massively multilingual NLU (MMNLU) modeling [3] as the initial model benchmarks. They choose two models, XLM-Roberta (XLM-R; [10]) and mT5 [11], with tree architectures, XLM-R Base, mT5 Base (encoder-only), and mT5 Base (text-to-text), to train on the MASSIVE dataset in two ways: one using the full training set and the other, zero-shot, trained on English data, validated in all languages, and tested on all non-English locales. We discuss the difference between setting up the model using the two training methods in Section 3. The configurations and results analyses for the two training methods are presented in Section 4.

**Table 1.** Benchmarks of MASSIVE dataset [3].

| Model | Avg Intent Acc (%) | Avg Slot F1 (%) | Avg Exact Match Acc (%) |
|---|---|---|---|
| XLM-R base | 85.1 ± 0.2 | 83.6 ± 0.2 | 75.0 ± 0.2 |
| mT5 base (encoder-only) | 86.1 ± 0.2 | 82.2 ± 0.2 | 74.7 ± 0.2 |
| mT5 base (text-to-text) | 85.3 ± 0.2 | 81.3 ± 0.2 | 73.8 ± 0.2 |
| (a) Test results when using the full training set. | | | |
| XLM-R base | 70.6 ± 0.2 | 64.2 ± 0.3 | 52.9 ± 0.3 |
| mT5 base (encoder-only) | 61.2 ± 0.2 | 56.9 ± 0.3 | 42.8 ± 0.3 |
| mT5 base (text-to-text) | 62.9 ± 0.2 | 50.6 ± 0.3 | 42.8 ± 0.3 |
| (b) Zero-shot test results after training on en-US alone. | | | |

### *1.2. Motivation*

As shown in Table 1a, in the full training set case, the gaps between the intent accuracies and slot-filling scores are 1.5 in the XLM-R base model, 3.9 in the mT5 base (encoder-only) model, and 4.0 points in the mT5 base (text-to-text) model. However, the gaps between the intent and exact match accuracies expand to 10.1, 11.4, and 11.5. We can see a similar situation in the zero-shot case, but the differences are much greater.

Usually, we set classifiers for detecting intent and filling slots and then evaluate the accuracy and scores independently. But, obviously, intent labels and slot labels are interrelated. If the intent label of the text is "transport_query", the possibility of filling the slot with "song_name" or "audiobook_author" would be extremely low.

The two tasks, namely intent detection and slot filling, are separate and relevant, giving us an idea about how to explore a way in which to increase the logical relevance between intent and slot labels. Based on this idea, we designed hierarchical and bidirectional architectures to utilize the intent and slot results for improving the model performance.

Instead of classifying intent and slots separately, in the hierarchical architecture, the model treats one task as the super-task of the other and the output of the super-task effect as the sub-task when classifying or predicting. In this case, intent classification is the super-task of slot filling, and we wish to improve the slot-filling scores for those samples with the correct intent prediction. Although the overall slot-filling scores may decline, this method could improve the exact match accuracy.

Similar to the hierarchical architecture, the bidirectional architecture also uses the results of intent classification as part of the input for predicting the slot labels. Meanwhile, the outputs of the slot classifier are utilized for updating the probabilities of intent labels.

We present several ways of utilizing the outputs, including updating the hidden features before applying the classifiers and computing attention-based transition probabilities to update the initial intent and slot logits. The related works are further described in Section 2. More details on the architecture can be found in Section 3.

For each structure, we tuned the models to search for the best results. In Section 4, we discuss how the hyper-parameters affected the results and analyze all the model performances.

### *1.3. Our Contributions*

The contributions of this study are as follows:

1. We designed hierarchical and bidirectional architectures based on the relevance of intent and slots to improve the model performance.
2. We presented two ways of utilizing the outputs, including updating the hidden features before applying classifiers and computing attention-based transition probabilities to update the initial intent and slot logits.
3. We tuned the models for the best results, discussed how the hyper-parameters affected the results, and analyzed all the models' performances.

## **2. Related Works**

With the development of virtual assistants in recent decades, more and more people are benefiting from this technology. Although every device has a multi-language system, the supporting languages are only a small part of the world's 7000+ languages [12]. Some researchers have advised exploring the unique characteristics of low-resource languages (LRLs). The authors of [13] provided 19 languages in the Research on English and Foreign Language Exploitation (REFLEX-LCTL) program as the beginner course for research on less commonly taught languages. Numerous language packages for multilingual resources have been released over the past ten years, like the Low Resource Languages for Emergent Incidents (LORELEI) program with more than 30 languages [14] and the FLORES-101 program consisting of 101 languages [15]. And, research on low-resource and multilingual areas has also seen some improvements. For example, Ref. [16] introduced a degradation test for establishing baselines for low-resource, low-data environments; Ref. [17] provided a neural machine translation (NMT) benchmark between English and five African LRL pair; and Ref. [18] summarized previous groundbreaking achievements in resolving LRL problems.

For the task of NLU, ATIS with Hindi and Turkish [6] is one of the first datasets to extend NLU tasks from a single language to multiple languages (three), and MultiATIS++ [4] increased this number to nine. But nine is not enough for the multilingual area, so we used the MASSIVE [3] dataset, which provided 51 languages for slot filling, intent classification, and virtual assistant evaluation tasks. Additionally, the two ways of setting up training models verified the performance in situations with enough samples (on the full training set) and involving low-resource environments (zero-shot).

Meanwhile, language models like mBERT [19], RoBERTa [20], XLM [21], XLM-R [10], mBART [22], MARGE [23], and MT5 [11] were pre-trained using a multilingual corpus to support the low-resource languages and induce a dramatic improvement in understanding them.

These pre-trained models are trained for general mask filling and usually serve as encoders used to search for hidden features in the hidden space. We still need to develop fine-tuned classifiers for these tasks. According to how a model learns the hidden states for intent classification and slot-filling tasks, either separately or jointly, a model can be defined as an independent model or a joint model. We focus on the joint model in this paper. The baseline is the result obtained using the XLM-R model based on JointBERT [24].

Although the encoder learned the hidden features for intent classification and slot filling jointly, the classifiers for the two tasks are independent. The authors of [25] mentioned a joint detection method for exploring the intent results to improve the slot-filling performance. The authors of [26] proposed a framework wherein the dialogue state tracking operated on a set of candidate values for each slot in the dialogue domain. The candidate values were generated via language comprehension slot annotations, dialogue acts, and the dialogue state. In [27], a routing-by-agreement schema based on the capsule neural network was introduced to synergize the slot-filling performance using the inferred intent representation. The authors of [28] presented a hierarchical multi-task structure for sequentially detecting dialogue acts, user intent, and slots. The representations of the last task are a part of the input for the next one. Following such ideas, we introduced two models based on a hierarchical architecture to explore the relationship between intent and slot labels.

In [29], the slot results were a part of the input for the intent classifier, in which the intent error declined with the assistance of slot logits. The authors of [30] introduced a bidirectional joint architecture, updating hidden features with the intent predictions for slot-filling tasks and using the same operation for intent classification. We modified the bidirectional structure in three ways to allow it to run faster and reduce the parameters.

## 3. Methodology

To achieve our goal, we chose the pre-trained XLM-R [10] Model for extracting the contextual, bidirectional hidden features of the utterances. Then, the hidden features were fed to the classifiers for the intent classification and slot-filling tasks.

We present several architectures for the classifiers in Section 3.2, attempting to increase the extra match accuracy and narrowing the gaps between intent accuracy and slot scores.

### *3.1. Pre-Trained Model*

The XLM-R model based on the Transformer model [31] followed the XLM approach [21] as closely as possible and used a masked language modeling (MLM) objective. XML-R was trained in 100 languages, and the related 'xlm-roberta-base' tokenizer was pre-trained on 2.5TB of filtered CommonCrawl data containing 100 languages, with over 25,000 words in the vocabulary.

The samples in MASSIVE were a series of user utterances with annotations created by localizing the Spoken Language Understanding Resource Package (SLURP) dataset [8]. We applied the tokenizer for mapping the words in the utterance to a well-pre-trained vocabulary and returned the word ID as the output.

When mapping the words in the sentence to a sequence of word IDs, the tokenizer also adds (CLS) and (SEP) tokens to the beginning and end of a sentence because our tasks are centered on classification. Let us suppose that the sequence length of the utterance is $l$; then, the token ID length $L = l + 2$ will be fed to the model.

The tokenizer maps nature words to its vocabulary ID, but the model still needs a way in which to convert such IDs to low-dimension word embeddings. The WordPiece embeddings [32] are a series of dense real-valued vectors representing the features of each word.

