# Distributionally Robust Finetuning BERT for Covariate Drift in Spoken Language Understanding

**Samuel Broscheit**[1,2], **Quynh Do**[2], **Judith Gaspers**[2],
[1]Data and Web Science Research Group, University of Mannheim, Germany
[2]Amazon Alexa AI

## Abstract

In this study, we investigate robustness against covariate drift in spoken language understanding (SLU). Covariate drift can occur in SLU when there is a drift between training and testing regarding *what* users request or *how* they request it. To study this we propose a method that exploits natural variations in data to create a covariate drift in SLU datasets. Experiments show that a state-of-the-art BERT-based model suffers performance loss under this drift. To mitigate the performance loss, we investigate distributionally robust optimization (DRO) for finetuning BERT-based models. We discuss some recent DRO methods, propose two new variants and empirically show that DRO improves robustness under drift.

## 1 Introduction

A common assumption in machine learning is that training data and test data are independent and identically distributed (i.i.d.). Unfortunately, this may not hold in practice and the test distribution might have drifted from the training distribution which can lead to a significant drop of performance in real-world applications (Moreno-Torres et al., 2012).

Consider spoken language understanding (SLU), i.e., the task of mapping an utterance to a machine readable semantic interpretation, which is commonly used in voice controlled devices like Alexa, Siri or Google Assistant. Distributional drifts can be caused by seasonal and non-seasonal factors. For example, festive holidays can lead to many requests outside the daily routine. New users might use an uncommon phrasing to express their intent or they might request an uncommon song to be played. Such drifts in the input distribution are referred to as covariate drift. When users' requests fail to be recognized by the device they might rephrase their intent until they succeed, essentially adapting to the SLU model's distribution. This means that, even when new training samples are drawn from new user utterances, the dominance of the old distribution already present in the SLU model is reinforced. Fine-tuned pre-trained language models (PLM), such as BERT (Devlin et al., 2019) yield strong performance on SLU benchmarks (Chen et al., 2019). Yet, it has been observed that also PLMs are vulnerable to drifts, and there is a high interest in understanding the robustness of PLMs (Oren et al., 2020a; McCoy et al., 2019; Tu et al., 2020; Cao et al., 2020).

The goals of this study are to investigate the impact of covariate drift on BERT's performance, and to experimentally investigate distributionally robust optimization (DRO) for finetuning BERT. While we focus on sequence classification for SLU, i.e., intent classification (IC) and slot filling (SF), we expect the insights of this study to be applicable also to other sequence classification tasks.

To study the impact of covariate drift on model robustness, we require a dataset with known properties of the drift. However, real data for this setting is not publicly available. Therefore, we devised a method to create a train/test split with a controlled drift for sequence labeling data, which we call SEQDRIFT. Roughly speaking, SEQDRIFT creates clusters of examples based on the example's tokens and sequence labels. Then those clusters are used to create a new train/test split while leaving the label distribution intact. Notably, SEQDRIFT does not artificially alter the utterances and only exploits natural lexical variations in the data in a non-adversarial manner. Our experiments on publicly available SLU datasets repartitioned with SEQDRIFT showed that a state-of-the-art BERT-based model for SLU (Chen et al., 2019) trained with standard optimization suffers up to 5% absolute performance loss.

Currently, it is an open question which range of measures are helpful to improve the generalization under drift. In this study, we investigated distributionally robust optimization (DRO), which has

recently gained interest in NLP for overparameterized models (Oren et al., 2019; Sagawa et al., 2020; Liu et al., 2021; Michel et al., 2021). It is an optimization concept which assumes that the training data is a mixture of distributions, e.g., different user demographics. The objective is to be optimal under each distribution. For example, the methods proposed by Oren et al. (2019) and Sagawa et al. (2020) assume knowledge about groups of training instances —such as topics or ethnic origin— that can be used by the optimization. Roughly speaking, they propose to compute the loss across groups instead across individual instances. However, such group knowledge might not be available and there are other methods which do not require such prior knowledge. TOPK (Levy et al., 2020; Kawaguchi and Lu, 2020), for example, simply uses the top-k largest losses in a batch and was shown to obtain robust models.

We performed an extensive experimental analysis to investigate the usefulness of several DRO methods across different scenarios. Most studies only evaluate DRO methods in one setting with in-distribution validation data and one drift type per dataset. To achieve a broader insight into the usefulness of the investigated methods we evaluated them in 8 scenarios per dataset, i.e., for different types of drift, or model selection with in-distribution and out-of-distribution validation data. Additionally, we propose an intuitive variant of TOPK, namely TOPK-GROUP or TOPK-AUTOENCODER to investigate if prior group knowledge or latent group knowledge could improve TOPK. We found that TOPK, TOPK-GROUP and TOPK-AUTOENCODER can significantly improve robustness in many scenarios, where TOPK is more reliable in terms of significant improvement, while TOPK-AUTOENCODER can be better in terms of relative improvement.

## 2 Background

In this section, we provide a brief background to spoken language understanding. Subsequently, we discuss common categorizations of dataset drifts, empirical analyses of drifts and then describe distributionally robust optimization.

### 2.1 Spoken Language Understanding

In this study, we focus on SLU for single-turn utterances and non-nested intents. Parsing utterances into API calls is broadly either done by task oriented semantic parsing (TOP), or as intent classification (IC) and slot filling (SF). IC is the task to classify an utterance into user intents, such as *PlayMusic*, *FindBook* or *GetWeather*. Meanwhile, SF is a sequence tagging task to identify spans of tokens that represent the intent's slot fillers, such as *ArtistName*, *AlbumName* or *TrackNumber*. In state-of-the-art approaches, IC and SF are typically modeled jointly using deep neural networks (Chen et al., 2019).

### 2.2 Distributional Drifts

A common assumption in machine learning is that the training and test data are independent and identically distributed (i.i.d.) and that the distributions are the same between training and test, i.e. $P_{train}(x, y) = P_{test}(x, y)$. Unfortunately, in practice, the test data is often out of distribution (o.o.d.), i.e. $P_{train}(x, y) \neq P_{test}(x, y)$. This can be caused by sampling bias, such that subpopulations are not equally represented in the samples of the two distributions. As this phenomenon is often caused by time-varying covariates, i.e., seasonal and non-seasonal changes, this phenomenon is referred to as a *drift*. However, this can be a general mismatch between the sampled subpopulations, which can be of geographic or demographic type, or they can be topics or domains. Drifts can also be caused by noise or automatic training data generation, in which filtering heuristics introduce a systematic issue, or by adversaries that exploit weaknesses of a specific model or model class.

Distributional drifts can be categorized into (Moreno-Torres et al., 2012): concept drift, i.e., when the meaning changes, prior probability drift and covariate drift.

