Supplementary Materials : Appendix

A Implementation

We provide an example pytorch implementation based on the `fairseq` code base in Figure 1-4, which consists of: (1) sampling $\lambda \sim \text{Beta}(\alpha, \alpha)$, (2) computing expected source sentence in the encoder forward function (3) computing expected previous target token in the decoder forward function, (4) computing cross entropy loss on expected target sentence. Our code can be found at: https://anonymous.4open.science/r/388f9f1c-aeee-4d84-8ac0-6d7b2391f3ab/

B Dataset Description

B.1 Machine Translation

There are 160k training examples for IWSLT' 14 German-English, and 176k, 174k training examples for IWSLT '14 English-Spanish and English-Italian datasets respectively.[1] The lager WMT' 14 English-German dataset consists of 4.5M training examples.[2] For IWSLT German-English, we use the standard test set: concatenation of TED tst2010, tst2011, tst2012, dev2010, and dev2012. For other IWSLT datasets, we use TED dev2010 for validation and tst2010, tst2011, tst2012 for test. On WMT, following previous work (Ott et al., 2018), we use WMT '16 as training dataset, and newstest2014 as test set. Following common practice, for IWSLT '14, we lowercase all words. All sentences are preprocessed with byte-pair-encdoding (BPE) (Sennrich et al., 2016). For IWSLT, we use a joint source and target vocabulary with 10k BPE types. For WMT, we built a BPE vocabulary with 40k types.

[1] https://wit3.fbk.eu/mt.php?release=2014-01

[2] https://drive.google.com/uc?export=download&id=0B_bZck-ksdkpM25jRUN2X2UxMm8

B.2 SCAN

We experiment with three different splits of the SCAN dataset [3]: `jump`, `turn left`, `around right`. There are approximately 14k, 21k, 15k training examples and 7k, 1k, 4k test examples respectively for the `jump`, `turn left` and `around right` splits. We provide illustrative examples for SCAN in Table 1 and Table 2.

In general, the `turn left` split is the easiest because the corresponding instruction token *turn left* (i.e. LTURN) is still seen in training (e.g. in the context of *walk left*). The `around right` split is the hardest because the models must learn how to compositionally apply *around* function to *right* even though they are not seen together in training.

Split	Train Command	Test Command
`jump`	*jump; turn left twice after look*	*turn left twice after jump; run and jump*
`turn left`	*turn left; run opposite left;*	*walk and turn left thrice*
`around right`	*jump around left; turn opposite right twice*	*walk around right; look around right and jump left*

Table 1: Example commands from the different SCAN dataset splits.

Command	Instruction
turn around right and jump	I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_TURN_RIGHT I_JUMP
jump left and turn left	I_TURN_LEFT I_TURN_LEFT I_JUMP

Table 2: Input-output examples for the SCAN dataset.

[3] https://github.com/brendenlake/SCAN

B.3 Semantic Parsing

We study two versions of the `GeoQuery` dataset:[4] the `query` and `question` splits of SQL queries. There are 880 English questions about United States geography, paired with SQL queries. See dataset examples provided by Andreas (2020) in Figure 5. We follow Andreas (2020) for dataset split and preprocessing steps. The standard train-test split `question` ensures that there are no natural language questions repeated between the train and test sets. This standard split is limited to test generalization (Andreas, 2020; Finegan-Dollak et al., 2018) because many test examples still have the same logical forms as some of the training examples. The more difficult `query` split ensures that neither question or logical forms (after anonymizing named entities) are repeated.

C Experimental Details

C.1 Machine Translation

Our codebase is based on fairseq (Ott et al., 2019). For IWSLT experiments, we use a 6-layer Transformer with 4 attention heads, embedding size 512 and FFN layer dimension 1024 (model configuration *transformer_iwslt_de_en*). For WMT experiments, we use a standard Transformer Base (Vaswani et al., 2017), with 8 attention heads, embedding size 512 and FFN layer dimension 2048 (model configuration *transformer_wmt_en_de*). We use `multi-bleu.pl`[5] for BLEU evaluation.

For IWSLT, we first tune the Transformer baselines (i.e. considering learning rate: $\{1e^{-4}, 5e^{-4}\}$, max tokens per batch: $\{4096, 5120\}$, label smoothing: $\{0.0, 0.1, 0.2\}$, dropout: $\{0.3, 0.4\}$, weight decay: $\{1e^{-4}, 5e^{-4}\}$). Then, to compare different methods, we fix these hyper-parameters and tune the method-specific hyper-parameters (i.e. considering word drop out rate: $\{0.1, 0.2, 0.3\}$, switchout rate: $\{0.4, 0.8, 1.2\}$, and SeqMix α parameter in range $[0.1, 1.5]$) on the validation set. We use Adam optimizer with Beta $(0.9, 0.98)$, and inverse square root learning rate scheduler with initial learning rate 5e-4 and $4,000$ warmup updates. We use cross entropy loss with label smoothing rate 0.1 and max tokens 4096 per batch. For IWSLT '14 (de↔en), we use dropout 0.4,[6] weight decay $5e^{-4}$, and beam search decoding with beam size 15. For IWLST '14 (en→{it, es}), we use dropout 0.3 and weight decay $1e^{-4}$, and beam search decoding with beam size 5. For all SeqMix experiments, we sample (X', Y') uniformly from all examples with similar target Y' lengths ($|Y| - |Y'| \leq 5$). We train all models until convergence. In practice, it takes around 10 to 15 hours on a single Tesla P100 GPU.