The vocabulary size is 250,002, which is a large number since the model needs to deal with 51 languages, but the hidden size of the embedding layer is not as large as the vocabulary size. We know people use different words to express similar meanings, but how does the model recognize different word IDs showing similar meanings in the way human beings would? One solution is to let the embedding layer output similar word features. So, the other role of the embedding layer in accomplishing the tasks using the MASSIVE dataset is to determine the word IDs that have the same meaning in different languages and output similar features for them.

The pre-trained model provides a pre-trained embedding layer for word feature representation and similar-word detection. When we used the full dataset for training, the backward procedure was used to update the weights of the embedding layer to achieve better performance.

On the other hand, the weights of the embedding layer were frozen in the zero-shot situation. A well-pre-trained embedding layer will output similar word features for synonyms, even in different languages. If the embedding weights were updated while training on the en-US set alone, the embedding layer might not have output similar features for synonyms.

We can set the hidden size of the embedding layer to $H_{em} = 768$. According to the token ID length $L$, the input shape of the embedding layer is $[L, 1]$, and the output shape will be $[L, H_{em}]$ after applying the embedding layer.

The encoder layer computes the contextual, bidirectional hidden features of the WordPiece embeddings from the embedding layer. The hidden features are fed to the classifiers for intent classification and slot-filling tasks.

The pre-trained model stacks 12 layers, with 12 attention heads and an $H_{ec} = 768$ hidden data size [10]. The stack layer consists of a Muti-head attention module, an intermediate module, and a layer output module. The Muti-head module uses a self-attention class with 12 attention heads and a hidden size of $H_{ec}$ to compute the contextual features. A dense, linear layer with input and output sizes of $H_{ec}$; a dropout layer with probabilities of 0.0; and a layer normalizer with a layer norm eps of $1 \times 10^{-5}$ were applied sequentially after the self-attention class was established. The components of the intermedia module are a dense layer and a GELU activation function. The layer output module connects the two slack layers to normalize the output features from the intermediate module and adjust them so that they are the input for the next slack layer. The module includes a dense layer, a dropout layer, and layer normalization with the same configuration as the Muti-head attention module.

The input of the encoder layer is a sequence of word features with the following shape: $[L, H_{em}]$. Suppose that the encoder is simple and easy to handle; we set the output shape of every slack layer to $[L, H_{ec}]$, so the final stack layer outputs the hidden features with the following shape: $[L, H_{ec}]$.

## 3.2. Classification Layer

Essentially, the classification layer for the tasks involves two separate classifiers, and each classifier outputs the possibilities of the labels. For the baseline [3], the classification layer based on JointBERT [24], the sequential connotation of the intent classifier is the max-pooling layer, a dropout layer with a dropout rate of $P_{ff} = 0.45$, a linear layer with $D_c = 2048$ dimensions, the GELU active function, another dropout layer with the same dropout rate, and a final linear layer with $D_{IP} = 60$ dimensions (the number of intent labels). The difference regarding the slot classifier is that the max-pooling layer is not used, and the dimensions of the final linear layer are $D_{SP} = 56$ (the number of slot labels) instead. In the zero-shot case, the feed-forward dropout rate $P_{ff}$ was set to 0.25, and the number of linear dimensions $D_c$ was 8192.

The intent label is unique for each sentence, so the output shape of the intent classier should be $[D_{IP}, 1]$. In consideration of the output shape, adding a max-pooling layer in the intent classifier could reduce the number of input feature dimensions $H_{ec}$ to 1. On the other hand, the slot-filling task computes the slot possibility of each word with an output shape $[L, D_{SP}]$, which is why the pooling reduction is not necessary for the slot-filling layer.

Because the characteristic of the filling of the slot labels is associated with the intent labels, we modified the classifiers in two ways, introducing a hierarchical classification layer and a bidirectional classification layer in Sections 3.2.1 and 3.2.2.

### 3.2.1. Hierarchical Classifiers

When given an intent label, the probabilities for some slot labels in a sentence would be pretty high, and some might be very low. For instance, if the utterance intention is "transport_taxi", slots labels like "transport_agency", "currency_name", "date", "place_name", and "transport_descriptor" would more likely be filled with some words in the sentence, while filling "player_setting", "music_album", "music_descriptor", and "audiobook_author" is almost impossible.

Generally, detecting the intent of an utterance is much easier than filling all the slots correctly. This situation gave us the idea that intent probabilities could be used to compute the likelihood of slot labels. Based on this idea, we present the classification layer with a hierarchical architecture in this section.

As shown in Figures 1 and 2 show, the hidden features from the encoder with the shape $[L, H_{ec}]$ were fed to the intent classifier, denoted by $[l_1, l_2, \cdots, l_{H_{ec}}]$, in which $l_i \in \mathbb{R}^L$. Max pooling reduces the shape of the hidden features to $[H_{ec}, 1]$ by maintaining the maximum in each $l_i$. Let the output of max pooling be $Inf \in \mathbb{R}^{H_{ec}}$; thus, we have each element $Inf_i$ in $Inf$:

$$Inf_i = max(l_i) \tag{1}$$

By feeding the max-pooling output to the other layers, the classifier can compute the intent probabilities $P_{IP} \in \mathbb{R}^{D_{IP}}$ by applying other dropout and linear layers. The output $P_{IP}$ of the final linear layer is provided by the following equation:

$$P_{IP} = dropout(Inf) \cdot W_I^T + b_I \tag{2}$$

The final linear layer of the intent classifier condenses the features to $D_{IP}$ heads, and the dimensions of the dropout layer output are correspond to $D_c$. Therefore, $droupout(Inf) \in \mathbb{R}^{D_c}$ represents the output of the last dropout layer, $W_I \in \mathbb{R}^{D_{IP} \times D_c}$ denotes the weights of the linear layer, and $b_I \in \mathbb{R}^{D_{IP}}$ denotes the biases. Here, we have the output of the intent classifiers $P_{IP}$ with a shape of $[1, D_{IP}]$.

In the next step, $P_{IP}$ will update the hidden features before applying slot clarification. There are more than two ways to update the hidden features, but we explored two in this paper:

(1) The first is concatenation, as shown in Figure 1. The intent logits will concatenate with the output from the encoder, the simplest method of updating;
(2) The second is attention-based computation, updating the initial slot logits via computing the intent and slot attention probabilities, which is another effective and smooth way of updating the hidden features. The architecture is shown in Figure 2.

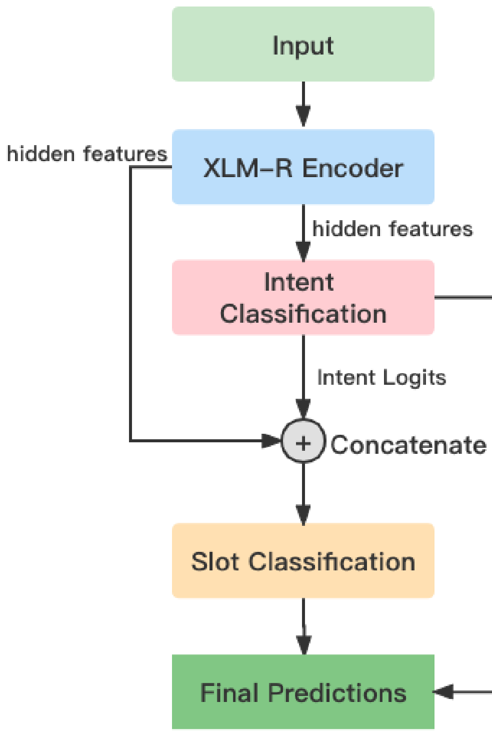

**Figure 1.** The architecture of the model with a hierarchical concatenation classification layer.

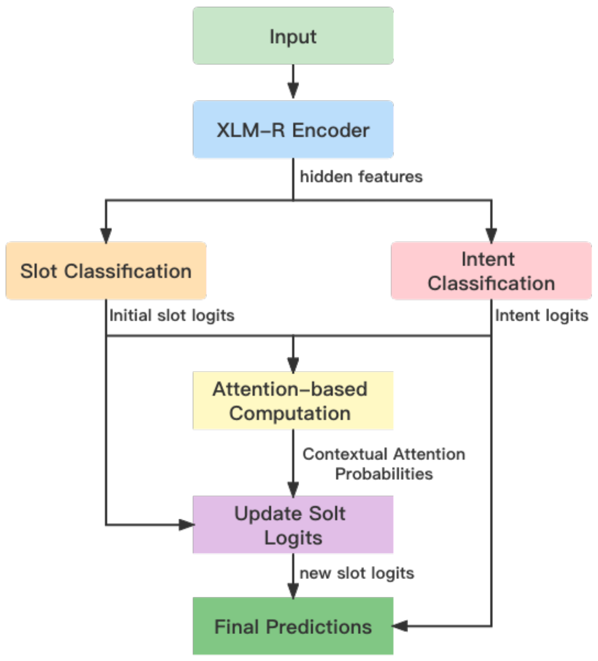

**Figure 2.** The architecture of the model with a hierarchical attention-based classification layer.

**Hier-concatenation:** We have the output of the encoder layer, the hidden features $H \in \mathbb{R}^{\mathbb{L}\times\mathbb{H}}$, the output of the intent classifier, and the intent logits $P_{IP} \in \mathbb{R}^{D_{IP}}$, but the shapes of the two matrices are not yet matched for concatenation. Let $H = (h_1, h_2, \cdots, h_L)$ denote the hidden features, while $h_l$ represents a feature vector for each word.