**Covariate drift.** When $P_{train}(x) \neq P_{test}(x)$, then there is a drift in the input distribution, and when the concept $P(y|x)$ does not change between training and testing, this is referred to as covariate drift. The challenge is that the population and its subpopulations are unknown because only samples are observed and thus $P(x|y)$ cannot be perfectly learned. For covariate drift a model has to generalize to samples from subpopulations that are almost unseen, e.g., a spam classifier should generalize to a new spam campaign $(x, y)_{test}$ = ( word_in_email="cheap", is_spam=True) while only observing $(x, y)_{train}$ = (word_in_email="money", is_spam=True). One of the conjectures for transfer learning using PLMs is

that task finetuning PLMs can generalize to inputs that are semantically similar to training instances. One reason for a lack of robustness to covariate drift is when models overfit on patterns between the input and the desired labels, e.g., when they would learn that only "cheap" is a predictor for spam. Another reason can be spurious correlations between patterns in the input and the label. Sagawa et al. (2020); Tu et al. (2020) showed this for entailment prediction. They grouped instances by a certain input attribute (does or does not contain negation) and target labels (is or is not entailment). The attribute did not have any direct relation to the label. Then they partitioned the data in such a way that there was a correlation between the polarity of the attribute and the label in the training data and an inverse correlation in the test data: $D_{train} = [(negation = +, entailment = -), (negation = -, entailment = +)]$ and $D_{test} = [(negation = +, entailment = +)], (negation = -, entailment = -)]$. They showed that models learn this correlation instead of the semantics of the actual task and then fail on test instances.

### 2.3 Empirical Analyses of Drifts

Currently, there is a rising interest to investigate distributional drifts in various domains (Tu et al., 2020; Sagawa et al., 2020; Koh et al., 2021; Dunn et al., 2020; Shankar et al., 2019; Oren et al., 2020b), most prominently in Computer Vision (CV), but also in NLP. To study distributional drifts, researchers need datasets with a controlled drift between training and test data. This can be broadly achieved in two ways:

**Synthetic methods.** A dataset drift is created by corrupting the input features with synthetic noise, for example, adding pixel noise (Goodfellow et al., 2015), perturbing the input with generative models (Dunn et al., 2020) or perturbing characters and words (Cao et al., 2020). It has been observed that robustness against synthetic noise does not imply robustness against semantically meaningful perturbations (Koh et al., 2021; Dunn et al., 2020).

**Natural variations.** Another option is to exploit natural variations, for example, using video frames for which a model's object prediction flips between adjacent frames (Shankar et al., 2019). Koh et al. (2021) collected a large benchmark for naturally occuring drifts in CV and NLP, e.g., user demographics for toxicity detection in online comments. Søgaard et al. (2021) investigated the difference of model performance comparing random splits with heuristics like splitting the data based on sentence length or by maximizing the divergence of the token feature vectors of the train and test split. In this work, we exploit natural variations in the data to create a drift in a non-adversarial manner. Our conjecture is that this setting is a good proxy to a realistic evaluation scenario.

### 2.4 Distributionally Robust Optimization

**Robustness.** The robustness of a machine learning model is the property that characterizes how effective the model is while being tested on a new dataset. In this paper, robustness is formally defined as follows. Let $D$ be a dataset split into $D_{train}, D_{valid}, D_{test}$ and let $E$ be a performance measure $E : \theta \times D \rightarrow R$ (w.l.o.g. greater is better). We assume that there is a covariate drift between $D_{train}$ and $D_{test}$. Given two models $A$ and $B$ with parameters $\theta_A, \theta_B \in \Theta$ estimated on $D_{train}$. We call a model $B$ more robust than model $A$ when $E(\theta_B, D_{test}) > E(\theta_A, D_{test})$ and $E(\theta_B, D_{valid}) - E(\theta_B, D_{test}) < E(\theta_A, D_{valid}) - E(\theta_A, D_{test})$

**Empirical Risk Minimization (ERM).** Commonly used optimization algorithms assume that all examples are from the same population. This assumption stems from ERM's optimization objective which treats each example in $D_{train} \sim P$ with equal importance, i.e., $\hat{\theta} = inf_\theta E_P[l(x, y; \theta)]$. This optimization may have a negative impact on model robustness. For example, Tu et al. (2020) found that using ERM for finetuning PLMs learns spurious correlations even in the presence of a few helpful counter examples.

**Distributionally Robust Optimization (DRO).** DRO is based on the assumption that $D_{train}$ consists of samples from many subpopulations, i.e., distributions $Q$ from an uncertainty set $U(x, y)$. The objective in DRO is then to optimize the parameters such that they are optimal under the worst case distribution in $U(x, y)$, i.e., $\hat{\theta} = inf_\theta sup_Q \mathbb{E}_Q[l(x, y; \theta)]$. DRO is effective when the proportions of the distributions $Q$ are highly skewed in $D_{train}$. For example, this can help to avoid learning spurious correlations, because even very few counter examples in the data are amplified. The challenge in applying the DRO concept is that the subpopulations are not observable and $U(x, y)$ has to be modeled by some prior knowledge about the data.

| O | B-SONG | O | B-ARTIST | I-ARTIST | O | O | B-PLAYLIST | O |
|---|---|---|---|---|---|---|---|---|
| add | song | by | too | poetic | to | my | piano-ballads | playlist |

Table 1: Example for the *slot value drift* feature representation. Only the values in black can be considered for the n-gram feature representation and the grey values are ignored.

| O | B-SONG | O | B-ARTIST | I-ARTIST | O | O | B-PLAYLIST | O |
|---|---|---|---|---|---|---|---|---|
| add | song | by | too | poetic | to | my | piano-ballads | playlist |

Table 2: Example for the *slot context drift* feature representation. Only the values in black are used for the n-gram feature representation and the gray values are ignored.

## 3 The SEQDRIFT Method to create Covariate Drift Benchmarks

Our goal is to study the impact of *covariate drift* on model performance. Therefore, we need a benchmark with controlled drift, but currently there are no publicly available SLU benchmarks in which real drifts can be studied. As motivated in Section 2.3, we do not want to employ synthetic noise, i.e., our goal is to design a method that exploits natural variations in the data. Moreover, the method should not be adversarial, i.e., not designed or optimized to target a specific model or model class. Instead, we target two semantic drifts that might occur in real data due to: *how* users express their intent, and *what* users request.

We conjecture that it is possible to capture *how* users express themselves by creating clusters of utterances with similar slot contexts. To capture *what* users request could be achieved by clusters of utterances with similar slot values. A drift can then be created by partitioning the data based on those clusters into training and testing. We avoid creating a mismatch of the label distributions between training and testing. If a mismatch would occur, it would not be possible to derive conclusions about covariate drift from changes in performance because the shift in the label distribution also leads to changes in the measured performance. In the following, we describe our approach in detail.

### 3.1 Overview

The high-level overview for creating a drift dataset version is as follows: (i) Join all splits from the original data. (ii) Transform examples into feature representations. (iii) Use spectral clustering to obtain $K$ clusters based on the feature representations. (iv) Create the test split based on the clusters by sampling clusters instead of sampling examples.

### 3.2 Feature representation for clustering

We propose two variants of feature representations to capture different drift types.