For WMT '14, we use Adam optimizer with Beta $(0.9, 0.98)$, and inverse square root learning rate scheduler with initial learning rate 0.0007, minimum learning rate $1e^{-9}$ and $8,000$. warmup updates. We use cross entropy loss with label smoothing rate 0.1 and max tokens per batch 3072 on per GPU (4 GPUs in total). We use dropout 0.1 but no weight decay nor norm clipping. Training takes approximately 4 days on 4 Tesla P100 GPUs. Due to its computational overhead, we only experiment with SeqMix alpha 0.1, SwitchOut 0.8 and word dropout 0.1, which did well based on our IWSLT experiments. We average the last 5 epoch checkpoints, and use beam search decoding with beam size 4 and length penalty 0.6. We sample (X', Y') by shuffling the batch.

C.2 SCAN

For all SCAN experiments, following Andreas (2020), we train a one-layer LSTM encoder-decoder model with embedding size of 64, hidden size of 512, a bidirectional encoder and an attentional decoder. We train the model with the Adam optimizer with Beta $(0.9, 0.98)$, and clip the gradient norm at 1.0. We use learning rate 0.001 and reduce the learning rate on plateau (shrink rate 0.5). We use batch size 64, dropout 0.5, but no weight decay nor label smoothing. We use greedy decoding and use accuracy as evaluation metric. Note that a predicted output is correct if and only if it exactly matches with the ground truth. We tune the method-specific hyper-parameters: we consider word drop out rate $\{0.1, 0.2, 0.3\}$, SwitchOut rate $\{0.4, 0.8, 1.2\}$, SeqMix $\alpha \in \{0.1, 0.5, 1.0\}$ for both hard and soft variant. We report the best test accuracy for each method. The best hyperparameter for SeqMix was: (without GECA) 0.5, 1.0, 0.1 and (with GECA) 1.0, 1.0, 1.0 respectively for `jump`, `around right`, `turn left`. For SeqMix, we approximate sampling (X', Y') by shuffling the current batch, which are already padded to the same length. For experiments without GECA, if the sequence length of (X', Y')

[4] https://github.com/jkkummerfeld/text2sql-data

[5] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

[6] Note that dropout here is different from word dropout.

is much shorter than (X, Y), we additionally repeat (X', Y') to the same length (to ensure *jump* will be mixed up with tokens at different positions). We train for 50 epochs for runs without GECA and 150 epochs for runs with GECA. Training takes approximately 30 minutes to 1 hour on single Tesla P100 GPU.

C.3 Semantic Parsing

For all semantic parsing experiments, we train a one-layer LSTM encoder-decoder model with an embedding size of 64, a hidden size of 512, a bidirectional encoder and an attentional decoder. Following Andreas (2020), we additionally introduce a copy mechanism (Finegan-Dollak et al., 2018). For SeqMix, like for SCAN, we approximate sampling (X', Y') by shuffling the current batch, and use (exact) accuracy as evaluation metric. The final best optimization setting for SeqMix with GECA is α 0.6, batch size 32, no dropout nor weight decay, learning rate $1e^{-3}$ and learning rate shrink ratio of 0.7 after plateau. The best hyperparameters without GECA are the same as with GECA except we use dropout 0.4. For all experiments, we train for 180 epochs, and report the test accuracy based on last checkpoint. As experimental results on semantic parsing have relatively high variance, we report the average accuracy across 10 different seeds. Training in total takes approximately 15 hours on single P100 GPU. We conduct t-test (two-tailed distribution, two-sample unequal variance heteroscedastic test) to compare SeqMix (soft) with baseline (with and without GECA).

References

Jacob Andreas. 2020. Good-Enough Compositional Data Augmentation. In Proceedings ACL.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology.

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

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. pages 1–9.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh neural machine translation systems for WMT 16. In Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc.

```python
from torch.distributions.beta import Beta
def get_lambda(self,
        batch_size):
    """
        Sample lambda given batch size.
    """
    dist = Beta(self.args.alpha, self.args.alpha)
    lambda_ = dist.sample(sample_shape=[bsz]).to("cuda")
    lambda_ = torch.max(lambda_, 1 - lambda_)
    return lambda_
```

