MAD for Robust Reinforcement Learning in Machine Translation

Anonymous ACL submission

Abstract

We introduce a new distributed policy gradient algorithm and show that it outperforms existing reward-aware training procedures such as REINFORCE, minimum risk training (MRT) and proximal policy optimization (PPO) in terms of convergence speed and stability, and overall performance at optimising machine translation models. Our algorithm, which we call MAD (on account of using the *mean absolute deviation* in the importance weighting calculation), has distributed data generators sampling multiple candidates per source sentence on worker nodes, while a central learner updates the policy. MAD depends crucially on two variance reduction strategies: (1) a new robust importance weighting scheme that encourages learning from examples that are not too likely or unlikely relative to the current policy and (2) by learning from balanced numbers of high- and low-reward training examples. Additionally, our algorithm has few hyperparameters, making it easy to use on new tasks with little or no adaptation. Experiments on a variety of translation tasks show the policies learned with MAD perform very well with both greedy decoding and beam search, and the learned policies are sensitive to the specific reward used during training.

1 Introduction

There is increasing interest in fine-tuning conditional language models on the basis of feedback from task-specific reward models or similarity functions that compare to human-generated reference outputs rather than relying exclusively on supervised learning (Stiennon et al., 2020; Ziegler et al., 2019; Wu et al., 2018; Paulus et al., 2018; Rennie et al., 2017; Ranzato et al., 2016). Maximising sequence level rewards has several advantages. First, it avoids the apparent conflict between the intuitive importance of "getting the full sequence right" in generation problems and the more conventional token-level cross entropy loss. Second, since a policy trained to maximize rewards is supervised with its own outputs—both good and bad ones—it mitigates issues arising from "exposure bias," in which a learned policy that has been trained only on correct examples has no experience recovering from errors and therefore performs poorly at test time (Ranzato et al., 2016). Third, feedback from (learned) rewards can be a cost-effective strategy for incorporating human preferences about how a systems should behave (Stiennon et al., 2020; Christiano et al., 2017).

Unfortunately, fine-tuning policies for generating in complex output spaces, such as language, on the basis of sparse rewards is challenging. On one hand, estimating and debugging reliable auxiliary critic/value functions that are needed by many learning algorithms is challenging (Wu et al., 2018; Bahdanau et al., 2017; Nguyen et al., 2017), and commonly used average or batch-level reward baselines (Kreutzer et al., 2017) are poor variance reducers since they are independent of the input, and input difficulty is a strong determinant of reward magnitude.

In this paper, we propose a new distributed policy gradient algorithm (§2) for fine-tuning translation models that addresses these issues. The distributed setup lets us use modest computation to obtain simple and effective empirical reward baselines (Rennie et al., 2017) rather than using inappropriate batch-level statistics or relying on brittle auxiliary value models. Our proposed algorithm has two components designed to make learning from the reward signal more effective: an importance weighting strategy that encourages the algorithm to pay attention to trajectories that are near to the current policy (i.e., it up-weights trajectories that are neither "too likely" nor "too unlikely"), and, second, an instance selection strategy that encourages batches to contain a mix of both positive and negative rewards. Thus, our algorithm learns from trajectories that are *already* relatively likely

under the current policy (meaning any updates to the policy will be relatively small), while also learning preferentially from instances that have a large impact on the reward metric. This enables the algorithm to make large improvements in reward while taking small, conservative (and therefore less risky) steps to change the behaviour.

These changes reduce the policy gradient variance and enable the model to improve over many training steps without divergence. This results in policies that produce high-quality translations, even when using greedy decoding. In our main experiments (§3), we use sentence BLEU as the reward and find that the average improvement on held-out test sets over the initial cross entropy trained model is 1.8 BLEU. Additionally, we carry out a careful empirical analysis (§4) that shows what impact that the various components of the algorithm have, and also that the algorithm learns different policies depending on the reward function used during optimisation, with the resulting policies showing reward-specific improvements on held-out data.

2 Algorithm

Our algorithm consists of workers generating training trajectories and rewards, in parallel, from a slightly out-of-date copy of the policy, and a central learner that updates the policy. The data generation algorithm is shown Alg. 1 and the learner in Alg. 2. The learning algorithm has four core components: sampling from a range of temperatures (§2.1), conditional reward normalization on the basis of empirical means and variances (§2.2), a novel robust importance weighting strategy that focuses learning efforts on samples that are neither "too likely" nor "too unlikely" under the current policy (§2.3), and a sample selection strategy that favours extreme values of the empirical reward distribution (§2.4). We discuss each of these components in turn.

2.1 Multi-Temperature Sampling

To obtain suitably diverse candidates to learn from, it is conventional to add a temperature hyperparameter used to generate samples (Shen et al., 2016; Papini et al., 2020). We identify two problems with this. First, there is the practical matter of needing to select a temperature in order to obtain good performance. Second, it is widely observed that policy gradient algorithms result in increasingly peaked distributions as training progresses (Kiegeland and Kreutzer, 2021; Choshen et al., 2020; Rennie et al., 2017), meaning that the amount of "exploration" being considered by the algorithm decreases over time. While some RL tasks are interested in maximising total accumulated returns over time (meaning that "exploitation" is important), we rather seek to learn a policy that is capable of behaving as intelligently as possible in states that are not encountered during a training phase, and therefore, we seek an exploration-heavy sampling strategy. To avoid the difficulties with drifting entropy, we use a simple approach of generating populations of samples at several different temperatures.

Concretely our data generation algorithm (lines 5–7) begins by sampling N_{pop} translations for the current policy and source sentence $\boldsymbol{x}$. Each sample is generated using a different temperature that is determined by finding *equally spaced values*[1] between the interval $[T_{min}, T_{max}]$. Duplicate translations are removed. The process we use is:

$$\mathcal{Y}_{\boldsymbol{x}} = \textsc{Unique}(\{\textsc{Sample}(\boldsymbol{x}, \boldsymbol{\theta}, t);\ t \in T\})$$

where

$$T = \{T_{min} + \delta_t \cdot (i-1);\ i \in \{1, \ldots, N_{pop}\}\}$$

$$\delta_t = \frac{T_{max} - T_{min}}{N_{pop} - 1}.$$

If the number of translations is less than the number of training examples we intend to select, $|\mathcal{Y}_{\boldsymbol{x}}| < N$ (where N is the number of training trajectories we will train on, discussed below in §2.4), we discard $\mathcal{Y}_{\boldsymbol{x}}$ and move on to the next source sentence. In this way, we will only train on examples where suitable diversity exists, and we always obtain the same number of trajectories for a source sentence.