**Slot value drift** To cluster examples by "*what* users request" we chose the feature representation of slot value n-grams. Table 1 shows an example in which only the slot values (the non-gray cells) are used to generate n-grams for an utterance, e.g. "`song`" or "`too poetic`". The expected effect of splitting the data based on clusters of examples using this representation is that the training split is missing certain slot values, and thus we encounter unseen artists during testing.

**Slot context drift** The feature representation to cluster training examples by "*how* users express an intent or slot" are n-grams of slot labels and the tokens around them. For example, using only the non-gray cells in Table 2 to generate n-grams would yield "`add B-SONG by B-ARTIST`" or "`to my B-PLAYLIST`" as features to represent the example. The expected effect of this drift is that the test data contains phrases which are not seen during training.

### 3.3 Creating new train/valid/test splits

Now, using the feature representation for either the *slot value drift* or *slot context drift*, we use spectral clustering to create $K$ clusters and proceed to create the data splits.

**Test split.** First the test split is created by sampling *clusters* and all the clusters' examples are added to the test split. To avoid a mismatch of the label distribution between training and the new test split, the method uses a projected label count per split to decide whether a cluster can be used. For example, let's assume we defined a 5% test split percentage and there are 1000 examples with the intent-slot label PLAYMUSIC-ARTIST. Then the test split should have $\approx 50$ examples with the

intent-slot label PLAYMUSIC-ARTIST. Hence, a cluster which contains 70 examples with the intent-slot label PLAYMUSIC-ARTIST cannot be used for the test split because it would exceed the projected label count. Thus, when a cluster is sampled *all* of its examples are added to the test split if they do not disturb the projected label count. This is repeated until all clusters have been sampled once and have been added to the test split or not. When the test split does not match the projected label count, it is filled using random *examples* from clusters that have not been used for test so far. These examples do not count into the controlled drift.

**Train and validation split.** The training and validation splits are created by sampling from the remaining *examples*.

### 3.4 Drift dataset variants

We also considered the following variations of the proposed algorithm to measure various effects.

**O.O.D. validation** One variation is that the validation data could be o.o.d. instead of distributed like the training data. This is a hypothetical setting in which we have access to o.o.d. data for validation and can observe to what degree hyperparameter tuning and model selection do factor into the drift effect. To achieve this we create the validation data in the same way as the test data, but validation and test do not share drift clusters.

**Full drift and partial drift** In the default behavior *all* the examples of a cluster are shifted into the test split which we call a **full drift**. However, a natural question is what happens when a small percentage of a test cluster leaks into training. We call this setting a **partial drift**.

## 4 DRO for Overparameterized Models

In the experiments in Section 5 we will show that using ERM optimization on the SEQDRIFT partitioned datasets is not robust. There might be many measures to mitigate this effect, and the best solution will most likely consist of a mix of methods. One candidate is DRO that has seen a rising interest to be applied to overparameterized models. In the following, we first briefly discuss the setting of finetuning a pretrained language model (PLM), and subsequently we describe existing and proposed DRO methods.

### 4.1 Finetuning Pretrained Language Models

In our setup, a pretrained language model $M$ consists of a pretrained encoder $ENC$, and one (or more) task classifier head(s) $C_{task}$. Let $X$ be a batch of inputs of size $b$, then the hidden representations of $M$ are the output of the encoder $X_{enc} = ENC(X)$. For example, in our study we denote the averaged hidden token representations of size $d$ after the last layer of BERT as $X_{enc} \in \mathbb{R}^{b \times d}$. To finetune $M$ for a new task, the parameters of the encoder and the task classifier heads are optimized with a loss function $L_{task}$ to obtain the task batch loss $l_{task} = L_{task}(C_{task}, X_{enc}) \in \mathbb{R}^b$. The following methods differ mainly in the way they manipulate the task batch loss $l_{task}$.

### 4.2 Existing DRO methods

The following DRO methods are by no means exhaustive. They represent either methods proposed so far in NLP or have a desirable property, e.g., being simple or conceptually interesting. The main differences between the methods is that they either use or do not use group knowledge in their objective. Those models that do require knowledge about groups in the data will use the clusters created by the SEQDRIFT algorithm. However, using the SEQDRIFT clusters is somewhat artificial because this is perfect information. Therefore, we are especially interested in methods that do not require group knowledge.

**TOPIC-CVAR** This method was proposed by Oren et al. (2019) for language modeling. They use a topic model to obtain a distribution over topics for each sentence to model the uncertainty set. The core idea is to accumulate the losses for each topic over the course of training. In each update a subset of losses in $l_{task}$ is selected, i.e., the losses of those batch items that are assigned to the topic that currently lies in the upper $\alpha$ percentile of accumulated losses.

**GROUP-DRO** This method was proposed by Sagawa et al. (2020) for data where groups are known such that each example is assigned to one group. Similar to TOPIC-CVAR their method keeps statistics of the accumulated losses, but for groups rather than for topics. In GROUP-DRO the batch losses in $l_{task}$ are first averaged per group and the final loss is a weighted average over group losses. For batch construction their method upsamples groups reciprocally to their frequency.

| | Prec. | Adapt. | Impl. |
|---|---|---|---|
| TOPK-AE (our) | | x | |
| TOPK-GROUP (our) | x | | |
| TOPK | | | x |
| GROUP-DRO | x | | |
| TOPIC-CVAR | x | | |

Table 3: How the group knowledge is modeled in DRO methods: precomputed, adaptive, implicit.

**TOPK** This method does not require group knowledge and is simple to implement: it simply computes the loss as an average over the top-$k$ largest losses in $l_{task}$ (Levy et al., 2020; Kawaguchi and Lu, 2020).

### 4.3 Our proposed variants

We found TOPK to be very effective in initial experiments. By contrast, GROUP-DRO and TOPIC-CVAR did not perform well in our setting, even though both have been shown to work well. Thus, we propose the following TOPK variants:

**TOPK-GROUP.** If group information is available, can TOPK be improved by it? Here the idea is —similar to TOPIC-CVAR and GROUP-DRO— to use the precomputed SEQDRIFT clusters as groups and compute the TOPK loss per group. Then only the largest TOPK group loss is picked, which has the effect of upsampling "difficult" groups and downsampling "easy" groups over the course of training. However, when the precomputed SEQDRIFT clusters are used, this is more an oracle, i.e., an upper bound of how much can be inferred from the training data using perfect information.

**TOPK-AUTOENCODER (TOPK-AE).** What if we do not have access to the precomputed clusters? Could we approximate them using the PLM's hidden representations $X_{enc}$? Our idea is to use the $X_{enc}$ representations to cluster the $b$ batch items into $c$ latent groups. The latent groups are then used in the loss computation like in TOPK-GROUP. The clustering is obtained from an autoencoder which is trained on $X_{enc}$ and is continuously updated during training. Thus, the group assignment of a training example can change over the course of training according to the model's changing hidden representations. We investigated hard cluster assignment TOPK-AE-BIN and soft cluster assignment TOPK-AE-PROB. See Appendix B for all the details regarding the autoencoder and its training.

