Application of Reward Learning to generate news

Thorben Schomacker, Stephan Pareigis

Hamburg University of Applied Sciences, Department of Computer Science, Hamburg, Germany
thorben.schomacker@haw-hamburg.de
stephan.pareigis@haw-hamburg.de

Abstract. This paper examines the usage of proximal policy optimization applied to pre-trained neural language models based on the transformer architecture. This approach is then used to generate convincing News.

Keywords: natural language processing · reward learning · Transformer · reinforcement learning · proximal poliy optimization · attention-based models · deep learning

1 Introduction

Today's neural language processing (NLP) models such as DeBERTa [7] are starting to outperform human results on common metrics such as GLUE and SuperGLUE. This does not indicate that the models are reaching the real human performance in natural language understanding but that the current metrics are getting pushed to their boundaries in evaluating current models. This leaves a demand for evaluating model performance beyond the existing metrics. GLUE and SuperGLUE are both basically mathematical indicators. An alternative way of evaluating model performance is the usage of judges. A judge could be human labeler or an additional model. [18] developed a concept of how a human-in-the-loop approach can be used to evaluate and optimize the performance of models using reinforcement learning. Their idea is similar to Generative Adversarial Nets (GANs) introduced in [6]. The GAN architecture consists of two main components: a generator model and discriminator model. The generator model generates outputs and the discriminator model evaluates the quality of the generator's output. [18] used human labelers as discriminators. Another recent application of a generator-discriminator approach in NLP is ELECTRA. ELECTRA [4] improved the benefits of the masked language modeling task by not only replacing the selected token with the `MASK` token but with a token, that is very similar to the original token. And afterwards training a discriminator to restore the original token based on the token's context. Thereby enabling the model to create a more differentiate token representation, which enriches the model's a language understanding capability.

In this paper a generator-discriminator approach is used and trained with Proximal Policy Optimization (PPO) to solve a NLP task. The implementation follows the findings and ideas proposed in [18], uses the trl library [15] and pre-trained models provided by [1]. The task of the experiments is to disguise fake news to convince a fake news classification model (discriminator) of the output's authenticity.

This paper is divided into eight sections. Section 1 provides an introduction into the topic of this paper. Section 2 gives background information needed to understand PPO, which is discussed in section 3. Section 4 talks about pre-trained language models and section 5 explains how these models can be fine-tuned based on human preferences. Afterwards section 6 elaborates the details of the implementation used in the experiments in section 7. The section, section 8 discusses the results of the experiments and delivers an outlook.

2 Background

This paper highly relies on the mechanisms and approaches in [18]. They are built on top of two ideas that are widely used in reinforcement learning. To better understand the novelties and advantages of [18]'s approaches the following subsections further explain these two ideas.

2.1 Policy Gradient Methods

Classical reinforcement learning algorithms (further explained in [12, p. 321]) are action-value methods. That means their policy is completely based on assigning a value to each action. Policy gradient methods learn a parameterized policy. This policy allows the model to select actions without assigning a specific value to the action. Despite this a value function can still be used to learn the policy parameter without requiring it for action selection. For vanilla policy gradients (e.g. the REINFORCE algorithm) the objective used to optimize the neural network looks like:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

Where $\hat{A}$ could be the discounted return, as in REINFORCE or the advantage function. By taking a gradient step on this loss with considering the network parameters actions that led to a higher reward will be incentivized.

2.2 Trust Region Methods

The vanilla policy gradient shown in equation 1 uses the *log* probability of the action to calculate the impact of the actions. [8] introduced an alternative of calculating the impact of the action. It uses the probability of the action under the current policy π, divided by the probability of the action under your previous policy π_{old}:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{2}$$

So $r(\theta)$ greater than 1 indicates that the action is more probable for the current policy than it is for the old policy. When $r(\theta)$ is between 0 and 1 the action is more probable for the old policy than for the current. Trust Region Policy Optimization (TRPO) build an objective function with $r(\theta)$:

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r(\theta) \hat{A}_t \right] \tag{3}$$

One problem discussed in [8] is that $r(\theta)$ tend to be really big when the action is, by far, more probable in the current policy. This big $r(\theta)$ leads to taking big gradient steps that might dramatically change the policy in a negative way. [8] adds several mechanisms such, as KL Divergence constraints, to tackle this problem help guarantee that it is monotonically improving.

3 Proximal Policy Optimization

The Proximal Policy Optimization (*PPO*) algorithm was introduced in [11] and became one of the most popular reinforcement learning methods. It is an on-policy algorithm that means the policy is updated based on small batches of collected experiences of interacting with the environment. PPO does not have a replay buffer, so once the policy is updated, the batch is discarded. PPO's wide acceptance and usage are mainly because it is sample efficient and because it is easy to implement and to tune. Furthermore PPO is a policy gradient method which means that it can only learn online. The agent follows his own policy and can pick actions. The corresponding gradient estimator is computed as follows:

$$\hat{g} = \hat{\mathbb{E}}_t [\nabla_\theta log \pi_\theta(a_t|s_t) \hat{A}_t] \tag{4}$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at timestep t. This estimator is basically a neural network which estimates the future reward. But this estimate is noisy. Furthermore PPO has three key features which are mainly responsible for its success. The following subsection will further discuss them.