2.2 Conditional Reward Normalization

Reward normalization is a well known variance reduction technique for policy gradient methods, with the simplest version being to subtract a constant from the reward (Williams, 1992). Other methods have been proposed such as subtracting a model *baseline reward* (Weaver and Tao, 2001), the running average of the rewards seen during training (Kreutzer et al., 2017), or z-scoring the rewards in a training batch (Stiennon et al., 2020). As demonstrated by Kiegeland and Kreutzer (2021), using the running reward average b helps to reduce variance when REINFORCE is applied to translation.

While empirically effective, these baselines explain less reward variation than we might hope. We

[1] https://numpy.org/doc/stable/reference/generated/numpy.linspace.html

$$\mathcal{L}_{\text{REINFORCE}} = (\Delta(\boldsymbol{y}, \boldsymbol{y}_{ref}) - b) \cdot \log p(\boldsymbol{y} \mid \boldsymbol{x})$$
$$\mathcal{L}_{\text{PPO}} = \min\{u \cdot \widetilde{r},\ \text{clip}(u, 1-\epsilon, 1+\epsilon) \cdot \widetilde{r}\}$$
$$\mathcal{L}_{\text{MRT}} = \sum_{\boldsymbol{y} \in \mathcal{Y}_x} \Delta(\boldsymbol{y}, \boldsymbol{y}_{ref}) \cdot \frac{p(\boldsymbol{y} \mid \boldsymbol{x})}{\sum_{\boldsymbol{y}' \in \mathcal{Y}_x} p(\boldsymbol{y}' \mid \boldsymbol{x})}$$
$$\mathcal{L}_{-\text{MAD}} = \min\{u,\ 2\} \cdot \overline{r} \cdot \log p(\boldsymbol{y} \mid \boldsymbol{x})$$
$$\mathcal{L}_{\text{MAD}} = \min\{u \cdot v,\ 2\} \cdot \overline{r} \cdot \log p(\boldsymbol{y} \mid \boldsymbol{x})$$

Figure 1: Sequence level losses used by the algorithms evaluated in this paper. See §2 for notation.

note that the difficulty of generating a good translation is strongly dependent on the intrinsic difficulty of the source sentence, not merely the current policy. Since the usual reward normalization methods do not take this dependency into account, this results in a bias toward giving difficult sentences negative rewards and easier sentences positive rewards, leading to less stable learning.

We therefore use a standardization method that is conditioned on the source sentence. We take the set of translations for source sentence $\boldsymbol{x}$ and obtain a vector of rewards $r_i = \Delta(\boldsymbol{y}_i, \boldsymbol{y}_{ref})\ \forall\, i \in [1, |\mathcal{Y}_{\boldsymbol{x}}|]$ where $\Delta(\boldsymbol{y}, \boldsymbol{y}_{ref})$ is a scalar valued reward indicating how well $\boldsymbol{y}$ communicates the contents of the reference translation $\boldsymbol{y}_{ref}$.[2] We then standardize these rewards by removing the mean and dividing by the standard deviation (lines 11–18),

$$\overline{r}_i = (r_i - \mu_r)/\sigma_r\ \forall\, i \in [1, |\mathcal{Y}_{\boldsymbol{x}}|],\ \text{where}$$
$$\mu_r = \frac{1}{|\mathcal{Y}_{\boldsymbol{x}}|}\sum r_i, \quad \sigma_r = \sqrt{\frac{1}{|\mathcal{Y}_{\boldsymbol{x}}|}\sum (r_i - \mu_r)^2}.$$

This ensures that every source sentence, irrespective of its translation difficulty, has examples with positive and negative rewards.

In contrast, the standard reward used for the PPO algorithm is the z-scored reward of the training batch, $\widetilde{r}_i = (r_i - \widetilde{\mu}_r)/\widetilde{\sigma}_r\ \forall\, i \in [1, |\mathcal{B}|]$, where $\mathcal{B}$ is the training batch with randomly sampled $(\boldsymbol{x}, \boldsymbol{y})$ examples and $\widetilde{\mu}_r$ and $\widetilde{\sigma}_r$ are respectively the mean and standard deviation of the rewards in $\mathcal{B}$.

2.3 MAD Importance Weights

A key feature of our algorithm—and the one that gives it its name—is the use of a new importance weighting scheme for deciding which sampled trajectories are most valuable for updating the policy and which others should be downweighted. For trajectory i, our importance sampling correction w_i (Eq. 1) consists of two terms: u_i, which is the standard importance weighting ratio (Precup et al., 2000) to deal with the fact that in a distributed setup, data is generated from a stale policy:

$$u_i = \exp(p_i - q_i),\ \text{where}$$
$$p_i = \log p(\boldsymbol{y}_i \mid \boldsymbol{x}; \boldsymbol{\theta}), \quad q_i = \log p(\boldsymbol{y}_i \mid \boldsymbol{x}; \boldsymbol{\theta}_{old}).$$

The second term in Eq. 1, v_i, encourages the learner to pay attention to samples that are "relatively likely" under the current policy (as approximated by q, since we want the calculation to be carried out on the worker nodes, and p is only available on the central learner). We operationalize the notion of "relatively likely" as something that is near to the median[3] probability under q of the elements in $\mathcal{Y}_{\boldsymbol{x}}$, using the exponentiated negative *median absolute deviation* (MAD; lines 13–18):

$$v_i = \exp(-|q_i - \tilde{\mu}_q|/\tilde{\sigma}_q)$$
$$\tilde{\mu}_q = \textsc{Median}(\mathbf{q})$$
$$\tilde{\sigma}_q = \textsc{Median}(|\mathbf{q} - \tilde{\mu}_q|).$$

Why do we want to concentrate learning on these relatively likely trajectories, rather than perhaps other trajectories that have even higher (or lower) rewards? First, this is a strategy for reducing the gradient norms, since gradients associated with smaller changes to the policy will require, on average, less significant updates to the network. This is advantageous since smaller gradient norms can lead to better generalisation error (Kreutzer et al., 2017). Second, Choshen et al. (2020) provide evidence that RL algorithms can make only relatively modest changes to the starting policies. Therefore, we focus learning effort on instances which are "in

[2] Except for §4.4, we use sentence BLEU.

[3] The median was chosen because $\mathbf{q}$ can have a long tail. It is not uncommon for $\tilde{\mu}_q \approx -6$ and $\min(\mathbf{q}) < -1000$, see Figure 2. Comparisons to alternative definitions of v_i are reported in Appendix D.

reach" of the current policy. Finally, the strategy of selecting targets on the basis of being close to a current policy in online reward-driven learning has been proposed previously for linear translation models, and found to be more effective than other sample selection methods (Chiang, 2012). However, rather than doing instance selection on the basis of a combined objective as was done in that work, we downweight outliers with v_i.