**Discussion** Table 3 compares the different methods discussed in this study, and shows if the method relies on precomputed groups or if the groups are implicit or adaptively inferred during training.

## 5 Experiments

In this section, we present our experiments to investigate the following questions: **(Q1)** Does the standard optimization ERM suffer a performance loss under the SEQDRIFT covariate drift? **(Q2)** How well can ERM and DRO methods exploit a scarce signal about the test distribution, i.e., when is DRO relevant? **(Q3)** As all optimization methods come with hyperparameters, how much better could each method perform with access to o.o.d. validation data to optimize hyperparameters and perform early stopping? Would DRO still be better than ERM? **(Q4)** Are the DRO methods more robust than ERM against the SEQDRIFT covariate drift, and which DRO method is the most effective?

**SLU Model.** We use the JointBERT model for SLU (Chen et al., 2019). Two small changes that we introduce are: (i) an intent loss scaler $\gamma$ for the joint tasks loss $L = L_{slot} + \gamma * L_{intent}$ and (2) using softmax layer instead of CRF for the sequence tagging classifier. We established the usefulness of those two changes with a hyperparameter study[1].

### 5.1 SEQDRIFT Datasets

The source datasets for SEQDRIFT are four commonly used SLU benchmarks, which are listed in Table 6. All technical details and settings for SEQDRIFT are discussed in Appendix A. Table 5 shows an excerpt from a cluster from the ATIS dataset and demonstrates how the *slot context drift* cluster contains examples with similar phrases, in this case utterances with the phrase "`between B-from.city and B-to.city`".

**Use of datasets.** To study robustness it is inevitable to look at test performance. Thus, we did not use all datasets for all stages of experimentation: Prototyping of SEQDRIFT was only done on ATIS, and then final experiments with ERM were done on all four datasets. The prototyping and initial experiments for the DRO methods were mostly done on ATIS and a few trials on SNIPS. The final DRO experiments were conducted on SNIPS and TOP.

[1] We found that CRFs did not help and task loss scalers for good models had a ratio of slot:intent of 100:1.

| | | | train | | valid | | | | test | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | i.i.d. | | o.o.d. | | | |
| | | | size | % drift | size | % drift | size | % drift | size | % drift |
| SNIPS | full | slot value | 10,231 | 0 | 1,426 | 0 | 1,423 | 25 | 1,404 | 66 |
| | | slot label | 10,197 | 0 | 1,390 | 0 | 1,416 | 52 | 1,400 | 75 |
| | partial | slot value | 10,238 | 2 | 1,412 | 0 | 1,415 | 40 | 1,419 | 67 |
| | | slot label | 10,214 | 2 | 1,400 | 0 | 1,421 | 59 | 1,449 | 79 |
| TOP | full | slot value | 16,145 | 0 | 2,295 | 0 | 2,257 | 23 | 2,253 | 44 |
| | | slot label | 16,131 | 0 | 2,227 | 0 | 2,251 | 65 | 2,341 | 45 |
| | partial | slot value | 16,128 | 2 | 2,272 | 0 | 2,294 | 26 | 2,256 | 45 |
| | | slot label | 15,940 | 3 | 2,255 | 0 | 2,316 | 65 | 2,439 | 47 |

Table 4: Statistics for all scenarios for the SEQDRIFT versions of SNIPS and TOP used in the main experiments.

| INTENT | SLOTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| atis_flight | show | flights | between | B-from.city boston | and | B-to.city philadelphia | | | |
| atis_flight | what | nonstop | slights | between | B-from.city boston | and | B-to.city washington | arrive | B-arrive_time today |
| atis_flight | what | flights | are | between | B-from.city boston | and | B-to.city atlanta | in | B-depart_time july |
| atis_flight | flights | between | B-from.city boston | and | B-to.city philadelphia | that | arrive | after | B-depart_time 2pm |

Table 5: An excerpt from a cluster from the ATIS dataset for the slot context drift.

**All dataset scenarios** In total we can evaluate a method in eight different *scenarios* per dataset, i.e., the cross product of

{slot value drift, slot context drift} ×
{partial drift, full drift} ×
{i.i.d. validation, o.o.d. validation}.

Table 4 shows the resulting statistics for the datasets SNIPS and TOP-NN. The percentage of examples resulting from a *controlled drift* in the test set are $66-79\%$ for SNIPS and $44-47\%$ for TOP. For the scenario with *partial drift* 2-3% of the training data split belong to clusters that have been deliberately shifted into the test split.

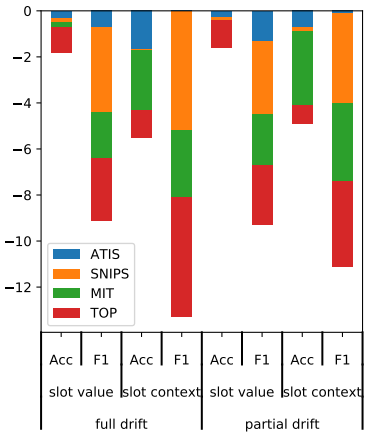
**Metrics.** We use the following metrics: F1 - the slot F1 metric; ACCURACY - the intent accuracy; COMBINED-IC-SF - the average of F1 and Accuracy.

**Hyperparameters.** To ensure a fair comparison of methods in the experiments, we performed a hyperparameter search with the objective to optimize for COMBINED-IC-SF for each optimization method for four scenarios {slot value drift, slot context drift} × {i.i.d. validation, o.o.d. validation} with partial drift (see Section 3.4). For each setting we ran 8 hyper-parameter optimization steps[2], then picked the two best hyper-parameter settings and retrained them with a different random seed. Then we reused the best hyper-parameters for the full drift. See Appendix C and Table C.9 for the remaining details about the hyperparameters.

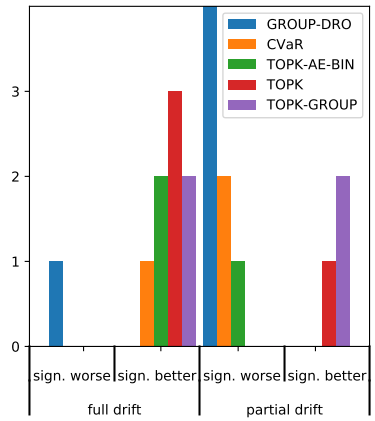
| | #int. | #slot | #int.-slot | #examp. |
|---|---|---|---|---|
| ATIS | 26 | 82 | 389 | 5871 |
| TOP-NN | 18 | 25 | 109 | 29104 |
| MIT | 2 | 18 | 20 | 21399 |
| SNIPS | 7 | 39 | 52 | 14484 |

Table 6: Source Datasets for our experiments. ATIS (Hakkani-Tür et al., 2016), TOP (Gupta et al., 2018), SNIPS (Coucke et al., 2018), MIT (Liu et al., 2013). For TOP we only use non-nested intents, which leaves roughly 70% of the original dataset.