As the basic idea behind the relationship between intent and slots, intent logits will affect every word's slot-filling task. Next, we expand $P_{IP}$ to a sequence of expanded features with a length of $L$:

$$\hat{P}_{IP} = (\hat{p}_1, \hat{p}_2, \cdots, \hat{p}_L) \tag{3}$$

where $\hat{p}_l = P_{IP}$, for $l = 1, 2, \cdots, L$.

Now, $H \in \mathbb{R}^{L\times H_{ec}}$ and $\hat{P}_{IP} \in \mathbb{R}^{L\times D_{IP}}$ can be concatenated as a new sequence of hidden features, leading to Formula (4), which will serve as the input of the slot classifier:

$$H_{cat} = H \oplus \hat{P}_{IP} \tag{4}$$

where $H_{cat} \in \mathbb{R}^{L\times(H_{ec}+D_{IP})}$.

The output of the first linear layer in the slot classifier is as follows:

$$SL_1 = dropout(H_{cat}) \cdot W_{S1}^T + b_{S1} \tag{5}$$

Before the linear layer is applied, a dropout layer randomly sets some values to 0 with the possibility $P_{ff}$ and scales others by the vector of $\frac{1}{1-p_{ff}}$. The shape of the dropout output is the same as $H_{cat}$, and we use $D_c$ to represent the output dimension of the linear layer; thus, $W_{S1}^T \in \mathbb{R}^{L\times D_c\times(H_{ec}+D_{IP})}$ and $b_{S1} \in \mathbb{R}^{L\times D_c}$. So, the output of the first linear layer of the slot classifier is $SL_1 \in \mathbb{R}^{L\times D_c}$.

We set the $D_{SP}$ head for the final layer; thus, the slot classifier finally outputs a matrix $P_{SP} \in \mathbb{R}^{L\times D_{SP}}$ for the prediction and loss computation.

**Hier-attention-based computation:** Updating the slot logits by computing the attention-based scores requires more operations than pure concatenation. We established a sequential network to compute the likelihood of the slot labels in given intent logits. The sequence of the network is as follows:

- A dropout layer: dropout probability = $P_{ff}$;
- A linear layer: input dimensions $= D_{IP}$ and output dimensions $= D_C$;

- An activation function: GELU;
- A dropout layer: dropout probability = $P_{ff}$;
- A linear layer: input dimensions $= D_C$ and output dimensions $= D_{SP}$;
- A layer normalization procedure: eps = $1 \times 10^{-5}$.

Let the function *Transit* denote the computations in the network above, to which the broadcasted intent logits $\hat{P}_{IP}$ (as Equation (3)) are fed; thus, we have

$$P_{I2S} = Transit(\hat{P}_{IP}) \tag{6}$$

Here, $\hat{P}_{IP} \in \mathbb{R}^{L \times D_{IP}}$ is the same shape as in the concatenation. Since the output dimension of the last linear is $D_{SP}$, we have $P_{I2S} \in \mathbb{R}^{L \times D_{SP}}$, which is the same size as the initial slot logits $P_{SP}$. In the next step, compute the dot-product of $P_{I2S}$ and $P_{SP}$, scale it by a factor of $\frac{1}{\sqrt{D_{SP}}}$; then, apply the softmax function to obtain attention probabilities $P_{atte} \in \mathbb{R}^{L \times L}$:

$$P_{atte} = softmax(\frac{P_{SP} \cdot P_{I2S}^T}{\sqrt{D_{SP}}}) \tag{7}$$

There are two popular ways of using attention probabilities: additive attention [33] and dot-product (multiplicative) attention. We used dot-product attention in this case because it is faster and more space-efficient in practice [31]. Thus, we attained the contextual likelihood of slot labels $P_{SC}$:

$$P_{SC} = P_{atte} \cdot P_{SP} \tag{8}$$

Furthermore, we wished to update the slot logits smoothly, applying the $softmax$ function to $P_{SC}$. Finally, the new slot logits $P_{SP}^*$ is provided by the following equation:

$$P_{SP}^* = P_{SP} + softmax(P_{SC}) \tag{9}$$

### 3.2.2. Bidirectional Classifiers

Not only will the intent labels affect the filling slots, but the slot results could revise the wrong intent of the utterance if the model has correctly predicted all the slots. A bidirectional classification layer will use both intent and slot logits to update the hidden features.

From intent logits to slot logits, the way in which to update the hidden features is the same as that used for hierarchical classifiers. In the other direction, let the slot logits update the intent features after the max-pooling layer; we can use bi-concatenation (Figure 3) and bi-attention-based computation structures (Figure 4) to achieve this goal. But, we need to slightly modify the algorithm.

**Bi-concatenation:** The second dimension of the slot probabilities $P_{SP}$ is $L$, and that of intent features $Inf_i$ is 1, thereby obstructing the possibility of direct concatenation in order to construct new intent features. We introduce two ways of extracting the information in $P_{SP}$ before concatenating it with $Inf_i$:

(1) Apply the max pooling layer to slot probabilities $P_{SP}$;
(2) Apply the LSTM layer and input $P_{SP}$ as a sequence of word slot probability vectors with time steps, and the output of the final LSTM time step will consist of all slot information.

**Max pooling:** The shape of slot probabilities $P_{SP}$ is $[L, D_{SP}]$, and the shape of intent features $Inf$ is $[H_{ec}, 1]$. The max pooling layer will reduce matrix $P_{SP}$ to a vector $P_{SPV}$. Usually, when we utilize the slot results to update intent features, all the values in $P_{SP}$ will concatenate to $Inf$ via reshaping into a vector with a shape of $[L \times D_{SP}, 1]$ [30]. But, we know that the lengths of the sentence $L$ are different in practice; thus, the uncertain $L$ leads to an indeterminate shape of $P_{SPV}$ and the new intent feature $Inf_{new}$. This uncertainty makes the initialization network difficult to work with when training or evaluating the data.

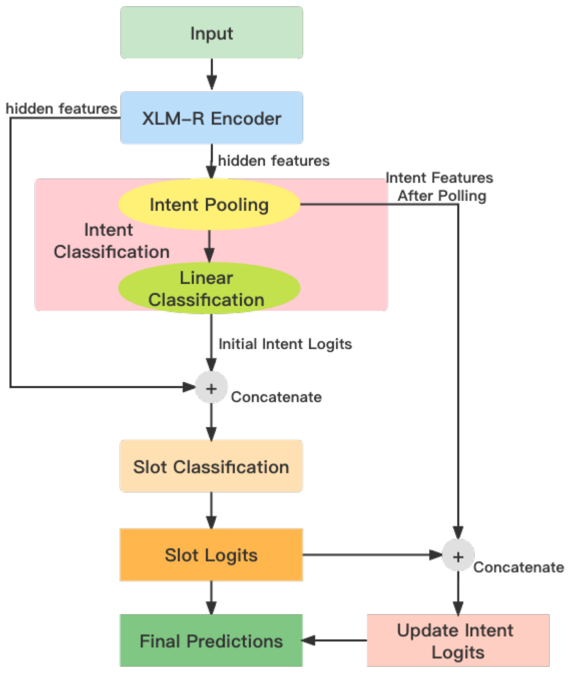

**Figure 3.** The architecture of the model with a bidirectional concatenation classification layer.

One solution to solving the uncertainty problem is placing a sequence of zeros after a sentence to guarantee that all the sentences have the same length $L$ and that $L$ is the maximum sequence length. With the help of the attention mask, the model can ignore irrelevant information in the prediction phase. But this solution requires a large amount of memory from the GPU and a long time for computation.

We used a max pooling layer to zip the sequence of slot logits in a vector with a shape of $[D_{SP}, 1]$. We present $P_{SP}$ in the column vector as $P_{SP} = (p_{SP_1}, p_{SP_2}, \cdots, p_{SP_{D_{SP}}}), p_{SP_i} \in \mathbb{R}^L$, for $i = 1, 2, \cdots, D_{SP}$, so the new vector $P_{SPV} = (p_{SPV_1}, p_{SPV_2}, \cdots, p_{SPV_{D_{SP}}}), p_{SPV_i} \in \mathbb{R}$, for $i = 1, 2, \cdots, D_{SP}$ was given as follows:

$$p_{SPV_i} = max(p_{SP_i}) \tag{10}$$

In the next step, $P_{SPV}$ is concatenated with $Inf$, we have new intent features $Inf_{new} \in \mathbb{R}^{H_{ec}+D_{SP}}$, and $Inf_{new}$ will be used to update the intent logits.