To compute the final importance weight, we multiply the two terms and cap the maximum value:

$$w_i = \min\{u_i \cdot v_i,\ 2\}. \quad (1)$$

Truncating importance weights is a standard variance reduction strategy (Cortes et al., 2010; Schulman et al., 2017).

2.4 Sample Selection

One reason we generate multiple samples for a given source sentence is to compute the sample level statistics described above. However, a further reason is that we can subselect from these samples to further reduce variance. For our experiments, we select $N = 12$ samples from the pool by taking the 6 highest reward and 6 lowest reward examples (lines 19–22). This encourages a roughly equal mix of positive and negative rewards to be used for training, and it keeps the mean reward for the selected examples near *zero*, which helps to reduce gradient norms and variance (see Figure 3). Intuitively, this mix of examples encourages the learner to reallocate input-conditional probability mass from trajectories with poor rewards to trajectories with good rewards.

Using 33% of the total samples per source sentence provided a reasonable trade off between training speed and fresh examples. Training on all of the samples speeds up training but does not produce as high of a BLEU score as using only a subset of them (§4.3).

Algorithm 1 Asynchronous data generator

1: **function** GENERATE(N, N_{pop}, Δ, T_{min}, T_{max})
2: **while** True **do**
3: $\boldsymbol{\theta}_{old} \leftarrow \boldsymbol{\theta}$ ▷ *Get current global weights*
4: $(\boldsymbol{x}, \boldsymbol{y}_{ref}) \sim \mathcal{D}_{train}$
5: **for** $i \in [1, N_{pop}]$ **do** ▷ *Obtain* $\mathcal{Y}_{\boldsymbol{x}}$, $\mathbf{q}$, *and* $\mathbf{r}$
6: $T \leftarrow T_{min} + (i-1) \times (T_{max} - T_{min})/(N_{pop} - 1)$
7: $\boldsymbol{y}_i \leftarrow \text{SAMPLE}(\boldsymbol{x}, \boldsymbol{\theta}_{old}, T)$
8: $q_i \leftarrow \log p(\boldsymbol{y}_i \mid \boldsymbol{x}_i; \boldsymbol{\theta}_{old})$
9: $r_i \leftarrow \Delta(\boldsymbol{y}_i, \boldsymbol{y}_{ref})$
10: **end for**
11: $\mu_r \leftarrow \frac{1}{|\mathcal{Y}_{\boldsymbol{x}}|} \sum_i r_i$
12: $\sigma_r \leftarrow \sqrt{\sum_{i=1}^{|\mathcal{Y}_{\boldsymbol{x}}|} (r_i - \mu_r)^2 / |\mathcal{Y}_{\boldsymbol{x}}|}$
13: $\tilde{\mu}_q \leftarrow \text{MEDIAN}(\mathbf{q})$
14: $\tilde{\sigma}_q \leftarrow \text{MEDIAN}(|\mathbf{q} - \tilde{\mu}_q|)$ ▷ *Median abs. dev.*
15: **for** $i \in [1, |\mathcal{Y}_{\boldsymbol{x}}|]$ **do**
16: $\overline{r}_i \leftarrow (r_i - \mu_r)/\sigma_r$
17: $v_i \leftarrow \exp(-|q_i - \tilde{\mu}_q|/\tilde{\sigma}_q)$
18: **end for**
19: **sort** $\mathbf{q}$, $\overline{\mathbf{r}}$, $\mathbf{v}$, $\mathcal{Y}_{\boldsymbol{x}}$ **by** $\overline{\mathbf{r}}$ ▷ *Order by rewards*
20: **for** $i \in [1, \ldots, \frac{N}{2}, |\mathcal{Y}_{\boldsymbol{x}}| - \frac{N}{2} + 1, \ldots, |\mathcal{Y}_{\boldsymbol{x}}|]$ **do**
21: ENQUEUE($\boldsymbol{x}, \boldsymbol{y}_i, q_i, \overline{r}_i, v_i$)
22: **end for**
23: **end while**
24: **end function**

Algorithm 2 Learning algorithm

1: **function** LEARN($S, \eta, \boldsymbol{\theta}_{\text{CE}}$)
2: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{CE}}$
3: **for** $i \in [1, S]$ **do**
4: $\boldsymbol{x}, \boldsymbol{y}, q, \overline{r}, v \leftarrow \text{DEQUEUE}()$
5: $p \leftarrow \log p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$
6: $u \leftarrow \exp(p - q)$
7: $\alpha \leftarrow \mathbb{SG}(\min\{u \times v,\ 2\})$
8: $\mathcal{L} \leftarrow \alpha \times \overline{r} \times \log p(\boldsymbol{y} \mid \boldsymbol{x}; \text{dropout}(\boldsymbol{\theta}))$
9: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \times \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$
10: **end for**
11: **end function**

3 Experiments

3.1 Datasets

We run our model along with all the baselines on a total of 9 dataset–language direction translation tasks. See Appendix B for dataset details, preprocessing, and tokenization.

3.2 Training

For each task, we pretrain a sequence-to-sequence transformer model (Vaswani et al., 2017), using a word level cross entropy loss, until convergence. We refer to this model as the Cross Entropy (CE) model. All treatments for a task are initialized using the same CE checkpoint, which is the checkpoint that had the highest development set BLEU. The treatments are trained using the same training/development sets for a total of 200K steps with early stopping if the average development BLEU is not improved for the last 20K steps. In the case of PPO and MAD, every 20 training steps the learning node saves a checkpoint which the workers and evaluators load asynchronously. We use a global batch size of 256 training examples per step. See Appendix A for detailed hyperparameter configurations and Appendix C for implementation details.

3.3 Compared RL Objectives

Figure 1 gives the primary RL objectives we compare. These are: the REINFORCE algorithm (Williams, 1992) using a moving average

Greedy Decoding

Model	NIST Zh-En	IWSLT'14 De-En	IWSLT'14 En-De	WMT'14 De-En	WMT'14 En-De	WMT'20 Zh-En	WMT'20 En-Zh	WMT'20 Ps-En	WMT'20 En-Ps	μ
Cross Entropy	45.5	30.1	26.7	29.3	25.1	25.0	37.3	6.6	6.0	25.7
REINFORCE	45.5	30.2	26.7	29.3	25.3	25.0	37.1	7.2	5.9	25.8
PPO	45.9	31.0	27.7	29.6	25.2	24.8	37.3	7.3	7.5	26.3
MRT	45.2	30.8	26.7	30.4	25.1	26.4	38.8	8.4	7.5	26.6
−MAD	46.6	31.7	27.6	30.6	25.9	**27.0**	39.1	8.4	7.7	27.2
MAD	**47.7**	**32.1**	**28.0**	**30.8**	**26.2**	26.9	**39.3**	**8.8**	**7.9**	**27.5**