3.1 Clipped Surrogate Objective

The previous subsection 2.2 about TRPO described the problem that a large $r(\theta)$ can lead to drastic changes of the police. These changes could destroy a effortfully trained policy. One way of overcoming this problem is by adding a few additional mechanisms as discussed in [8]. Another, more simple way, was proposed by Schulman et al. in [11]. They solved the problem with the Clipped Surrogate Objective (*CSO*):

same objective from before

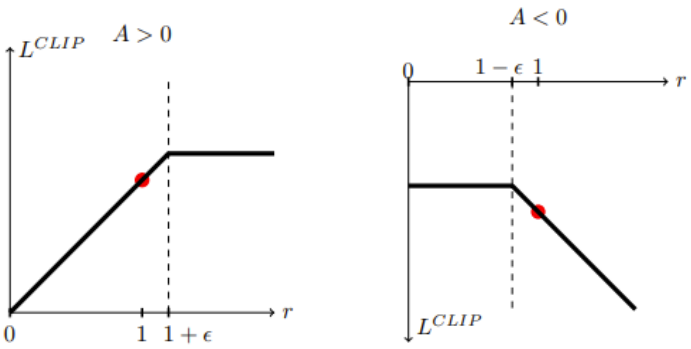
same, but r(θ) is clipped between (1 - e, 1 + e)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

min of the same objective from before and the clipped one

Fig. 1: The term $r_t(\theta)\hat{A}_t$ (highlighted in blue) describes the as $r(\theta)\hat{A}_t$ in Eqn. 3. The second term (red) describes $r_t(\theta)$ clipped between (1 - e, 1 + e). Note that [11] suggested $e = 0.2$ so r can vary between (0.8, 1.2). The minimization of both terms in highlighted in green. Illustration by [16]

Fig. 2 shows two possible values of the objective in Fig. 1.



(a) In this scenario the action had an estimated **positive** effect on the outcome.

(b) In this scenario the action had an estimated **negative** effect on the outcome.

Fig. 2: Comparison of two different scenarios of the CSO depicted by [11]

A further illustration of CSO is given in Fig. 2. This illustration compares two different values of the CSO. Fig. 2a shows what happens in the case of an action becoming a lot more probable under the current policy than it was for the old: the r-value gets clipped. This clipping prevents from stepping too far in changing the policy. Due to the fact that this is just a local approximation and sample of our policy, taking such great steps would not be accurate. Clipping has the effect of blocking the gradient in the backward pass. The flat lines after $1 + e$ in Fig. 2a indicates this effect. Fig. 2b depicts the same mechanism but in the case where the action would have an estimated negative effect on the outcome.

Another issue that the CSO solves is $r(\theta)$ growing indefinitely. This scenario can be caused by a gradient step that made an action a lot more probable but make our policy worse. The Clipped Surrogate Objective allows to 'undo'

such steps. The function will become negative in this case so it will cause the gradient to move in the opposite direction proportionally to the value of $r(\theta)$. The mechanism also takes effect in the scenario depicted in Fig. 2b. These 'undo' regions are the reason why the minimization term in Fig. 1 must be included. If this minimization term would not be included these regions would be flat (gradient = 0), so 'undoing' would not be possible.

3.2 Adaptive KL Penalty Coefficient

[11] also discussed an alternative or an addition to CSO: Using a penalty on *KL divergence*. And to adapt the penalty coefficient so that some target value of the KL divergence d_{targ} is achieved each policy update. Despite the fact that in the experiments in [11] KL penalty performed worse than the CSO, [18] added KL penalty to their implementation.

[11] performed the following steps in each policy update, in the simplest instantiation of this algorithm:

> Using several epochs of minibatch stochastic gradient descent (SGD), optimize the KL-penalized objective
>
> $$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi\theta_{old}(a_t|s_t)} \hat{A}_t - \beta KL\left[\pi\theta_{old}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right] \right] \tag{5}$$
>
> Compute $d = \hat{\mathbb{E}}_t KL\left[\pi\theta_{old}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]$
> – If $d < d_{targ}/1.5, \beta \leftarrow \beta/2$
> – If $d > d_{targ} \times 1.5, \beta \leftarrow \beta \times 2$

The updated β is used for the next policy update. With this scheme, occasionally policy updates, where the KL divergence is significantly different from d_{targ}, can occur. These occurrences are rare and β adjusts quickly. [11] heuristically chose the parameters 1.5 and 2 but the algorithm is not very sensitive to them. Another hyperparameter is the value of β its tuning can be neglected because the algorithm quickly adjusts it.

3.3 Multiple epochs for policy updating

The previously discussed CSO or alternatively the KL-Penalty allow PPO to run multiple epochs of gradient ascent on your samples without causing destructively large policy updates. This leads to a more efficient way of using your data and helps to reduce sample inefficiency. Algorithm 1 shows the usage of multiple epochs and the parallel actor approach popularized in [9].