The mechanism of max pooling consists of retaining the max value of each slot label regardless of the position and the frequency. If the sentence contains a slot label, this slot logit would be higher than the labels not contained. High logits will have more impactful effects on updating intent logits.

**LSTM:** The LSTM network is a variant of an RNN, where the output from the last time step is input into the next time step. By setting input, forget, cell, and output gates, the LSTM network can retain long-term memories from the beginning of the sequence. Thus, the final time step output of the LSTM can be regarded as a summary of the whole utterance and yields the probability distribution $p^*_{LSP}$ of the slot labels.

Let the $LSTM$ function consist of all the gate computations; the result of the $LSTM$ function is the output of an LSTM cell. When we input $P_{SP} = p_{1SP}, p_{2SP}, \cdots, p_{LSP}$ row by row in sequence, we obtain

$$p^*_{iSP} = LSTM(p_{iSP}, p^*_{(i-1)SP}) \tag{11}$$

The final output of the LSTM network is

$$p^*_{LSP} = LSTM(p_{LSP}, p^*_{(L-1)SP}) \tag{12}$$

$Inf_{new}$, in this case, is yielded by the following equation:

$$Inf_{new} = Inf \oplus p^{*}_{LSP} \tag{13}$$

To date, we have $Inf_{new} \in \mathbb{R}^{H_{ec}+D_{SP}}$ to compute new intent logits. Other neural networks similar to the intent classifier were applied for this objective. The neural networks were stacked in sequence as follows:

- A dropout layer—dropout probability = $P_{ff}$;
- A linear layer—input dimensions $= (H_{ec} + D_{SP})$; output dimensions $= D_C$;
- An activation function—GELU;
- A dropout layer—dropout probability = $P_{ff}$;
- A linear layer—input dimensions $= D_C$; output dimensions $= D_{IP}$.

**Bi-attention-based computation:** As Figure 4 shows, firstly, the model computes the initial intent logits $P_{IP}$ and slot logits $P_{SP}$ using the classifiers. Secondly, $P_{IP}$ is used to compute the contextual slot probabilities, as in the hier-attention architecture, and update $P_{SP}$ to obtain new slot logits $P^{*}_{SP} \in (R)^{L \times D_{SP}}$.

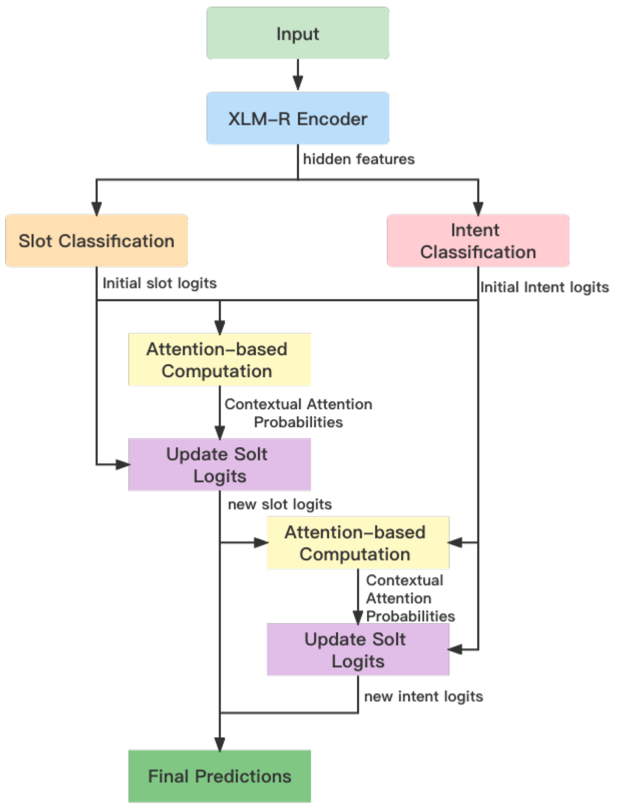

**Figure 4.** The architecture of the model with a bidirectional attention-based classification layer.

We need to reduce the matrix $P^{*}_{SP}$ to a vector $P^{*}_{SPV}$ (like in Equation (10)) with a shape of $D_{SP}$ for the slot for intent attention-based computation. Here, we let the max logit value denote the likelihood of the slot label being filled in the sentence, regardless of the position and quantity of the slot label, like the max pooling step in bi-concatenation.

Then, $P^{*}_{SPV}$ is input into the transition function *Transit* to compute the transition probabilities for slots transitioning into intent:

$$P_{S2I} = Transit(P^{*}_{SPV}) \tag{14}$$

Let the dimensions of the linear layer adjust to the data size; we set the input dimensions of the first linear layer to $D_{SP}$, and the output dimensions of the final linear layer are $D_{IP}$.

Finally, by summing the initial intent logits $P_{IP}$ and the contextual likelihood of the slot-to-intent transition labels $P_{IC}$, we attain new intent logits $P^*_{IP}$:

$$P^*_{IP} = P_{IP} + softmax(P_{IC}) \tag{15}$$

where

$$P_{IC} = softmax(\frac{P_{IP} \cdot P^T_{S2I}}{\sqrt{D_{IP}}}) \cdot P_{IP} \tag{16}$$

### 3.3. Evaluation

When FitzGerald et al. posted the benchmark of the MASSIVE dataset, they also provided the evaluation equations in their paper [3], and we followed their algorithm in our experiment. If the number of correct samples in intent prediction in $N_{IC}$ and the number of samples are both correct in intent prediction, and all the slot-filling is $N_{EM}$, the number of total samples is $N$, for intent accuracy ($ACC_{IT}$) and the exact match accuracy ($ACC_{EM}$), they were computed by the equation:

$$ACC_{IT} = \frac{N_{IC}}{N} \tag{17}$$

$$ACC_{EM} = \frac{N_{EM}}{N} \tag{18}$$

As for the slot-filling F1 score, we compute the micro-average for all the samples. First, we need to know the total *precision* and *recall*, let $TP$ be the quantity of true positive samples in all the label categories, $FP$ for the false positives, and $FN$ for the false negatives, and we have:

$$precision = \frac{TP}{FP + TP} \tag{19}$$

$$recall = \frac{TP}{TP + FN} \tag{20}$$

So the F1-score for the slot-filling is:

$$Score_{SF} = \frac{2 * precision \cdot recall}{precision + recall} \tag{21}$$

Assuming a normal distribution for all samples, the standard deviation of intent accuracy $STD_{IT}$, slot F1 scores $STD_{SF}$, and exact match accuracy $STD_{EM}$ were given by:

$$STD_{IT} = \frac{\sqrt{ACC_{IT} * (1 - ACC_{IT})}}{N} \tag{22}$$

$$STD_{SF} = \frac{\sqrt{Score_{SF} * (1 - Score_{SF})}}{N} \tag{23}$$

$$STD_{EM} = \frac{\sqrt{ACC_{EM} * (1 - ACC_{EM})}}{N} \tag{24}$$

## 4. Experiments

We trained the model on the MASSIVE dataset in two ways. One involved training it on the full training dataset, and the other was zero-shot, training with the en-US set only, validating using all locales, and testing using all languages but en-US. We used different hyper-parameters in the two ways of training.

Based on the characteristics of the architectures in Section 3, the same configuration may not fit all the architectures. It is also necessary to tune the hyper-parameters to see the best results of the models.

### *4.1. Settings*

In this section, we introduce the hyper-parameters we used in the pre-trained model, the details of the configurations for two training approaches, and the architectures. In most cases, tuning the model to find the proper hyper-parameters was an efficient way of attaining good results.

#### 4.1.1. Encoder Configuration

There are several hyper-parameters for the encoder part, following the choice made in FitzGerald's paper [3]; the configurations for the full training set and the zero-shot set are provided in Table 2. The encoder settings for the two situations are similar, except for the attention dropout probabilities (attention_probs_dropout_prob) and freeze_layers.