Beam Search | Beams = 5 | Length Normalization (α) = 1.0

Model	NIST Zh-En	IWSLT'14 De-En	IWSLT'14 En-De	WMT'14 De-En	WMT'14 En-De	WMT'20 Zh-En	WMT'20 En-Zh	WMT'20 Ps-En	WMT'20 En-Ps	μ
Cross Entropy	47.6	31.4	27.9	30.6	25.8	26.4	37.5	7.0	6.0	26.7
REINFORCE	47.5	31.4	28.0	30.6	25.9	26.3	37.8	8.1	6.0	26.8
PPO	47.5	31.3	28.2	30.4	25.8	26.2	37.5	7.7	7.6	26.9
MRT	47.0	31.9	27.9	**31.4**	25.9	27.4	39.3	9.2	7.9	27.6
−MAD	47.9	32.2	28.1	31.1	26.2	**27.5**	39.4	9.1	7.5	27.7
MAD	**48.4**	**32.4**	**28.3**	31.1	**26.4**	27.3	**39.6**	**9.4**	**8.1**	**27.9**

Table 1: Performance of different algorithms on translation. We report sacreBLEU on the held-out test set between the detokenized model hypothesis and original (not tokenized) references. Note that for IWSLT'14 and NIST both the hypotheses and references were lower-cased to keep comparable with prior works. μ is the average BLEU across all datasets for each method. MAD outperforms other methods by a large margin when greedy decoding is used. The gap shrinks when beam search is used, but MAD still outperforms the next best method, MRT.

baseline (Weaver and Tao, 2001), a proximal policy optimization algorithm (Schulman et al., 2017, PPO), minimium risk training (Shen et al., 2016, MRT), our algorithm without the MAD importance weights (−MAD), and our full algorithm (MAD). Since REINFORCE and MRT are on-policy algorithms, we optimise these objectives on a single worker; the distributed infrastructure is only used for MAD and PPO.

3.4 Evaluation and Hyperparameters

We use the *sacreBLEU* script (Post, 2018)[4] with detokenized model hypothesis and the original references.[5] When running on the test set, we select the checkpoint that had the highest development set BLEU under greedy decoding.

The important hyperparameters of our algorithm are N_{pop}, N, T_{min}, and T_{max}. For our main results, we used $N_{pop} = 36$ and $N = 12$ while the optimal temperature range was found for each model by sweeping over $[T_{min}, T_{max}] \in \{[0.2, 0.6], [0.4, 0.8], [0.6, 1.0], [0.8, 1.2]\}$.[6]

[4] https://github.com/mjpost/sacreBLEU
[5] See Table 9 in Appendix A for settings.
[6] See Table 10 in Appendix A for selected temperatures.

3.5 Main Results

Table 1 presents the performance of our algorithm compared to other objectives (§3.3) on four different machine translation datasets. We compare these models on both greedy decoding and beam search. As can be seen, our reinforcement learning algorithm outperforms both REINFORCE and PPO by an average of 1 BLEU across all the datasets we tried under both greedy and beam search decoding algorithms. The MAD algorithm is particularly strong when it comes to greedy decoding, where it outperforms the strong MRT baseline by .9 BLEU on average. More test set results using larger beams and other decoding hyperparameters can be found in Table 13 in Appendix D; these results show that MAD continues to outperform other training objectives and continues to improve with larger beams, even without any length normalization—in marked contrast to CE (Meister et al., 2020).

4 Analysis and Ablation Experiments

In this section we report on more detailed experiments on the IWSLT'14 German→English task, to show the impact of various components of the MAD algorithm. In particular, we look at the im-

Sampling	Reward	PPO	−MAD	MAD
$\mathrm{U}(1) \times 12$	$\widetilde{r}$	31.7	n/a	n/a
$\mathrm{U}(1) \times 12$	$\overline{r}$	32.5	31.8	31.3
$\mathrm{U}(12)$	$\overline{r}$	32.6	32.8	**33.1**

Table 2: PPO benefits from switching to input-conditional reward standardization, $\overline{r}$. −MAD and MAD benefit from training on more unique samples, while PPO is better able to reuse data. When using identical and ideal training conditions, MAD outperforms PPO by 0.5 BLEU. Results are development set *max* BLEU; CE baseline BLEU is 31.3.

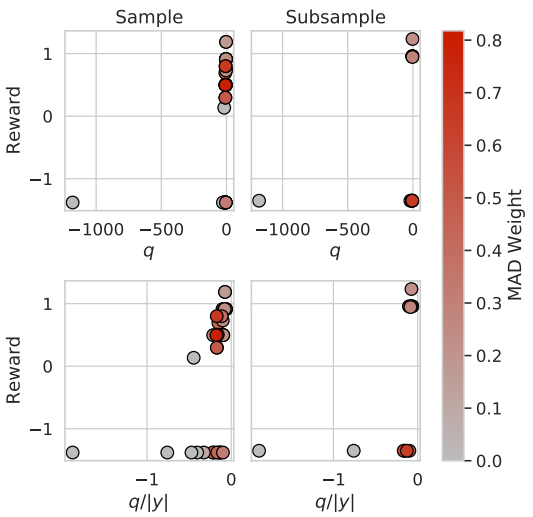

Figure 2: Plot of data generation for a single source sentence, see Table 14 in Appendix for sampled translations. Left column shows the complete sample while the right column shows only the examples selected for training. Top row x-axes are the sequence log probabilities q while the bottom row x-axes are the mean token log probabilities. The color indicates the weight assigned by our MAD importance weight.

pact of source-conditional reward standardization and the role of multiple samples (§4.1), more insight into the empirical behaviour of MAD importance weights (§4.2), the impact of the sample selection strategy (§4.3), the impact of optimizing different rewards (§4.4), and the training time (§4.5).

Notation. We briefly introduce some notation to help with clarity. We use the syntax $\mathrm{Type}(n) \times m$ to describe various selection methods (§2.4). The selection "Type" is either U for uniform random without replacement, T for top reward samples, or TB for top and bottom reward samples. n is the number of examples selected while m is the number of times each example is trained upon, which is an innovation proposed in Schulman et al. (2017).

4.1 Reward Standardization and Multiple Samples

The input-conditional reward normalization method (§2.2) has a large impact on performance. The first two lines of Table 2 compare the PPO algorithm with the standard batch-normalized reward ($\widetilde{r}$) versus the input-conditional normalized reward ($\overline{r}$). The second two lines compare the impact of using 1 sample 12 times or 12 samples 1 time each. While the PPO algorithm benefits little from different samples, our algorithm with and without the MAD weighting performs well.

4.2 MAD Importance Weight

This section investigates what the MAD importance weight is doing and to what extent it helps.