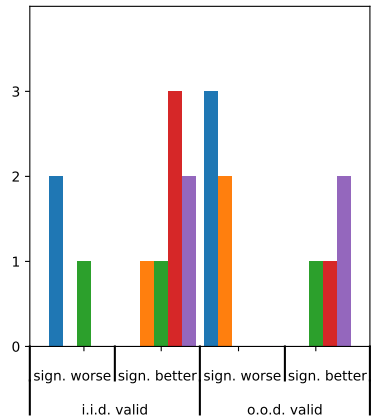
[2] using Optuna




(a) Drop in absolute ERM performance from validation to test on four datasets with two drift types (slot value, slot context) and drift percentage (full, partial).




(b) Amount of times a method was significantly worse or better than ERM on SNIPS for either full drift or partial drift out of 4 scenarios each.




(c) Amount of times a method was significantly worse or better than ERM on SNIPS when the validation data is i.i.d. or o.o.d., i.e. out of 4 scenarios each.

### 5.2 Results

The reported results for each scenario are always averaged from four models, i.e., the models obtained with the two best hyperparameter settings that each have been trained with two random seeds. We computed significance with $p < 0.05$ between models with approximate randomization (Noreen, 1989). Due to the repetition with different random seeds, this effectively results in a family-wise error rate of 0.185. In Figures 1b and 1c we count in how many scenarios a DRO method was significantly better or worse than ERM. The remaining instances performed the same as ERM.

**(Q1) Does the SEQDRIFT covariate drift lead to a drop in performance for ERM?** In Figure 1a, it can be observed that ERM's performance does drop up to 5% in slot F1 between validation and test. However, the amount of change varies between datasets. In most scenarios, slot F1 suffers a higher drop in performance than intent accuracy. *Slot context drift* yields a higher loss than the *slot value drift*, so it seems that it is easier to generalize to unseen slot values than to unseen phrases. This makes intuitively sense, e.g., *"Please play New Unknown Artist."* can be recognized by just knowing the sequence "`please play B-ARTIST I-ARTIST ...`", but it is more difficult to generalize to a new unseen phrase. See Appendix D Table 10 for the numerical results.

**(Q2) How well can ERM and DRO methods exploit a scarce signal about the test distribution?** In Section 3.4 we described the *partial drift*, in which $2-3\%$ of the training data are leaked examples from test clusters. Thus, during training there is some information about test clusters that could be exploited. For ERM, Figure 1a shows indeed that the *partial slot context drift* leads to a smaller drop in performance than the *full slot context drift*. Thus, we conclude that ERM can exploit this information.

For DRO, Figure 1b shows that there are less scenarios with significant improvement from DRO methods over ERM with a *partial drift* than with a *full drift* (see numerical results Appendix D Table 14). It is important to note that all methods —ERM and DRO— do improve, but ERM improves more than most of the DRO methods. Only TOPK-GROUP and TOPK still improve over ERM. Anecdotally, in a SEQDRIFT setting in which only 80% or less of the clusters are drifted into the test split and the percentage of the drift cluster examples make up more than 5% of the training data, the significant improvement of all the DRO methods vanishes.

**(Q3) Does o.o.d. validation data help hyperparameter optimization and early stopping?** In Figure 1c, we observe that the amount of significant improvement over ERM shrinks when the validation data is o.o.d. and thus contains information how to perform well for the test split. This affects hyper-parameter optimization and early stopping which also helps ERM to obtain a model from the training data that performs better on the test distribution. This can serve as an upper bound of possible improvement that can be derived from the

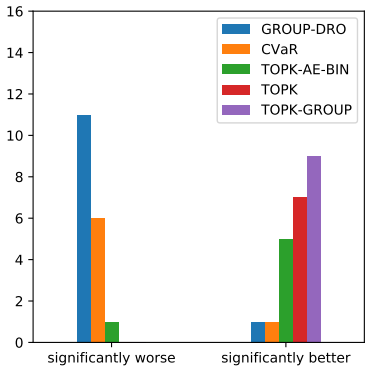



Figure 2: Amount of times a method was significantly worse or better than ERM on SNIPS and TOP out of 16 scenarios.

training data alone. See Appendix D Table 13 for the detailed numerical results.

**(Q4) Are the DRO methods more robust against the drifts than ERM?** Figure 2 shows in how many scenarios the DRO methods improved significantly over ERM on the SNIPS and TOP dataset. TOPK and TOPK-GROUP only improve significantly over ERM and do not perform worse. TOPK-AE-BIN only performs worse one time but otherwise the same or better. The results indicate that TOPK-based methods do improve robustness. TOPK-GROUP performs best amongst all methods, i.e., group information helps TOPK-based methods. TOPK-AE-BIN performs slightly worse in terms of significant improvement than TOPK, however, in terms of average relative improvement over ERM it is on par or better than TOPK (see Tables 11 and 12 in the Appendix). Yet, the lesser amount of significant improvement of TOPK-AE-BIN in comparison to TOPK-GROUP shows that approximating the group information is difficult. Without perfect group information a simple method like TOPK might be the most reliable method to obtain a robust model. See more detailed results in Appendix D Table 11 and Table 12.

**Discussion** GROUP-DRO, TOPIC-CVAR and TOPK-GROUP use the SEQDRIFT clusters in their optimization. Therefore, these results should be rather seen as an upper bound of how much can be inferred from the training data using perfect information. Still, GROUP-DRO and TOPIC-CVAR both fail to perform well in this experiment. Note that both methods had the same amount of budget for hyper-parameter optimization as other methods. For GROUP-DRO we used the authors' published code[3] and also their implementation of TOPIC-CVAR. Our conjectures about this finding are: (1) GROUP-DRO and TOPIC-CVAR both have been proposed and studied for groups that have much higher lexical variance than the groups in our data. The groups in our dataset consist by construction of many examples with similar lexical patterns and can be of small size, i.e., as little as 10 examples. This might explain why they seem to overfit heavily. (2) Another difference to our methods is that our proposed methods do not use an exponential average of historical group loss statistics.

## 6 Conclusions

We studied finetuning BERT for SLU datasets with covariate drift. We presented the SEQDRIFT method to induce a covariate drift for SLU sequence classification tasks. The experimental results showed that this drift in the input distribution leads to a drop in performance on four SLU datasets for a common BERT-based SLU model finetuned with ERM. We investigated DRO methods that either use or do not use knowledge about groups in the data. Our empirical results in an extensive study indicate that TOPK-based DRO methods are successful in improving robustness on the drift datasets.

## Acknowledgements

We would like to thank Rainer Gemulla, Patrick Lehnen and ACL reviewers for helpful feedback for revising and improving the paper.

## References

Xu Cao, Deyi Xiong, Chongyang Shi, Chao Wang, Yao Meng, and Changjian Hu. 2020. Balanced joint adversarial training for robust intent detection and slot filling. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 4926–4936. International Committee on Computational Linguistics.