**Table 2.** The chosen encoder hyper-parameter for training on (a) the full training set; and (b) the zero-shot set.

| | **(a) Full Training Set** | **(b) Zero-Shot** |
|---|---|---|
| attention_probs_dropout_prob | 0.0 | 0.35 |
| bos_token_id | 0 | 0 |
| eos_token_id | 2 | 2 |
| hidden_act | gelu | gelu |
| hidden_size | 768 | 768 |
| initializer_range | 0.02 | 0.02 |
| intermediate_size | 3072 | 3072 |
| layer_norm_eps | $1 \times 10^{-5}$ | $1 \times 10^{-5}$ |
| max_position_embeddings | 514 | 514 |
| num_attention_heads | 12 | 12 |
| num_hidden_layers | 12 | 12 |
| output_past | true | true |
| pad_token_id | 1 | 1 |
| type_vocab_size | 1 | 1 |
| vocab_size | 250,002 | 250,002 |
| use_crf | false | false |
| freeze_layers | - | xlmr.embeddings. word_embeddings.weight |

The attention dropout probability for the full training set was 0, and it was 0.35 for the zero-shot set. This means one kept all the hidden features in every attention layer, and the other randomly dropped some hidden features for the purposes of regularization and preventing the co-adaptation of neurons. As we know, if the size of the training set is small and the training times increase, the model may suffer from overfitting. So, when our model was trained on the zero-shot set, we set dropout probabilities for the attention layer in the encoder network.

Freezing the weights of the embedding layer was carried out to retain the initial word embedding output from the pre-trained XLM-R model and let the synonyms in different languages maintain similar word embedding features. If the embedding weights were updated in training, then only the weights related to English would be changed, while other languages would not update at the same time. So, we froze the weights of the embedding layer, and then the encoder layer was allowed to deal with the synonyms in different languages that were never included in the zero-shot set.

#### 4.1.2. Classification Layer Configuration

The hyper-parameters for the classification layer are more diverse than those of the encoder since we presented two architectures for the classification layer, and each architecture has at least two versions. We added three additional parameters to fit the modification for the architectures: head_slot_pooling, update_way, and layer_norm_eps. The parameter head_slot_pooling was only used in the bi-concatenation architecture to distinguish the slot-pooling methods in the classifier. layer_norm_eps was a necessary parameter for the LayerNorm layer in the attention-based computation. To keep matters easy and simple, we chose the same value $1 \times 10^{-5}$ as that selected for the encoder layer. There are two ways to update the slot logits and intent logits, so we set the parameter update_way to specify the updating method, while the string "cat" denotes concatenation, and "att" means attention-based computation. The details can be observed in Table 3.

**Table 3.** Classification layer settings.

| | **Hier-Architecture** | | | **Bi-Architecture** | |
|---|---|---|---|---|---|
| | **Concat** | **Attention** | **Max Concat** | **LSTM Concat** | **Attention** |
| slot_loss_coef | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 |
| hidde_layer_for_class | 11 | 11 | 11 | 11 | 11 |
| head_num_layers | 1 | 1 | 1 | 1 | 1 |
| head_layer_dim | 2048 | 2048 | 2048 | 2048 | 2048 |
| head_activation | elu | gelu | elu | elu | gelu |
| hidden_dropout_prob | 0.45 | 0.45 | 0.45 | 0.45 | 0.45 |
| head_intent_pooling | max | max | max | max | max |
| head_slot_pooling | - | - | max | LSTM | - |
| layer_norm_eps | - | $1 \times 10^{-5}$ | - | - | $1 \times 10^{-5}$ |
| update_way | cat | att | cat | cat | att |
| a. Chosen parameters for classification layer training on the full training set. | | | | | |
| slot_loss_coef | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 |
| hidde_layer_for_class | 10 | 10 | 10 | 10 | 10 |
| head_num_layers | 2 | 2 | 2 | 2 | 2 |
| head_layer_dim | 8192 | 8192 | 8192 | 8192 | 8192 |
| head_activation | gelu | gelu | gelu | gelu | gelu |
| hidden_dropout_prob | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |
| head_intent_pooling | max | max | max | max | max |
| head_slot_pooling | - | - | max | LSTM | - |
| layer_norm_eps | - | $1 \times 10^{-5}$ | - | - | $1 \times 10^{-5}$ |
| update_way | cat | att | cat | cat | att |
| b. Chosen parameters for classification layer training on zero-shot set. | | | | | |

This model needs to accomplish two tasks simultaneously. When we designed the loss function for the combining tasks, to keep matters simple and clear, we computed the cross-entropy loss for each task in parallel and summed the intent-detecting loss and slot-filling loss with a coefficient as the overall loss for the model.

The loss is

$$Loss = mean(Loss_I) + c_{sc} \cdot mean(Loss_s) \tag{25}$$

where $Loss_I$ and $Loss_S$ are expressed as follows:

$$Loss_I = CrossEntropy(P_{IP}, T_I) \tag{26}$$

$$Loss_S = CrossEntropy(P_{SP}, T_S) \tag{27}$$

$T_I$ and $T_S$ in (26) and (27) denote the target labels of intent and slots, so $Loss$ is the summation of the averaged intent loss $Loss_I$ and the product of the coefficient slot_loss_coef times the averaged slot loss $Loss_S$. A larger slot_loss_coef means that $Loss_S$ will decline faster than $Loss_I$ when the optimizer minimizes the overall loss ($Loss$). Moreover, slot_loss_coef measures the dynamic balance of the intent accuracy and slot-filling scores, so the larger the value of slot_loss_coef, the larger the effect of $Loss_S$, and the slot-filling scores might be higher to some extent (intuitively).

By setting different values for slot_loss_coef in the full training and zero-shot sets, we expected the model to attain the best performance in exact match accuracy, and we set the searching space to [0.5, 1.0, 2.0, 4.0, 8.0, 16.0]. Regarding the size of the training samples, the slot_loss_coef in the full training set is 4.0, and this value is 2.0 in the zero-shot set. In the full training case, the large number of samples for every language allowed the model to learn the hidden features sufficiently. Meanwhile, in the zero-shot setup, only the en-US training set was insufficient for the other 50 languages, causing the slot-filling task to be much harder than intent classification. Hence, the slot_loss_coef in zero-shot is smaller than that in full training.

The output of the encoder consists of four matrices, namely attentions, sequence_ output, pooled_output, and hidden_states; thus, the parameter hidde_layer_for_class was utilized to configure which matrices we needed to feed to the classifier. Setting hidde_layer_for_class = 11 in the full-training case, we chose the sequence_output matrix for the classification when trained on the full dataset. On the other hand, zero-shot can select a required part in the hidden_states matrix by setting it to 10.

Due to the characteristics of the zero-shot setup, we set more layers and dimensions for the classifiers to learn more features from such a small number of training samples. Thus, we set head_num_layers = 2 and head_layers_dim = 8192 in zero-shot from the searching spaces [1, 2, 3, 4] and [512, 1024, 2048, 3072, 4096, 8192, 16,384], while these values are 1 and 2048 in full training with searching spaces of [1, 2, 3, 4] and [512, 1024, 2048, 3072, 4096].

The other parameter settings are hidden_dropout_prob and head_activation. Their choices depend on their performance when training in the hyper-parameter search space [0.0:0.5] with a step of 0.05 and [gelu, elu, tanh, reLU].

### 4.1.3. Training Settings

We tuned the model using a single Nvidia V100 GPU with 32 G of memory, and the training time was less than 36 h. We built the hyper-parameter-searching space following the Tree of Parzen Estimators algorithm and the asynchronous successive halving algorithm (ASHA) [34] and trialed the performance of each model. The searching space and the choices for training parameters are shown in Table 4.

**Table 4.** The training settings for training on (a) full training set and (b) zero-shot.

| | **(a) Full Training Set** | **(b) Zero-Shot** |
|---|---|---|
| eval_strategy | [steps, epoch]<br>choice, steps | [steps, epoch]<br>choice, epoch |
| eval_steps | [500, 1000, 2000, 4000]<br>choice, 1000 | - |
| learning_rate | $[1 \times 10^{-7}, 0.0001, 1 \times 10^{-7}]$<br>qloguniform, $7 \times 10^{-6}$ | $[1 \times 10^{-7}, 0.0001, 1 \times 10^{-7}]$<br>qloguniform, $4.7 \times 10^{-6}$ |
| lr_scheduler_type | [linear, constant_with_warmup]<br>choice, constant_with_warmup | [linear, constant_with_warmup]<br>choice, constant_with_warmup |
| warmup_steps | [0, 10,000, 1000]<br>quniform, 5000 | [0, 1000, 100]<br>quniform, 500 |
| train_batch_size | [32, 64, 128]<br>choice, 128 | [32, 64, 128]<br>choice, 128 |
| eval_batch_size | [32, 64, 128]<br>choice, 128 | [32, 64, 128]<br>choice, 128 |
| num_train_epochs | [20, 100, 10]<br>quniform, 50 | [20, 200, 10]<br>quniform, 150 |

Our hierarchical and bidirectional classification layers update the possibilities of intent and slots smoothly, especially in the full training case. The learning_rate was lower than the reference value in the origin baseline [3], and the evaluation frequency was also higher (every 1000 steps, executed by setting evaluation_strategy = steps and eval_step = 1000). For the zero-shot setup, the learning_rate is the same as the reference, but the frequency was increased by setting the evaluation_strategy = epoch since there are 90 steps in one epoch.