Data generation example. Figure 2 shows the reward, log probability, and MAD weight of the 36 transaltions sampled for the source sentence "*rsw : ich habe was ganz beachtliches gefunden.*" from the IWSLT'14 training dataset. The actual sampled translations are available in Table 14 of the supplementary material. While several very low-reward samples survive the selection process, the ones with very low q have a low MAD weight (corresponding to very strange translations) and will contribute little to the learning objective.

Training dynamics. We expect to see lower variance gradients during training due to the MAD importance weight and that is what Figure 3 shows. Regardless of the selection method used, including the MAD weight reduces training variance. We can also see from Table 1 that using MAD provides better generalization performance than not using it (−MAD).

4.3 Selection Methods

As stated in §2.4, we used $\mathrm{TB}(6, 6)$ as the selection method for the main experiments. An alternative choice that provides very similar properties in expectation is $\mathrm{U}(12)$ and as Table 3 shows, it works equally well in terms of development set BLEU performance. However, uniform sampling can lead to selections that are imbalanced especially when there are many samples that get a zero reward as is the case in Figure 2. This leads to higher gradient variance and norms during training (Figure 3).

An alternative sampling strategy is to take only the highest reward examples (Liang et al., 2006).

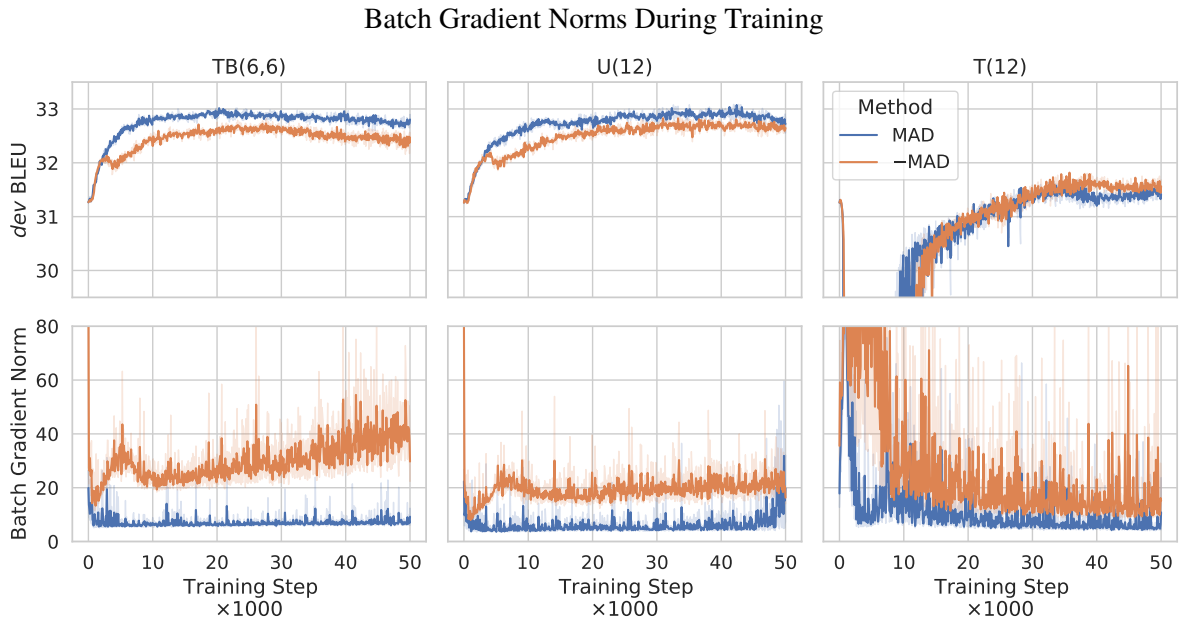

Figure 3: Learning curves for {−MAD, MAD} × {TB(6,6), U(12), T(12)} experiment matrix with 3 seeds per cell. Dataset is IWSLT'14 De-En. Top row is the BLEU score on the development set over the course of training. Bottom row is the batch level gradient norm. We see that MAD produces smaller and lower variance gradient updates than −MAD. There is not much difference between Top Bottom sampling and Uniform sampling, however, only using the Top examples leads to high variance and unstable training.

We explore this option in Table 3 and find that, although the *max* development BLEU is reasonably high, the *min* BLEU can be very low. The magnitude and variance of the gradients during training reflect this, thus, we recommend a selection method that uses a mix of both positive and negative reward examples.

Per Source		Selection	*dev* BLEU	
total	*unique*	Method	*max*	*min*
36	36	All	32.6	31.2
6	1	$\mathrm{T}(1) \times 6$	32.4	2.0
6	6	$\mathrm{T}(6)$	32.5	1.7
6	1	$\mathrm{U}(1) \times 6$	31.4	3.5
6	6	$\mathrm{U}(6)$	32.9	31.3
12	12	$\mathrm{U}(12)$	**33.1**	31.3
6	2	$\mathrm{TB}(1,1) \times 3$	32.0	4.1
6	6	$\mathrm{TB}(3,3)$	33.0	31.3
12	12	$\mathrm{TB}(6,6)$	**33.1**	31.3
36	12	$\mathrm{TB}(6,6) \times 3$	32.9	31.3

Table 3: Exploration of different selection methods when using MAD on the IWSLT'14 De-En dataset. General findings: using all samples is not optimal, using unique samples is better than reusing the same example, and $\mathrm{T}(N) < \mathrm{U}(N) \leq \mathrm{TB}(\frac{N}{2}, \frac{N}{2})$.

4.4 Impact of Reward Type

In the experiments reported so far, we have used sentence BLEU as a reward function. However, Choshen et al. (2020) have shown that uninformative rewards work as well as informative rewards at improving the BLEU score, conjecturing that the reported improvements in these algorithms are due to training artifacts such as reduced entropy of the predictive distribution rather than reward-driven learning. In this section we look at whether training with MAD on different rewards results in a policy that improve those rewards on held-out test sets.[7] We can see in Table 4 that the models are able to generalize on their metric and usually outperform models trained to optimize different metrics. We are able to train a model to optimize multiple metrics and generalize well on all of them.

4.5 Training time

Because of our distributed algorithm, we are able to obtain faster training times by increasing the number of workers that sample from the current policy (this in contrast to purely "on-policy" REINFORCE and MRT algorithms). Table 5 shows that

[7] We note that the constant reward used by Choshen et al. (2020) would result in no learning in our algorithm on account of the reward standardisation discussed above in §2.2, so we do not compare to this condition.