Q. Chen, Z. Zhuo, and W. Wang. 2019. BERT for joint intent classification and slot filling. *arXiv:1902.10909*.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded

[3] https://github.com/kohpangwei/group_DRO

spoken language understanding system for private-by-design voice interfaces. *CoRR*, abs/1805.10190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Isaac Dunn, L. Hanu, Hadrien Pouget, D. Kroening, and T. Melham. 2020. Evaluating robustness to context-sensitive feature perturbations of different granularities. *arXiv: Computer Vision and Pattern Recognition*.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2787–2792. Association for Computational Linguistics.

Dilek Hakkani-Tür, Gökhan Tür, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 715–719. ISCA.

Kenji Kawaguchi and Haihao Lu. 2020. Ordered SGD: A new stochastic optimization framework for empirical risk minimization. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 669–679. PMLR.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M. Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. 2021. WILDS: A benchmark of in-the-wild distribution shifts. 139:5637–5664.

Daniel Levy, Yair Carmon, John C. Duchi, and Aaron Sidford. 2020. Large-scale methods for distributionally robust optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Evan Zheran Liu, Behzad Haghgoo, Annie S. Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. 2021. Just train twice: Improving group robustness without training group information. 139:6781–6792.

Jingjing Liu, Panupong Pasupat, Scott Cyphers, and James R. Glass. 2013. Asgard: A portable architecture for multilingual dialogue systems. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 8386–8390. IEEE.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3428–3448. Association for Computational Linguistics.

Paul Michel, Tatsunori Hashimoto, and Graham Neubig. 2021. Modeling the second player in distributionally robust optimization.

Jose G. Moreno-Torres, Troy Raeder, RocíO Alaiz-RodríGuez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recogn.*, 45(1):521–530.

Eric W Noreen. 1989. *Computer-intensive methods for testing hypotheses*. Wiley New York.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020a. Improving compositional generalization in semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 2482–2495. Association for Computational Linguistics.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020b. Improving compositional generalization in semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2482–2495, Online. Association for Computational Linguistics.

Yonatan Oren, Shiori Sagawa, Tatsunori B. Hashimoto, and Percy Liang. 2019. Distributionally robust language modeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4226–4236. Association for Computational Linguistics.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. 2020. Distributionally robust neural networks.

Vaishaal Shankar, Achal Dave, Rebecca Roelofs, Deva Ramanan, Benjamin Recht, and Ludwig Schmidt. 2019. Do image classifiers generalize across time? *arXiv preprint arXiv:1906.02168*.

Anders Søgaard, Sebastian Ebert, Jasmijn Bastings, and Katja Filippova. 2021. We need to talk about random splits. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1823–1832, Online. Association for Computational Linguistics.

Lifu Tu, Garima Lalwani, Spandana Gella, and He He. 2020. An empirical study on robustness to spurious correlations using pre-trained language models. *Trans. Assoc. Comput. Linguistics*, 8:621–633.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

| | | SNIPS | TOP |
|---|---|---|---|
| Nr of clusters | | 100 | 100 |
| Min. freq. of intents | | 150 | 150 |
| Min. freq. of slots | | 50 | 50 |
| Min. proj. size | | 10 | 10 |
| n-gram | min | 2 | 2 |
| | max | 6 | 5 |
| Top freq. n-grams | | 10,000 | 10,000 |
| Drift percentage | Full | 100% | 100% |
| | Partial | 90% | 90% |

Table 7: Hyperparameters for the creation of the SNIPS and TOP drift datasets.

## A Drift dataset creation

The procedure described in Section 3 has a range of hyperparameters that we did set manually. Our goal was to improve the clustering not in an adversarial way. Thus those hyperparameter choices were picked by inspecting the clusters and assessing if they did display desired properties independently of the downstream experiments. For example, the number of clusters was set to 100 as shown in Table 7 and was set large enough such that the clustering algorithm did not have to mix clusters into each other.

**Clustering algorithm** The clustering algorithm we used was the spectral clustering implementation in https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html. The feature vector of an example was a weighted indicator vector over the top most frequent n-grams in the dataset. If an example contains the kth most frequent n-gram, then the kth component of this vector was set to $n^2$, otherwise it was set to 0. The effect of the $n^2$ weighting is to create a higher affinity between examples that share longer n-grams than shorter n-grams. The affinity matrix for the spectral clustering was computed with cosine similarity.

**Settings for the dataset** The following settings were used in the dataset creation in our experiments and are listed in Table 7: (a) The number of clusters. (b) Thresholds to filter out low frequency intents and low frequency slot label types, i.e., the minimum frequency of intent labels and slot labels in the dataset. (c) The minimum size of projected label counts that we attempt to match. If this parameter is set too low then many clusters might be discarded because they would violate the projected label count. Even when we did not match some of the projected label counts we did achieve correlations $\geq 98\%$ between training and testing. (d) The range of n-gram sizes. (e) How many of the top most frequent n-grams will be used for the feature vector of an example. (f) We either create a full drift in which 100% of the examples in a cluster are assigned to the test split and a partial drift in which we assign only 90% of the examples in a cluster to the test split.

**Discussion** Table 4 lists the statistics for the drift versions of SNIPS and TOP. As can be seen in the "% drift" column of the train splits, the partial shift leads to around 2-3% of training data containing examples from the clusters that have been shifted into test. Our main objectives during the creation of the drift splits was to shift entire clusters into the splits and to match the intent distribution. We did not constrain the amount of examples in the test split that have been deliberately shifted into the test split, i.e., observe in Table 4 that the "% drift" on the validation and test splits varies from 23-79%.

## B TOPK-AUTOENCODER

In Algorithm 1 we present our proposed strategy to train a BERT-based SLU model (Chen et al., 2019) with TOPK-AE. In the following we will first describe the autoencoder and its optimization and then the training steps to train a model with TOPK-AE.

**Autoencoder.** The input to the autoencoder are the averaged token representation $X_{enc}$ each batch item. Let $X_{enc} \in \mathbb{R}^{b \times d}$, with $b$ being the batch size and $d$ the hidden size. Let the layers of the autoencoder be defined as $A_{enc} \in \mathbb{R}^{d \times c}$ and $A_{dec} \in \mathbb{R}^{c \times d}$ with $c$ being the size of the latent code, i.e. the number of latent groups. The autoencoder is then defined as:

$$\begin{aligned} H &= softmax(X_{enc}A_{enc}) \\ R &= HA_{dec} \end{aligned} \tag{1}$$

Notably, we employ a softmax in the bottleneck $H$ such that the auto-encoder's latent code is a distribution over $c$ latent groups. $R$ is the reconstruction of the autoencoder's input.