Because of the small learning rate we used, the model needs more epochs to reach the global minimum, which is the best exact match accuracy in this case. We set 50 epochs for the full training set and 150 for zero-shot. Generally, the more samples trained in one step, the better the performance, but the limited GPU memory size only allowed for a maximum batch size of 128, so we used this size in our model. The warmup_steps value was set to 5000 in the full training for a similar reason, for this number covered all the samples in one epoch (4880 steps). We retained 500 warm-up steps in the zero-shot case since it covered more than five epochs, which is large enough for a warm-up.

### *4.2. Results*

We ran all the models several times and recorded the best performance by selecting the best average exact match accuracy evaluation in all locales. Because of the different settings we chose in the experiment, we also reran the baseline for better comparison. In order to make the results clear and comparable, we counted the number of languages in different percentage ranges, as shown in Figures 5 and 6. The summary results for intent accuracy, micro-averaged slot F1 score, and exact match accuracy in the best locales, worst locales, and averaged locales are given in Tables 5 and 6.

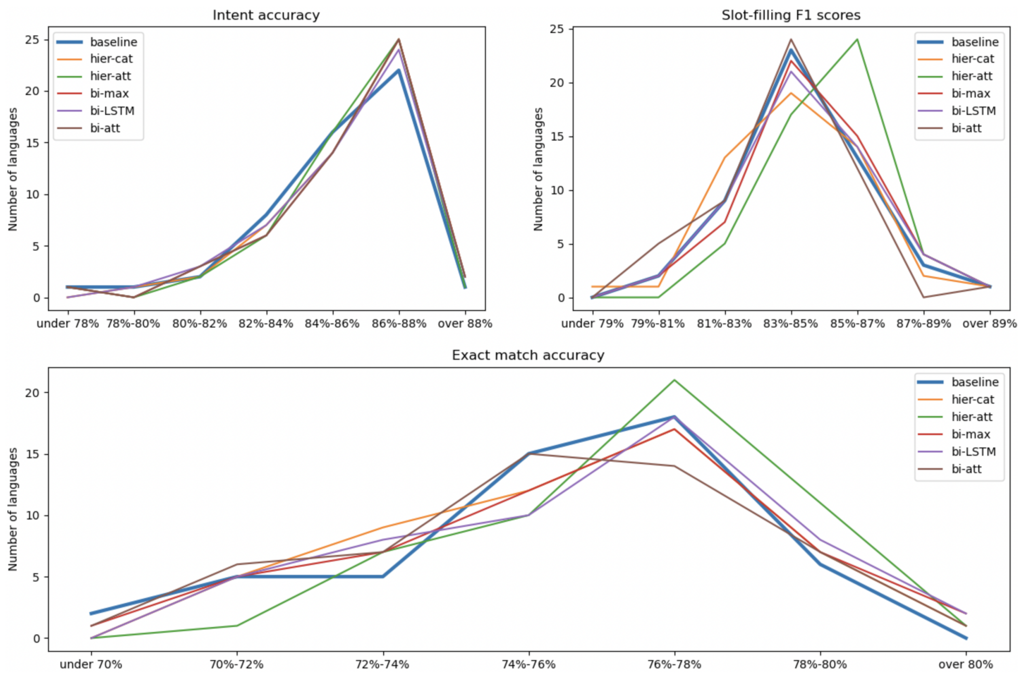

**Figure 5.** The counted number of languages in the percentage ranges for full-training.

In Figure 5, for the full training test, the green line (xmlr-hier-att) shows a significant increase in the high percentage part in slot-filling and exact-matching tasks; meanwhile, this model slightly raised the number of languages with a high intent accuracy. Overall, the model shows fewer languages in the low-accuracy part in the intent-detecting and exact-matching sections.

The situation in the zero-shot test was quite different (Figure 6): all the lines were close to each other, especially for intent accuracy. We can see a bit of a difference in the slot F1 scores: the quantity of languages with low scores is lower or the same as the baseline, and they also have a better performance in the high-score part. As for the exact match accuracy, our models did well in the low-accuracy part, as the number of languages is obviously smaller than the baseline, while the results for the high-accuracy part were not as good as we expected.

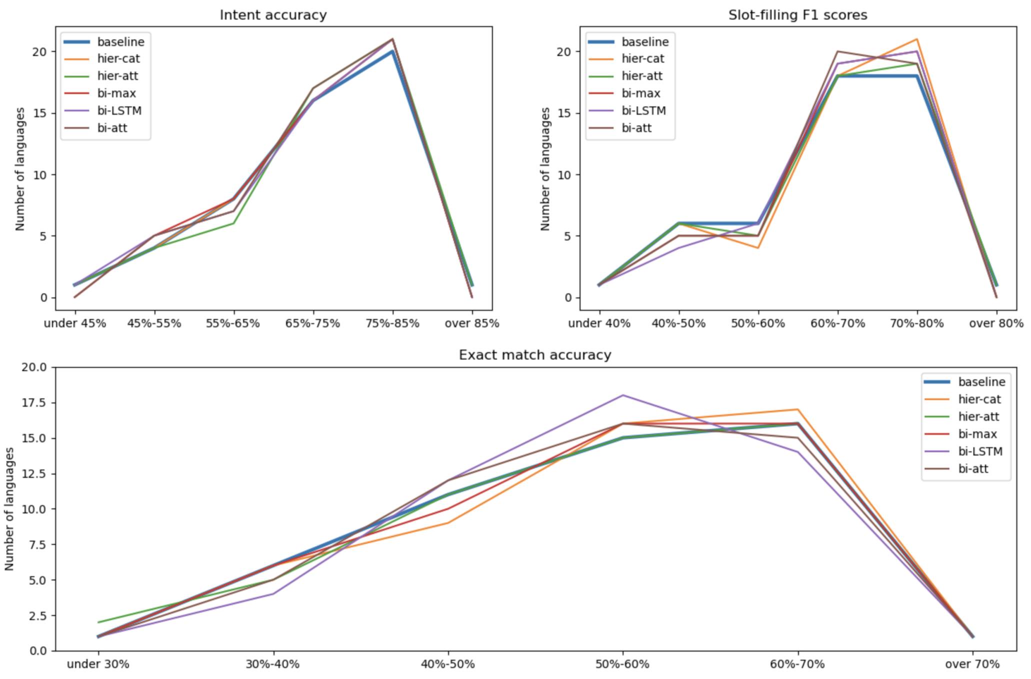

**Figure 6.** The counted number of languages in the percentage range for zero-shot.

**Table 5.** The results for training on the full training dataset, including the locale-averaged and the best and worst locale results for intent accuracy, micro-averaged slot F1 score, and exact match accuracy.

| **Model** | **Avg Intent Acc (%)** | **Avg Slot F1 (%)** | **Avg Exact Match Acc (%)** |
|---|---|---|---|
| xlmr(baseline) | 85.19 ± 0.09 | 84.30 ± 0.10 | 75.38 ± 0.11 |
| xlmr-hier-concat | **85.51 ± 0.09** | 85.04 ± 0.09 | 75.54 ± 0.11 |
| **xlmr-hier-att** | 85.48 ± 0.09 | **85.04 ± 0.09** | **76.39 ± 0.11** |
| xlmr-bi-max | 85.40 ± 0.09 | 84.42 ± 0.10 | 75.70 ± 0.11 |
| xlmr-bi-lstm | 85.48 ± 0.09 | 84.47 ± 0.10 | 75.81 ± 0.11 |
| xlmr-bi-att | 85.46 ± 0.09 | 83.89 ± 0.10 | 75.41 ± 0.11 |

a. The locale-averaged results

| Model | High Intent Acc (%) | High Slot F1 (%) | High Exact Match Acc (%) |
|---|---|---|---|
| xlmr(baseline) | 88.70 ± 0.58 en-US | 89.56 ± 0.59 th-TH | 79.96 ± 0.73 en-US |
| xlmr-hier-concat | 88.47 ± 0.58 en-US | 89.38 ± 0.59 th-TH | 80.26 ± 0.73 en-US |
| xlmr-hier-att | 88.10 ± 0.59 en-US | **89.97 ± 0.58** th-TH | 80.03 ± 0.73 en-US |
| xlmr-bi-max | **89.90 ± 0.58** en-US | 89.16 ± 0.60 th-TH | 80.33 ± 0.73 en-US |
| xlmr-bi-lstm | 88.84 ± 0.58 en-US | 89.53 ± 0.59 th-TH | 80.26 ± 0.73 en-US |
| **xlmr-bi-att** | 88.70 ± 0.58 sv-SE | 89.12 ± 0.60 th-TH | **80.50 ± 0.73** sv-SE |

b. The best locale results

| Model | Low Intent Acc (%) | Low Slot F1 (%) | Low Exact Match Acc (%) |
|---|---|---|---|
| xlmr(baseline) | 76.56 ± 0.59 km-KH | 79.38 ± 0.61 ja-JP | 69.20 ± 0.85 km-KH |
| xlmr-hier-concat | **78.14 ± 0.76** km-KH | 78.99 ± 0.77 ja-JP | 70.38 ± 0.84 ja-JP |
| **xlmr-hier-att** | 77.77 ± 0.76 km-KH | **81.07 ± 0.74** ja-JP | **71.12 ± 0.83** km-KH |
| xlmr-bi-max | 77.44 ± 0.77 km-KH | 79.71 ± 0.76 ja-JP | 69.84 ± 0.84 km-KH |
| xlmr-bi-lstm | 78.08 ± 0.76 km-KH | 79.92 ± 0.76 ja-JP | 70.51 ± 0.84 km-KH |
| xlmr-bi-att | 77.00 ± 0.77 km-KH | 79.46 ± 0.76 ja-JP | 69.47 ± 0.84 km-KH |

c. The worst locale results

**Table 6.** The results for training on the zero-shot, including the locale-averaged and the best and worst locale results for intent accuracy, micro-averaged slot F1 score, and exact match accuracy.