Reward Optimized	BLEU	GLEU	ChrF	TER	Token F1	BLEURT	μ
sBLEU (Papineni et al., 2002)	32.1	30.0	55.5	50.0	55.9	58.6	30.3
GLEU (Mutton et al., 2007)	**32.2**	**30.5**	55.7	49.1	57.0	59.5	31.0
ChrF (Popović, 2015)	32.0	30.0	**56.5**	49.7	56.0	59.0	30.6
−TER (Snover et al., 2006)	32.2	30.0	55.8	**48.8**	56.3	59.2	30.8
Token F1	31.9	30.4	55.5	49.5	**57.1**	59.5	30.8
BLEURT (Sellam et al., 2020)	29.5	28.4	54.9	52.0	54.8	**62.3**	29.6
ALL (*equal weight*)	32.2	30.4	56.0	49.0	56.8	59.9	**31.1**

Table 4: MAD was used to optimize different rewards on the IWSLT'14 De-En dataset. We report *test* set results for the checkpoint with the max validation performance on the metric being optimized. For TER, a lower score is better, so we optimize −TER. The last row, ALL, optimized the equally weighted average of the rewards, $(1/6)(\text{BLEU} + \text{GLEU} + \text{ChrF} + \text{Token F1} - \text{TER} + \text{BLEURT})$.

we get near linear scaling as the number of workers is increased.

Training Speed

Algorithm	Workers	Sec per 1K
REINFORCE	n/a	2,038
MRT	n/a	1,728
PPO & MAD	4	1,898
	8	946

Table 5: We show the training speed as a function of the number of 2x2 TPU workers used. Speed is measured in seconds per 1000 training steps.

5 Related Work

During the era of linear translation models, setting parameters to optimise rewards was standard. Although cross entropy is still widely used, Ranzato et al. (2016) inaugurated using RL techniques for neural translation models, and Edunov et al. (2018) provide a thorough survey of the topic and find that RL fine-tuning usually improves the original model. Although it has been shown to optimise a biased approximation to the expected reward objective (Choshen et al., 2020), minimum risk training remains extremely effective (Shen et al., 2016; Kiegeland and Kreutzer, 2021).

More complicated policy gradient methods have been proposed that rely on auxiliary networks. MIXER (Ranzato et al., 2016) predicts a per-token reward baseline while actor-critic methods (Bahdanau et al., 2017) learn a per-timestep Q function concurrently with the translation policy. Although not widely used yet in translation, proximal policy optimization (PPO) (Schulman et al., 2017) has been used to train summarization models from human preferences (Stiennon et al., 2020), and this algorithm provided inspiration for some of the techniques we used here.

Another important source of inspiration in the development of our algorithm was the "hope and fear" online learning algorithm of Chiang (2012), which used a margin-based analysis to argue for a decision rule that selected positive and negative training samples on the basis of a combination of current model score and reward. Additionally, pairwise ranking optimisation (Hopkins and May, 2011) reduced the parameter learning problem to classifying pairs of positively and negatively rewarded trajectories, and our objective can be understood as a weighted variant of that objective.

6 Conclusion

We introduce a new policy gradient algorithm, MAD, for machine translation and show that it outperforms existing reward-aware training procedures such as REINFORCE, minimum risk training (MRT) and proximal policy optimization (PPO). We test these algorithms on a variety of datasets, language pair directions, and reward functions and demonstrate good test set performance. Our algorithm excels at producing models that generate quality translations, even when greedy decoding is used. Our analysis shows that our method greatly reduces the variance of training gradients, which can be attributed to our robust importance weighting scheme and selecting both positive and negative reward examples for training. We use a distributed setup that enables simple scaling, allowing for fast training time if enough workers are used to generate the data. Finally, our algorithm has few hyperparameters and is reasonably insensitive to their values, making it easy to use on new tasks.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2017. An actor-critic algorithm for sequence prediction. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. 2021. Reverb: A framework for experience replay.

David Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13(40):1159–1187.

Leshem Choshen, Lior Fox, Zohar Aizenbud, and Omri Abend. 2020. On the weaknesses of reinforcement learning for neural machine translation. In *International Conference on Learning Representations*.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Corinna Cortes, Yishay Mansour, and Mehryar Mohri. 2010. Learning bounds for importance weighting. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 355–364, New Orleans, Louisiana. Association for Computational Linguistics.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Samuel Kiegeland and Julia Kreutzer. 2021. Revisiting the weaknesses of reinforcement learning for neural machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1673–1681, Online. Association for Computational Linguistics.

Julia Kreutzer, Artem Sokolov, and Stefan Riezler. 2017. Bandit structured prediction for neural sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1503–1513, Vancouver, Canada. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of ACL*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of EMNLP*.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 761–768, Sydney, Australia. Association for Computational Linguistics.

Clara Meister, Ryan Cotterell, and Tim Vieira. 2020. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2173–2185, Online. Association for Computational Linguistics.

Andrew Mutton, Mark Dras, Stephen Wan, and Robert Dale. 2007. GLEU: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 344–351, Prague, Czech Republic. Association for Computational Linguistics.

Khanh Nguyen, Hal Daumé III, and Jordan Boyd-Graber. 2017. Reinforcement learning for bandit neural machine translation with simulated human feedback. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1464–1474, Copenhagen, Denmark. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Matteo Papini, Andrea Battistello, and Marcello Restelli. 2020. Balancing learning speed and stability in policy gradient via adaptive exploration. 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 1188–1199. PMLR.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*,

pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Doina Precup, Richard S. Sutton, and Satinder Singh. 2000. Eligibility traces for off-policy policy evaluation. In *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML-00)*, pages 759–766.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1179–1195.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692, Berlin, Germany. Association for Computational Linguistics.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Lex Weaver and Nigel Tao. 2001. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 538–545, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. A study of reinforcement learning for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3612–3621, Brussels, Belgium. Association for Computational Linguistics.

Fan Yang, Gabriel Barth-Maron, Piotr Stańczyk, Matthew Hoffman, Siqi Liu, Manuel Kroiss, Aedan Pope, and Alban Rrustemi. 2021. Launchpad: A programming model for distributed machine learning research. *arXiv preprint arXiv:2106.04516*.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

A Hyperparameters

We provide the hyperparameters for our Transformer models in Tables 6 and 7. The algorithm specific settings are in Table 8. Finally, the setting used when calculation *sacre*BLEU are found in Table 9.

Transformer - Base Settings

Setting	Value
embeddings dim	512
feed forward	2048
num layers	6 *enc* + 6 *dec*
num heads	8
dropout	.3
tied embeddings	True
tied softmax	True
optimizer	Adam
learning rate (η)	1e-5
η warmup steps	1000
η scheduling	Constant
gradient clipping	global_norm(1)
training steps	200,000
seq length	128
global batch size	256

Table 6: Base settings for our Transformer models.