The auto-encoder is optimized with two losses:

i) a *reconstruction loss*, i.e. the cross entropy loss of the objective that each reconstruction $R_i$ should be closer to its original input $X_{enc_i}$ than to

| Intent Type | Slot Type | Train | Valid | Test | Test-Valid |
|---|---|---|---|---|---|
| DepartureTime | B-Criterion | 57 | 5 | 5 | 0 |
| | B-StationStart | 62 | 6 | 6 | 0 |
| | B-Vehicle | 61 | 5 | 6 | 1 |
| FindConnection | B-Criterion | 11 | 3 | 3 | 0 |
| | B-StationStart | 99 | 10 | 10 | 0 |
| | B-StationDest | 106 | 11 | 11 | 0 |

Table 8: The intent-slot distribution for an 80/10/10 train/valid/test splits from a Chatbot SLU dataset (shortened example)

**Algorithm 1** Training SLU with Online Auto-encoder DRO

$M$ is the main model (SLU) with two task losses (slot and intent)
$\theta_M$ are the main model's parameters
$\theta_{AE}$ are the auto-encoder's parameters
**for** data_batch (X, Y) in training_data **do**
  $X_{enc} = ENC(X)$
  $l_{slot}, l_{intent}$ = compute the task losses of $M$ on $(X, Y)$
  $l_{AE} = L_{Recon}(X_{enc}, \theta^t_{AE}) + \beta L_{Divers}(X_{enc}, \theta^t_{AE})$
  $\theta^{t+1}_{AE} = update(\theta^t_{AE}, \nabla L_{AE})$
  $\hat{l}_{slot}$ = compute group loss$(A^{t+1}_{enc}, X_{enc}, L_{slot})$
  $\hat{l}_{intent}$ = compute group loss$(A^{t+1}_{enc}, X_{enc}, L_{intent})$
  $\theta^{t+1}_{M} = update(\theta^t_{M}, \nabla(\hat{L}_{slot} + \hat{L}_{intent}))$
  $t = t + 1$
**end for**

other batch items in $X_{enc}$. $\theta_{AE}$ denote the auto-encoder's parameters:

$$L_{Recon}(X_{enc}, \theta_{AE}) = \\ -\frac{1}{b}\sum_{i=1}^{b} log(softmax(X_{enc}R^\top))_i \quad (2)$$

For regularization we apply dropout to $R$ before computing the reconstruction loss.

ii) a *diversity loss* to prevent the auto-encoder from collapsing into one mode, which is similar to the loss used in T-SNE (van der Maaten and Hinton, 2008).

$$\mathrm{L}_{Divers}(X_{enc}, \theta_{AE}) = \frac{\sum_{i \neq j} KL(H_i, H_j)}{b(b-1)} \quad (3)$$

where $\theta_{AE}$ denote the auto-encoder's parameters and $KL$ the Kullback-Leibler divergence.

This method adds the following hyper-parameters for the autoencoder: size $c$ of the autoencoders bottleneck, learning rate $\lambda_{AE}$ and weight decay $\alpha_{AE}$, and $\beta_{AE}$ a scalar for the reconstruction loss.

We considered the following TOPK-AE variants:

**TOPK-AE-PR** The bottleneck output of the autoencoder is $H \in [0, 1]^{b \times c}$, i.e. a distribution over $c$ groups for each batch item. Let $\circ$ denote elementwise multiplication along a matching mode. Then $\hat{H} = H \circ l_{task}$, i.e. $\hat{H} \in \mathbb{R}^{b \times c}$ are the losses weighted according to each latent group. Instead of averaging over all losses per latent group, $\hat{H}$ is truncated to the top-$k$ largest weighted losses per group, i.e. $\hat{H} \in \mathbb{R}^{k \times c}$ and then averaged per group to yield $\hat{l} \in \mathbb{R}^c$. The final batch loss is $max(\hat{l})$.

**TOPK-AE-BIN** Convert H into one-hot distributions, i.e. hard assignments to a latent group for each batch item, then proceed like in TOPK-AE-PR.

**Algorithm.** The model is finetuned for two task losses, one for the intent classification task and one for the sequence tagging classifier for the slot filling task. While it would be possible to use a separate autoencoder for each task, we found it beneficial to share one autoencoder for both task losses.

One update step is as follows: (i) For each batch during training, first the auto-encoder's parameters

are updated. (ii) Subsequently, we compute the group losses for the two SLU's tasks (i.e. slot and intent) based on $H$, i.e. the latent groups. This yields a vector $\hat{l}_{task}$ of loss-aggregations over the latent groups. $H \in \mathbb{R}^{b \times c}$ is a distribution over $c$ groups for each batch item. $\hat{l}_{task} = H^{\top} l_{task}$ is the vector of group losses in which each batch item is weighted according to the autoencoder's distribution over latent groups. In other words, each component in $\hat{l}$ contains the accumulated losses of all batch items that the autoencoder considers to be similar. Finally, we update the SLU model's parameters using the task group losses.

## C Hyperparameters

See Table 9 for a detailed list for all hyperparameters and their search range. The hyper-parameters that were tuned for all methods are the learning rate $\lambda$ and the intent loss scaler $\gamma$. Each optimization method can have additional hyper-parameters: GROUP-DRO (Sagawa et al., 2020) has a step size to compute the exponential average of group losses. As we discussed in Section 5.2 we observed overfitting of the GROUP-DRO method and not producing good results on many occasions. We did attempt to address this and added a geometric decay of the exponential average as an option in the hyperparameter search, which did help a little bit. TOPIC-CVAR (Oren et al., 2019) has the CVaR percentage and also a step size for the exponential average of losses. The batch size, weight decay, maximum number of epochs and the intent loss scaler (see 5.1) were determined in a prior larger hyperparameter search. We did not find a lot of variance for their preferred setting, also not in interplay with the other DRO methods, which is why we fixed them to save computation from this point on. See Section 4.2 and 4.3 for the hyperparameters of TOPK-AE and TOPK-AE-PR/BIN respectively. Anecdotally the hyper-parameter $k$ determining the topk losses in their objective which was tuned for TOPK and TOPK-AE-PR/BIN typically ended up in the lower regions of the range, i.e. between $2-8$.

## D Results

In the following tables we report the numerical results for the experiments from Section 5 with the metrics reported in Section 5.1. Additional metrics we report here is the macro average over intents, i.e. "MA INT. COMBINED", and SEMER (semantic error rate) - a metric which is defined as follows:

$$\textit{SEMER} = \frac{\#(\text{slot+intent errors})}{\#\text{slots in reference + 1}} \quad (4)$$

The columns containing a "%" indicate relative change to ERM.