| **Model** | **Avg Intent Acc (%)** | **Avg Slot F1 (%)** | **Avg Exact Match Acc (%)** |
|---|---|---|---|
| xlmr(baseline) | 70.80 ± 0.12 | 64.82 ± 0.13 | 53.43 ± 0.13 |
| **xlmr-hier-concat** | 70.50 ± 0. 12 | **65.63 ± 0.13** | **53.91 ± 0.13** |
| xlmr-hier-att | 70.69 ± 0.12 | 64.84 ± 0.13 | 53.34 ± 0.13 |
| xlmr-bi-max | 70.83 ± 0.12 | 65.58 ± 0.13 | 53.62 ± 0.13 |
| xlmr-bi-lstm | **70.86 ± 0.12** | 65.57 ± 0.13 | 53.59 ± 0.13 |
| xlmr-bi-att | 70.83 ± 0.12 | 65.24 ± 0.13 | 53.42 ± 0.13 |

a. The locale-averaged results

| Model | High Intent Acc (%) | High Slot F1 (%) | High Exact Match Acc (%) |
|---|---|---|---|
| **xlmr(baseline)** | **85.27 ± 0.65** sv-SE | 80.01 ± 0.75 sv-SE | **72.09 ± 0.82** sv-SE |
| xlmr-hier-concat | 84.77 ± 0.66 sv-SE | 79.34 ± 0.77 sv-SE | 71.22 ± 0.83 sv-SE |
| xlmr-hier-att | 85.10 ± 0.65 sv-SE | **80.05 ± 0.75** sv-SE | 71.86 ± 0.82 sv-SE |
| xlmr-bi-max | 84.33 ± 0.59 sv-SE | 79.52 ± 0.63 sv-SE | 70.75 ± 0.73 sv-SES |
| xlmr-bi-lstm | 84.06 ± 0.67 esv-SE | 79.48 ± 0.76 sv-SE | 70.58 ± 0.84 sv-SE |
| xlmr-bi-att | 84.2 ± 0.67 sv-SE | 79.62 ± 0.76 sv-SE | 70.88 ± 0.83 sv-SE |

b. The best locale results

| Model | Low Intent Acc (%) | Low Slot F1 (%) | Low Exact Match Acc (%) |
|---|---|---|---|
| xlmr(baseline) | 44.42 ± 0.91 ja-JP | 33.04 ± 0.89 ja-JP | 21.89 ± 0.76 ja-JP |
| xlmr-hier-concat | 42.84 ± 0.91 ja-JP | **36.90 ± 0.91** ja-JP | 22.83 ± 0.77 ja-JP |
| xlmr-hier-att | 43.04 ± 0.91 ja-JP | 33.68 ± 0.89 ja-JP | 21.22 ± 0.75 ja-JP |
| xlmr-bi-max | 45.02 ± 0.67 ja-JP | 33.29 ± 0.76 ja-JP | 22.02 ± 0.83 ja-JP |
| xlmr-bi-lstm | 44.86 ± 0.91 ja-JP | 34.57 ± 0.90 ja-JP | 22.09 ± 0.76 ja-JP |
| **xlmr-bi-att** | **47.24 ± 0.92** ja-JP | 34.86 ± 0.9 ja-JP | **24.11 ± 0.78** ja-JP |

c. The worst locale results

As shown in Table 5, all the models improved the locale-averaged exact match accuracy when trained on the full dataset. The hierarchical architecture with attention-based computation (xlmr-hier-att) reached the top by increasing the exact match accuracy from 75.38 (baseline) to 76.39. The averaged intent accuracy and slot F1 scores for the xlmr-hier-att model are also better than the baseline, namely 0.29 and 0.74 units higher.

Regarding the best-performing locales, we can see that the best exact match accuracy was 80.50, corresponding to the bidirectional architecture with attention-based computation (xlmr-bi-att), while it improved the least in the averaged locales. But all the performances in Table 5b show similar points, so we can infer that all the models we presented in this paper did improve the performance for those top locales.

According to Table 5c, the improvement for the worst-performing locales was similar to that for the averaged locales. Compared to the baseline, the xlmr-hier-att architecture had a 1.21 increase in intent accuracy in km-KH, a 1.63 increase in slot score in ja-JP, and a 1.92 increase in extra match accuracy in km-KH, yielding the best results for all the models.

The gaps between the best and worst locales for all the measurements were smaller in most of our models, except for the slot scores in the hierarchical concatenation model (xlmr-hier-concat) and the exact match accuracy in the bidirectional attention-based computation model (xlmr-bi-att). The xlmr-hier-att model had the lowest difference for three measurements: 10.33 for intent accuracy, 8.9 for slot scores, and 8.91 for exact match accuracy.

## 5. Discussion

Comparing the two kinds of methods, namely the attention-based computation models and the concatenation models, we can observe that the hierarchical architecture worked better in the attention-based models, while the concatenation models had a better performance using the bidirectional architecture. When extracting the information from the slot logits, LSTM networks are more efficient than max pooling because of the higher averaged locale accuracies and scores.

The situation was different in the zero-shot case. The two models with attention-based computation performed worse than those using the concatenation method, especially with regard to the averaged exact match accuracy, which was even worse than the baseline. Furthermore, the hierarchical concatenation model (xmlr-hier-concat) showed better results than the bidirectional model, while the bidirectional attention-based model attained the best accuracies in the averaged results and the worst locale.

The best locale-averaged exact match accuracy was 53.91 when the xlmr-hier-concat model was trained on the zero-shot setup; this value was 0.48 higher than the baseline. This model performed well in improving the slot score in the worst locale, increasing it from 33.04 to 36.90. The xlmr-hier-concat model utilized the intent logits to bias the hidden states from the encoder, and this method showed a good performance in updating the slot probabilities in zero-shot training.

Although the result for the xlmr-bi-att model is worse than the baseline in the averaged and best locales, it showed a significant improvement in the worst locales. The intent accuracy grew by 2.82 points, the slot score rose from 33.04 to 34.86, and the exact match accuracy increased by 2.22 points. The number of languages in the low slot score part is also the smallest in all models.

Considering the other goal of reducing the difference between the best locales and the worst locales, all the models accomplished the task except for the xlmr-hier-att model. The xlmr-bi-att model had a minimum gap of 48.39, which is 1.81% smaller than the baseline. Hence, we can say that the xlmr-bi-att model is the best model for zero-shot training.

Overall, the xlmr-hier-att model performed the best for training on the full dataset but was the worst in zero-shot. Xlmr-hier-concat is the best model for the zero-shot setup but was not very good for full training. Some models improved the performance for both full training and zero-shot, like xlmr-bi-max model and xlmr-bi-lstm model, although these improvements were not as remarkable as the others. The xlmr-bi-att model may not be fit for our tasks since it only showed a slight increase in the full training case.

## 6. Conclusions

In this paper, we introduced five models with two architectures based on the backbone of the XLM-R encoder. The hierarchical architecture with concatenation (xlmr-hier-concat) concatenated the intent logits and the hidden features from the encoder output, which is the input of the slot classifier. The bidirectional architecture with concatenation updated the inputs of the slot and intent classifiers in a similar way. According to the method of extracting the slot logits information, we had two versions of a bidirectional concatenation architecture: the max pooling version (xlmr-bi-max) and the LSTM version (xlmr-bi-LSTM).

The attention-based models computed the initial intent logits and slot logits first and improved the performance by summing the attention-based contextual probabilities and initial logits. The hierarchical architecture (xlmr-hier-att) only updated the slot logits according to the initial intent logits, while the bidirectional architecture also updated the intent logits according to the new slot logits.