Transformer - Deltas

Dataset	Setting	Value
WMT'20 Ps ↔ En	seq length	64
WMT'14 De ↔ En	seq length	144
IWSLT'14 De ↔ En	feed forward	1024
	num heads	4

Table 7: Deviations from Transformer base settings.

B Datasets

We evaluate all the models on the following translation tasks: NIST[8] Open MT Chinese → English task, IWSLT'14[9] English ↔ German translation task, WMT'14[10] English ↔ German news translation task, and the WMT'20[11] Chinese ↔ English and Pashto ↔ English news translation tasks. The sizes of each dataset is avaliable in Table 11.

[8] https://www.nist.gov/itl/iad/mig/open-machine-translation-evaluation

[9] https://sites.google.com/site/iwsltevaluation2014/mt-track

[10] http://statmt.org/wmt14/translation-task.html

[11] http://www.statmt.org/wmt20/translation-task.html

Algorithm Settings

Algorithm	Setting	Value
PPO	ϵ	0.2
MRT	$\lvert\mathcal{Y}_{\boldsymbol{x}}\rvert$	5
MAD *reverb settings*	max times sampled	1
	min size to sample	512
	max size	4096
	sampler	Uniform
	remover	Fifo

Table 8: Settings for sequence level algorithms. Note that we did run experiments with $\lvert\mathcal{Y}_{\boldsymbol{x}}\rvert = 12$ for MRT and found no improvement in development set max BLEU. Given the slow training speed of MRT, we opted to use the convention of 5 samples.

*sacre*BLEU Settings

Language	Setting	Value
De	tokenizer	intl
En	tokenizer	13a
Ps	tokenizer	intl
Zh	tokenizer	zh
All	smooth method	exp
	smooth value	0

Table 9: *sacre*BLEU settings for each language.

Preprocessing We perform text normalization on the datasets before tokenization.

- All languages - Unicode canonicalization (NKFD from), replacement of common multiple encoding errors present in training data, standardization of quotation marks into "directional" variants.
- English - Replace non-American spelling variants with American spellings using the aspell library.[12] Punctuation was split from English words using a purpose-built library.
- Chinese - Convert any traditional Chinese characters into simplified forms and segment into word-like units using the Jieba segmentation tool.[13]

Tokenization We encode text into sub-word units using the `sentencepiece` tool (Kudo and Richardson, 2018). When generating our own sub-word segmentation, we used the algorithm from Kudo (2018) with a minimum character coverage of 0.9995.

[12] http://wordlist.aspell.net/varcon-readme/

[13] https://github.com/fxsjy/jieba

Temperatures			
Dataset	Algo	T_{min}	T_{max}
IWSLT'14 De → En	PPO	1.0	
	MRT	0.9	
	MAD	0.8	1.2
IWSLT'14 En → De	PPO	1.2	
	MRT	0.6	
	MAD	0.8	1.2
WMT'14 De → En	PPO	0.2	
	MRT	0.6	
	MAD	0.4	0.8
WMT'14 En → De	PPO	1.2	
	MRT	0.4	
	MAD	0.4	0.8
WMT'20 Ps → En	PPO	1.2	
	MRT	1.2	
	MAD	0.8	1.2
WMT'20 En → Ps	PPO	1.2	
	MRT	1.0	
	MAD	0.4	0.8
NIST Zh → En	PPO	1.0	
	MRT	0.8	
	MAD	0.6	1.0
WMT'20 Zh → En	PPO	0.8	
	MRT	0.8	
	MAD	0.4	0.8
WMT'20 En → Zh	PPO	0.8	
	MRT	0.6	
	MAD	0.2	0.6

Table 10: Temperatures used for models in main results. REINFORCE used same temperatures as PPO. Only MAD uses a temperature range, other algorithms use as single temperature.

C Implementation Details

Software We use Launchpad (Yang et al., 2021) to create and launch our DAG computing graph. The graph consists of a Reverb table (Cassirer et al., 2021) that holds training examples, multiple worker nodes that generate and publish examples, a learner node that pulls and trains on examples, and an evaluator node that loads fresh checkpoints to calculate dev set sacreBLEU.

Compute All of our program nodes run on a 2x2 TPU configuration. Our hyperparameter sweeps for PPO and MAD used 4 data generating workers running for 50K steps, while final results used 8 workers and ran for 200K steps. Table 5 in the body of the paper provides the wall clock training speed for each algorithm.

Dataset	Train	Dev	Test
IWSLT'14 De → En	164K	7,466	6,750
IWSLT'14 En → De	173K	1,474	6,750
WMT'20 Ps ↔ En	516K	5,860	2,719
NIST Zh → En	1.45M	1,664	5,146
WMT'14 De ↔ En	4.7M	3,000	3,003
WMT'20 Zh → En	21.8M	2,000	2,000
WMT'20 En → Zh	21.8M	1,997	1,418

Table 11: Number of training, development, and test examples in each dataset.

D Supplementary Experiments

D.1 Additional Beam Search Results

Table 13 shows results (on the test set) using larger beams and different decoding hyperparameters. MAD trained with sentenceBLEU continues to outperform other training objectives, and displays the notable behaviour that decoding without length normalization with large beams is effective, whereas performs drops significantly with CE and REINFORCE trained models.

D.2 Alternative Importance Weightings

Our motivation for the particular form of the MAD weight is described in §2.3. However, there are other ways this could have been done. Here we evaluate 2 alternative formulations of the MAD weight. 1) Using the log probability of the greedy translation as $\tilde{\mu}_q$, this down weights examples that are far from the greedy translation. 2) Z-scoring $\mathbf{q}$ rather than using median absolute deviations. We try these alternatives on a subset of the datasets and report the results in Table 12.

Importance Weighting	IWSLT'14 De-En	WMT'14 En-De	NIST Zh-En	WMT'20 En-Ps	WMT'20 Zh-En	WMT'20 En-Zh	μ	σ
CE	31.3	24.7	47.9	6.8	24.4	32.5	27.9	-
$-$MAD	32.7	25.7	50.0	8.4	27.1	34.0	29.6	.22
MAD z-score	32.7	**25.9**	50.1	8.5	**27.3**	**34.3**	**29.8**	.14
MAD greedy	**33.2**	25.1	50.1	8.4	25.6	33.4	29.3	.15
MAD	33.1	25.7	**50.8**	**8.6**	26.9	34.0	**29.8**	**.12**

Table 12: Alternative formulations of the MAD importance weight. μ is the average development set *max* BLEU across all the datasets in the table while σ is the standard deviation of BLEU when looking at the 10K step window centered on *max* BLEU. Using any of the importance weights reduces BLEU variance, but MAD provides the highest performance (tie) and lowest variance.