Figure 1: Example code snippet to sample lambda

```python
def encoder_forward(self,
        lambda_,
        src_tokens_a,
        src_lengths_a,
        src_tokens_b,
        src_lengths_b,
        ...):
    """
        Args:
            lambda_ (FloatTensor): lambda used to permute sentences of shape `(batch)`
            src_tokens_a (LongTensor): tokens in the source sentence X of shape `(batch, src_len)`
            src_lengths_a (LongTensor): lengths of each source sentence X of shape `(batch)`
            src_tokens_b (LongTensor): tokens in the source sentence X' of shape `(batch, src_len)`
            src_lengths_b (LongTensor): lengths of each source sentence X' of shape `(batch)`
    """

        if self.layer_wise_attention:
            return_all_hiddens = True

        xa, encoder_embedding_a = self.forward_embedding(src_tokens_a)
        xb, encoder_embedding_b = self.forward_embedding(src_tokens_b)

        x = xa * lambda_.reshape(-1, 1, 1) + xb * (1-lambda_).reshape(-1, 1, 1)
        encoder_embedding = encoder_embedding_a * lambda_.reshape(-1, 1, 1) + \
                encoder_embedding_b * (1 - lambda_).reshape(-1, 1, 1)


        # B x T x C -> T x B x C
        x = x.transpose(0, 1)

        # compute padding mask
        encoder_padding_mask = src_tokens_a.eq(self.padding_idx)

        ......
```

Figure 2: Example code snippet of encoder forward function

```
def decoder_extract_features(self,
    lambda_,
    prev_output_tokens_a,
    perv_output_tokens_b,
    ...):

    """
        Args:
            lambda_ (FloatTensor): lambda used to permute sentences of shape `(batch)`
            prev_output_tokens_a (LongTensors): previous decoder outputs of target sentence Y of shape `(batch, tgt_len)`
            prev_output_tokens_b (LongTensors): previous decoder outputs of target sentence Y' of shape `(batch, tgt_len)`
    """

    def get_embedding(prev_output_tokens):
        ......

    xa = get_embedding(prev_output_tokens_a)
    xb = get_embedding(prev_output_tokens_b)

    x = xa * lambda_.view(-1, 1, 1) + xb * (1-lambda_).view(-1, 1, 1)

    ......
```

Figure 3: Example code snippet of decoder extract features function, as part of the forward function

```
def labeled_smooth_cross_entropy_forward(self,
        model,
        sample,
        lambda_,
        reduce=True):

    net_output = model(lambda_,
                    sample["net_input_a"]["src_tokens"],
                    sample["net_input_a"]["src_lengths"],
                    sample["net_input_b"]["src_tokens"],
                    sample["net_input_b"]["src_lengths"],
                    sample["net_input_a"]["prev_output_tokens"],
                    prev_output_tokens_b=sample["net_input_b"]["prev_output_tokens"])
    lprobs = model.get_normalized_probs(net_output, log_probs=True)
    lprobs = lprobs.view(-1, lprobs.size(-1))
    loss_a, nll_loss_a = label_smoothed_nll_loss(
            lprobs, sample["target_a"].view(-1,1), self.eps, ignore_index=self.padding_idx, reduce=False,
    )
    loss_b, nll_loss_b = label_smoothed_nll_loss(
            lprobs, sample["target_b"].view(-1,1), self.eps, ignore_index=self.padding_idx, reduce=False,
    )

    bsz, slen = sample["target_a"].size()
    loss_a = loss_a.reshape(bsz, slen)
    nll_loss_a = nll_loss_a.reshape(bsz, slen)
    loss_b = loss_b.reshape(bsz, slen)
    nll_loss_b = nll_loss_b.reshape(bsz, slen)

    loss = loss_a * lambda_.view(-1, 1) + loss_b * (1-lambda_).view(-1, 1)
    nll_loss = nll_loss_a * lambda_.view(-1, 1) + nll_loss_b * (1-lambda_).view(-1, 1)
    valid_indices = (sample["target_a"] != self.padding_idx)
    loss = loss * valid_indices.float()
    nll_loss = nll_loss * valid_indices.float()

    if reduce:
        loss = loss.sum()
        nll_loss = nll_loss.sum()

    ......
```

Figure 4: Example code snippet of loss function

Logical forms

what is the lowest point in rhode island

```
( A , lowest ( A , ( place ( A ) , loc ( A , B ) , const ( B , stateid ( rhode island ) ) ) ) )
```

what states does the florida run through

```
( A , ( state ( A ) , const ( B , riverid ( florida ) ) , traverse ( B , A ) ) )
```

what state borders the state with the lowest population density

```
( A , ( state ( A ) , next_to ( A , B ) , smallest ( C , ( state ( B ) , density ( B , C ) ) ) ) )
```

SQL queries

what rivers run through west wyoming

```
SELECT RIVER0.NAME FROM RIVER AS RIVER0 WHERE RIVER0.TRAVERSE = " west wyoming "
```

which states have towns major named springfield

```
SELECT CITY0.STATE_NAME FROM CITY AS CITY0 WHERE CITY0.NAME = " springfield " AND CITY0.POP > 150000
```

what is the population of the area of the largest state

```
SELECT CITY0.POP FROM CITY AS CITY0 WHERE CITY0.NAME = ( SELECT STATE0.AREA FROM STATE AS STATE0
WHERE STATE0.AREA = ( SELECT MAX ( STATE1.AREA ) FROM STATE AS STATE1 ) )
```

Figure 5: GeoQuery SQL Queries data examples provided by Andreas et al (Andreas, 2020)