| Optimization | description | name | type | range |
|---|---|---|---|---|
| ERM + all | learning rate | $\lambda$ | loguniform | 1.e-5 - 1.e-4 |
| | intent loss scaler | $\gamma$ | loguniform | 1.e-2 - 10.0 |
| | batch size | | fixed | 32 |
| | weight decay | | fixed | SNIPS: 0.02, TOP: 0.002 |
| | max epochs | | fixed | 100 |
| | max warmup steps | | fixed | 0 |
| TOPIC-CVAR | alpha | | uniform | 1.e-4 - 0.5 |
| | gamma | | loguniform | 1.e-4 - 0.5 |
| GROUP-DRO | step size | | loguniform | 1.e-4 - 1. |
| | geometric decay | | categorical | True, False |
| TOPK | topk | $k$ | logint | 1 - 16 |
| TOPK-AE-PR/BIN | topk | $k$ | logint | 2 - 16 |
| | ae learning rate | $\lambda_{AE}$ | loguniform | 1.e-4 - 1.e-3 |
| | ae cluster loss weight | $\beta_{AE}$ | loguniform | 1.e-1 - 1.0 |
| | ae cluster size | $c$ | int | 128, 256, 512 |
| TOPK-GROUP | topk | $k$ | logint | 2 - 16 |

Table 9: Hyperparameters for the different optimization methods used in the experiments.

| | full shift | | | | partial shift | | | |
|---|---|---|---|---|---|---|---|---|
| | slot value | | slot context | | slot value | | slot context | |
| | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 |
| ATIS | -0.3 | -0.7 | -1.7 | 0 | -0.3 | -1.3 | -0.7 | -0.1 |
| SNIPS | -0.2 | -3.7 | 0.0 | -5.2 | -0.1 | -3.2 | -0.2 | -3.9 |
| MIT | -0.2 | -2.0 | -2.6 | -2.9 | 0 | -2.2 | -3.2 | -3.4 |
| TOP | -1.1 | -2.7 | -1.2 | -5.2 | -1.2 | -2.6 | -0.8 | -3.7 |

Table 10: Drop in performance from validation to test for *ERM* on four of the datasets with different drift types (slot value, slot context) and drift percentage (full, partial).

| | COMBINED | | SIGN. 0.05 ERM | | SEMER | | MA INT. COMBINED | |
|---|---|---|---|---|---|---|---|---|
| | | % | < | > | | % | | % |
| GROUP-DRO | 91.8 | -0.6 | 6 | 1 | 11.8 | 4.8 | 85.7 | -1.3 |
| TOPK-AE-PR | 91.9 | -0.4 | 5 | 1 | 11.7 | 3.4 | 86.7 | -0.1 |
| TOPIC-CVAR | 92.1 | -0.2 | 4 | 0 | 11.5 | 2.0 | 87.1 | 0.3 |
| ERM | 92.3 | - | - | - | 11.3 | - | 86.8 | - |
| TOPK | 92.5 | 0.1 | 0 | 3 | 11.0 | -2.7 | 87.2 | 0.5 |
| TOPK-AE-BIN | 92.5 | 0.2 | 0 | 3 | 11.0 | -2.7 | 87.5 | 0.8 |
| TOPK-GROUP | 92.5 | 0.2 | 0 | 5 | 11.0 | -2.4 | 87.5 | 0.8 |

Table 11: Results for TOP, averaged over all eight scenarios. Are the DRO methods more robust against the drifts than ERM? Which DRO method is the most effective? The columns containing a "%" indicate relative change to ERM.

| | COMBINED | | SIGN. 0.05 ERM | | SEMER | | MA INT. COMBINED | |
|---|---|---|---|---|---|---|---|---|
| | | % | < | > | | % | | % |
| GROUP-DRO | 96.2 | -0.2 | 5 | 0 | 7.1 | 7.8 | 96.0 | -0.2 |
| TOPIC-CVAR | 96.4 | 0.0 | 2 | 1 | 6.6 | 1.2 | 96.2 | 0.0 |
| ERM | 96.4 | - | - | - | 6.6 | - | 96.2 | - |
| TOPK-AE-PR | 96.5 | 0.0 | 1 | 2 | 6.6 | -0.1 | 96.3 | 0.0 |
| TOPK | 96.6 | 0.1 | 0 | 4 | 6.3 | -3.7 | 96.4 | 0.2 |
| TOPK-AE-BIN | 96.7 | 0.2 | 1 | 2 | 6.2 | -5.4 | 96.5 | 0.2 |
| TOPK-GROUP | 96.7 | 0.2 | 0 | 4 | 6.2 | -6.0 | 96.5 | 0.3 |

Table 12: Results for SNIPS, averaged over all eight settings. Are the DRO methods more robust against the drifts than ERM? Which DRO method is the most effective? The columns containing a "%" indicate relative change to ERM.

| | validation i.i.d. | | | | | validation o.o.d. | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 0.05 ERM | | | | | 0.05 ERM | |
| | % CMB. | % SEM. | < | > | | % CMB. | % SEM. | < | > |
| GROUP-DRO | -0.2 | 5.8 | 2 | 0 | GROUP-DRO | -0.3 | 9.7 | 3 | 0 |
| ERM | - | - | - | - | TOPK-AE-PR | -0.1 | 5.4 | 1 | 0 |
| TOPK-AE-BIN | 0.1 | -2.6 | 1 | 1 | TOPIC-CVAR | -0.1 | 3.7 | 2 | 0 |
| TOPIC-CVAR | 0.1 | -1.2 | 0 | 1 | ERM | - | - | - | - |
| TOPK-AE-PR | 0.2 | -5.5 | 0 | 2 | TOPK | 0.1 | -1.5 | 0 | 1 |
| TOPK | 0.2 | -5.8 | 0 | 3 | TOPK-GR-DRO | 0.2 | -5.6 | 0 | 2 |
| TOPK-GR-DRO | 0.3 | -6.4 | 0 | 2 | TOPK-AE-BIN | 0.4 | -8.3 | 0 | 1 |

Table 13: Results on SNIPS. Comparing validation i.i.d. with validation o.o.d.. This influences hyperparameter optimization and early stopping. The columns containing a "%" indicate relative change to ERM.

| | full drift | | | | | partial drift | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 0.05 ERM | | | | | 0.05 ERM | |
| | % CMB. | % SEM. | < | > | | % CMB. | % SEM. | < | > |
| GROUP-DRO | -0.2 | 5.2 | 1 | 0 | GROUP-DRO | -0.3 | 10.6 | 4 | 0 |
| ERM | - | - | - | - | TOPIC-CVAR | -0.1 | 4.2 | 2 | 0 |
| TOPK-AE-PR | 0.1 | -1.4 | 1 | 1 | TOPK-AE-PR | 0 | 1.4 | 0 | 1 |
| TOPIC-CVAR | 0.1 | -1.5 | 0 | 1 | ERM | - | - | - | - |
| TOPK | 0.2 | -3.9 | 0 | 3 | TOPK | 0.1 | -3.4 | 0 | 1 |
| TOPK-AE-BIN | 0.2 | -5.3 | 0 | 2 | TOPK-GR-DRO | 0.2 | -5.3 | 0 | 2 |
| TOPK-GR-DRO | 0.3 | -6.7 | 0 | 2 | TOPK-AE-BIN | 0.2 | -5.6 | 1 | 0 |

Table 14: Results on SNIPS. Comparing full vs partial drift on SNIPS. The partial drift means only 90% of the examples per cluster are shifted into testing. The columns containing a "%" indicate relative change to ERM.