Beam Search | Beams = 5 | Length Normalization (α) = None

Model	NIST Zh-En	IWSLT'14 De-En	IWSLT'14 En-De	WMT'14 De-En	WMT'14 En-De	WMT'20 Zh-En	WMT'20 En-Zh	WMT'20 Ps-En	WMT'20 En-Ps	μ
Cross Entropy	47.0	31.1	27.5	29.9	26.3	25.1	34.7	7.8	5.9	26.1
REINFORCE	47.0	31.1	27.5	29.9	26.3	25.0	34.9	7.6	5.8	26.1
PPO	47.1	31.2	28.0	29.9	26.2	25.0	34.7	8.1	7.5	26.4
MRT	47.4	31.7	27.5	**31.2**	26.3	27.0	38.8	9.2	7.8	27.4
$-$MAD	48.0	32.2	27.9	31.0	26.3	**27.2**	39.2	8.9	7.4	27.6
MAD	**48.6**	**32.4**	**28.2**	31.0	**26.5**	27.0	**39.4**	**9.3**	**8.0**	**27.8**

Beam Search | Beams = 50 | Length Normalization (α) = None

Model	NIST Zh-En	IWSLT'14 De-En	IWSLT'14 En-De	WMT'14 De-En	WMT'14 En-De	WMT'20 Zh-En	WMT'20 En-Zh	WMT'20 Ps-En	WMT'20 En-Ps	
Cross Entropy	42.8	28.1	26.5	28.8	24.4	22.9	32.3	4.0	1.3	23.4
REINFORCE	43.0	28.0	26.4	28.8	25.0	22.8	32.0	2.5	1.3	23.3
PPO	45.9	31.2	28.0	29.2	24.5	23.1	32.3	6.9	6.3	25.3
MRT	47.1	30.8	26.5	**31.0**	26.4	26.8	38.4	**9.0**	**7.1**	27.0
$-$MAD	47.9	32.2	27.6	31.0	26.4	**27.1**	39.1	7.2	6.9	27.3
MAD	**48.6**	**32.4**	**28.1**	30.9	**26.6**	26.9	**39.2**	7.9	6.4	**27.4**

Beam Search | Beams = 50 | Length Normalization (α) = 1.0

Model	NIST Zh-En	IWSLT'14 De-En	IWSLT'14 En-De	WMT'14 De-En	WMT'14 En-De	WMT'20 Zh-En	WMT'20 En-Zh	WMT'20 Ps-En	WMT'20 En-Ps	
Cross Entropy	48.2	31.6	27.9	30.5	24.8	25.9	36.8	5.4	2.9	26.0
REINFORCE	48.3	31.6	28.0	30.5	25.6	25.9	36.9	6.2	2.8	26.2
PPO	47.8	31.3	28.2	30.3	24.8	25.9	36.9	7.1	6.9	26.6
MRT	47.6	32.1	27.9	**31.3**	25.7	**27.5**	39.0	**9.0**	7.2	27.5
$-$MAD	48.1	31.9	27.1	31.1	26.2	27.4	39.5	8.4	7.2	27.4
MAD	**48.5**	**32.5**	**28.4**	31.1	**26.4**	27.3	**39.5**	8.8	**7.4**	**27.8**

Table 13: Additional test set results when using different Beam Search hyperparameters. Even when large beams are used without length normalization, MAD performs well indicating that it is better calibrated.

$\mathcal{Y}_{\boldsymbol{x}}$	$\overline{r}$	q	v
rsw : i found what was a really remarkable thing . (*greedy*)	0.91	-3.1	1.7e-01
rsw : i found what was a really remarkable thing . (*duplicate*)	0.91	-3.1	1.7e-01
rsw : i found what was a remarkable thing .	1.19	-3.2	1.8e-01
rsw : i found what was a remarkable thing . (*duplicate*)	1.19	-3.2	1.8e-01
rsw : i found something quite remarkable .	0.90	-3.6	2.2e-01
rsw : i found what was quite a point .	0.50	-3.6	2.2e-01
rsw : i found what was quite a point . (*duplicate*)	0.50	-3.6	2.2e-01
rsw : i found what was a very remarkable thing .	0.91	-3.7	2.3e-01
rsw : i found what was a great thing .	0.73	-3.8	2.4e-01
rsw : i found what was quite a remarkable thing .	0.91	-3.9	2.5e-01
rsw : i 've found something quite remarkable .	-1.38	-4.5	3.3e-01
rsw : i found what was a real thing .	0.73	-4.7	3.5e-01
rsw : i found something quite substantial .	0.80	-4.9	3.8e-01
rsw : i found something quite a point .	0.75	-5.0	4.0e-01
rsw : i found what was a very substantial thing .	0.50	-5.2	4.4e-01
rsw : i found what was a pretty remarkable thing .	0.91	-5.2	4.5e-01
rsw : i 've found what was quite a point .	-1.38	-5.4	4.9e-01
rsw : i 've found what was quite a point . (*duplicate*)	-1.38	-5.4	4.9e-01
rsw : i 've found a remarkable thing .	-1.38	-5.9	6.2e-01
rsw : i 've found a real thing .	-1.38	-6.5	8.2e-01
rsw : i found what was a pretty substantial thing .	0.50	-7.3	8.2e-01
rsw : i found what was a really striking .	0.50	-7.4	7.9e-01
rsw : i found something quite extraordinary .	0.80	-7.8	6.7e-01
rsw : i found what was quite a convincing thing .	0.50	-7.8	6.6e-01
rsw : i found what was a pretty significant one .	0.30	-8.4	5.0e-01
rsw : i 've found what 's quite a remarkable thing .	-1.38	-8.6	4.4e-01
rsw : i found what was a pretty substantial point .	0.30	-8.6	4.4e-01
rsw : i found what was really a really remarkable thing .	0.69	-8.7	4.3e-01
rsw : i 've found something remarkable .	-1.38	-8.8	4.1e-01
rsw : i found what was a substantial amount .	0.50	-9.6	2.8e-01
rsw : i 've found a really a point .	-1.38	-11.7	1.1e-01
rsw : i found a lot of a point of all .	0.13	-17.4	7.4e-03
rsw : i 've found a burk .	-1.38	-19.7	2.5e-03
rsw : well , i did find what was most important .	-1.38	-19.8	2.4e-03
rsw : but but i found what was the most substantial case .	-1.38	-27.4	6.9e-05
rsw : i 've figured out what was oder oder oder oder oder oder od...	-1.38	-1180.3	0.0e+00
rsw : ich habe was ganz beachtliches gefunden . (*source*)			
rsw : i found the most remarkable thing . (*reference*)			

Table 14: Example of sampling and TB(6,6) selection during training, together with the MAD importance weights, rewards, and policy log likelihood. Elements in gold represent the top and bottom 6 elements of $\mathcal{Y}_{\boldsymbol{x}}$ that will be presented to the learner.