All models improved the three evaluation measurements in most languages when trained on the full dataset, and the best model increased the intent accuracy, micro-averaged slot F1 score, and exact match accuracy in all testing samples by 0.29, 0.74, and 1.01.

The concatenation updating method worked better than the attention-based method in the zero-shot setup case. This method accomplished our tasks by improving the locale-averaged exact match accuracy from 53.43 to 53.91 in the xlmr-hier-concat model.

We exported the relationship between the intent labels and the slot labels, and the results of all models proved that strengthening the relevance between labels in our algorithm could increase the accuracy of intent and slot classification, especially when we combined the two tasks together and tried to correctly predict intent and slots for every utterance.

The models still have room for improvement. For example, filtering the logits before the computation may be useful since incorrect predictions are also a part of the input in our algorithms. Applying more sophisticated techniques when updating the logits could be another way of improving performance.

Intent classification and slot filling are small parts of natural language understanding, the tasks are the simulation of the part of communication between humans and machines, and we hope that the improvement in dealing with the two tasks together could let the machine understand human's words better.

**Author Contributions:** Conceptualization, software, and writing—review and editing, X.J.; methodology and writing—original draft, W.H.; supervision, project administration, and funding acquisition, Y.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is supported by the National Natural Science Foundation of China (Grant No. 62262059) and the Natural Science Foundation of Guangxi Province (Grant No. 2021JJA170178); the Science and Technology Development Fund of Macau (0004/2020/A1, 0070/2020/AMJ); and the Guangdong Provincial Key R&D Programme: 2019B010148001.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** https://github.com/alexa/massive (accessed on 25 October 2023).

**Acknowledgments:** The authors would like to thank the anonymous reviewers for their contributions to this paper.

**Conflicts of Interest:** Author Yanyan Liang was employed by the CEI High-Tech Research Institute Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## References

1. Wang, Y.-Y.; Deng, L.; Acero, A. Spoken language understanding. *IEEE Signal Process. Mag.* **2005**, *22*, 16–31. [CrossRef]
2. Young, S.J. Talking to machines (statistically speaking). In Proceedings of the INTERSPEECH, Denver, Colorado, USA, 16–20 September 2002.
3. FitzGerald, J.; Hench, C.; Peris, C.; Mackie, S.; Rottmann, K.; Sanchez, A.; Nash, A.; Liam Urbach, V.K.; Singh, R.; Ranganath, S.; et al. Massive: A 1m-example multilingual natural language understanding dataset with 51 typologically-diverse languages. *arXiv* **2022**, arXiv:2204.08582.
4. Xu, W.; Haider, B.; Mansour, S. End-to-end slot alignment and recognition for cross-lingual nlu. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 5052–5063. Available online: https://aclanthology.org/2020.emnlp-main.410/ (accessed on 25 October 2023).
5. Li, H.; Arora, A.; Chen, S.; Gupta, A.; Gupta, S.; Mehdad, Y. Mtop: A comprehensive multilingual task-oriented semantic parsing benchmark. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, Online, 19–23 April 2021; pp. 2950–2962. [CrossRef]
6. Upadhyay, S.; Faruqui, M.; Tur, G.; Dilek, H.-T.; Heck, L. (Almost) zero-shot cross-lingual spoken language understanding. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 6034–6038. [CrossRef]

7. Schuster, S.; Gupta, S.; Shah, R.; Lewis, M. Cross-lingual transfer learning for multilingual task oriented dialog. In *Human Language Technologies, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, Minneapolis, MN, USA, 2–7 June 2019*; Long and Short Papers; Association for Computational Linguistics: Toronto, ON, Canada, 2019; Volume 1, pp. 3795–3805. [CrossRef]
8. Bastianelli, E.; Vanzo, A.; Swietojanski, P.; Rieser, V. Slurp: A spoken language understanding resource package. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 7252–7262. [CrossRef]
9. Liu, X.; Eshghi, A.; Swietojanski, P.; Rieser, V. Benchmarking natural language understanding services for building conversational agents. *arXiv* **2019**, arXiv:1903.05566.
10. Conneau, A.; Khandelwal, K.; Chaudhary, N.G.V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; Stoyanov, V. Unsupervised cross-lingual representation learning at scale. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 8440–8451. [CrossRef]
11. Xue, L.; Constant, N.; Roberts, A.; Kale, M.; Al-Rfou, R.; Siddhant, A.; Barua, A.; Raffel, C. mt5: A massively multilingual pre-trained text-to-text transformer. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online, 6–11 June 2021; pp. 483–498. [CrossRef]
12. Simons, G. *Ethnologue: Languages of the World*; SIL International: Dallas, TX, USA, 2022. Available online: https://www.ethnologue.com/ (accessed on 25 October 2023).
13. Simpson, H.; Cieri, C.; Maeda, K.; Baker, K.; Onyshkevych, B. Human language technology resources for less commonly taught languages: Lessons learned toward creation of basic language resources. In Proceedings of the Collaboration: Interoperability between People in the Creation of Language Resources for Less-Resourced Languages, Marrakech, Morocco, 27 May 2008; p. 7.
14. Strassel S.; Tracey, J. Lorelei language packs: Data, tools, and resources for technology development in low resource languages. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16), Portoroz, Slovenia, 23–28 May 2016; pp. 3273–3280. Available online: https://aclanthology.org/L16-1521 (accessed on 25 October 2023).
15. Goyal, N.; Gao, C.; Chaudhary, V.; Chen, P.; Wenzek, G.; Ju, D.; Krishnan, S.; Ranzato, M.; Guzman, F.; Fan, A. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *arXiv* **2021**, arXiv:2106.03193.
16. Cruz, J.C.B.; Cheng, C. Establishing baselines for text classification in low-resource languages. *arXiv* **2020**, arXiv:2005.02068.
17. Lakew, S.M.; Negri, M.; Turchi, M. Low resource neural machine translation: A benchmark for five african languages. *arXiv* **2020**, arXiv:2003.14402.
18. Magueresse, A.; Carles, V.; Heetderks, E. Low-resource languages: A review of past work and future challenges. *arXiv* **2020**, arXiv:2006.07264.
19. Devlin, J. Multiligual Bert. 2018. Available online: https://github.com/google-research/bert/blob/master/multilingual.md (accessed on 25 October 2023).
20. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
21. Lample, G.; Conneau, A. Cross-lingual language model pretraining. *arXiv* **2019**, arXiv:1901.07291.
22. Liu, Y.; Gu, J.; Goyal, N.; Li, X.; Edunov, S.; Ghazvininejad, M.; Lewis, M.; Zettlemoyer, L. Multilingual denoising pre-training for neural machine translation. *arXiv* **2020**, arXiv:2001.08210.
23. Lewis, M.; Ghazvininejad, M.; Ghosh, G.; Aghajanyan, A.; Wang, S.; Zettlemoyer, L. Pre-training via paraphrasing. *arXiv* **2021**, arXiv:2006.15020.
24. Chen, Q.; Zhuo, Z.; Wang, W. Bert for joint intent classification and slot filling. *arXiv* **2019**, arXiv:1902.10909.
25. Guo, D.; Tur, G.; Tau Yih, W.; Zweig, G. Joint semantic utterance classification and slot filling with recursive neural networks. In Proceedings of the 2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, CA, USA, 7–10 December 2014; pp. 554–559.
26. Rastogi, A.; Gupta, R.; HakkaniTur, D. Multi-task learning for joint language understanding and dialogue state tracking. In Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue, Melbourne, VIC, Australia, July 2018; pp. 376–384. Available online: http://aclweb.org/anthology/W18-5045 (accessed on 25 October 2023).
27. Zhang, C.; Li, Y.; Du, N.; Fan, W.; Yu, P. Joint Slot Filling and Intent Detection via Capsule Neural Networks. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; Association for Computational Linguistics: Toronto, ONT, Canada, 2019; pp. 5259–5267. Available online: https://aclanthology.org/P19-1519 (accessed on 25 October 2023).
28. Vanzo, A.; Bastianelli, E.; Lemon, O. Hierarchical multi-task natural language understanding for cross-domain conversational ai: Hermit nlu. *arXiv* **2019**, arXiv:1910.00912.
29. Liu, B.; Lane, I. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv* **2016**, arXiv:1609.01454.
30. Han, S.C.; Long, S.; Li, H.; Weld, H.; Poon, J. Bi-directional joint neural networks for intent classification and slot filling. *arXiv* **2022**, arXiv:2202.13079.
31. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. *arXiv* **2017**, arXiv:1706.03762.

32. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv* **2016**, arXiv:1609.08144.
33. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
34. Li, L.; Jamieson, K.; Rostamizadeh, A.; Gonina, E.; Hardt, M.; Recht, B.; Talwalkar, A. Massively parallel hyperparameter tuning. *arXiv* **2018**, arXiv:1810.05934.