Running K epochs of gradient ascent on the trajectory is one central feature of PPO. As stated in [11] vanilla policy gradient methods are unable to run multiple epochs because they are possibly taking taking too large steps when changing the policy. But the previously discussed mechanisms allow PPO to run multiple epochs.

Algorithm 1 PPO, actor-critic from [11]

1: **for** iteration=1,2,... **do**
2: **for** actor=1,2,...,N **do**
3: Run policy $\pi\theta_{old}$ ▷ Sampling the environment with pi_{old}
4: Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
5: **end for**
6: Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
7: $\theta_{old} \leftarrow \theta$
8: **end for**

For each iteration, after line 3 and at line 6 in Algorithm 1 the current policy π will be exactly equal to π_{old}. That means at first, none of the updates will be clipped and it is guaranteed that these example affect the policy. As the policy is updated over multiple epochs, the objective will eventually start hitting the clipping limits. So, the gradient go to 0 for these samples and the training will gradually stop for this iteration.

4 Pre-Trained Models

Every machine learning model can only be as good as the data, it is trained on. In the field of NLP labeled and prepared data is rather sparse in comparison to the abundant supply of unlabeled data. The idea behind transfer learning is to use this unlabeled data to pre-train models for world knowledge and a general language understanding. These *pre-trained models* are afterwards fine-tuned on task-specific data to solve a particular task. Thereby the task-specific, labeled and prepared data is more efficient used. The first highly successful approach of bringing transfer learning to NLP, was the Transformer architecture introduced in [13]. The models used in this paper are also based on this architecture and follow the pre-train and fine-tune approach.

5 Fine-Tuning Language Models from Human Preferences

One of the greatest challenges in training neural language models is defining a metric to which the model is optimized and evaluated on. The current standards such as GLUE and SuperGLUE are based on comparing the outputs of the model to a gold standard. The problem of this way of evaluating the model's performance is that the task output is reduced to one correct answer: The gold standard. But for many NLP tasks there is not one correct answer but many. Optimizing the model towards only one answer leads to a model with a bias. Additionally papers like [3] showed that language models in general are vulnerable to biases in the input data.

One way of reducing the bias of the results of neural language models could be increasing the amount of training data by e.g. data augmentation as shown in [14]. This could add more variance to the model's output but still carries at least some of the bias of the original input data. Another way could be the involvement of human judges. This approach is detached from the original gold standard set and is based on the subjective evaluation of a human judge. This could lead to a metric much closer to the way the human language understanding is evaluated naturally. The approach proposed in [18] implemented this idea. They included the label of human labelers to calculate the reward of the model's generated output. Thereby applying the previously discussed PPO method to fine-tune pre-trained language models with a human-in-the-loop approach. Fig. 3 depicts the involvement of Human labelers in the training process.

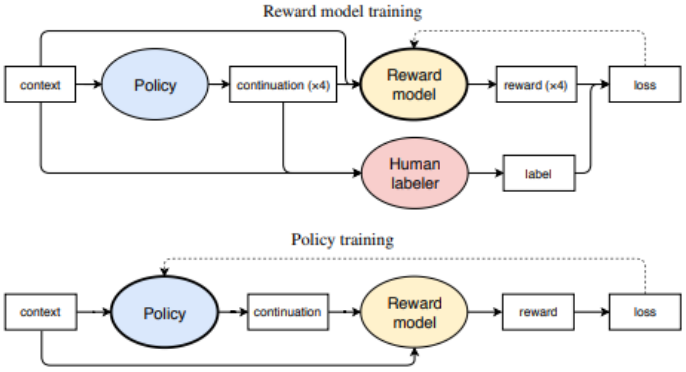


Fig. 3: Depiction of the architecture of policy training and the involvement of a human labeler in training described in [18]

6 Implementation Details

The implementation used in this paper is built on top of the Transformer Reinforcement Learning library [15] initiated by Leandro von Werra. The goal of this library is to fine-tune pre-trained models from the transformer library [1] with [18]'s reward learning approach discussed in section 5. Additionally [15] uses PPO2 which is a PPO optimized for using it on GPU [2]. It is basically the same as the normal PPO but with a few modifications.

In this paper instead of a human labeler there will be a pre-trained model which is fine-tuned to classify inputs. The generator model is fine-tuned on a similar data set as the classifier and tries to maximize a reward, which is based of the classifiers output, by generating text. This is shown in Fig. 4.

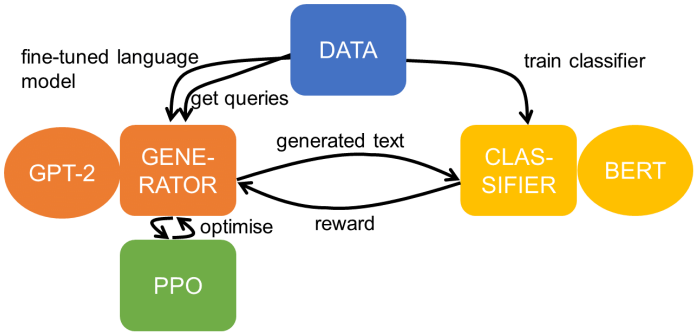


Fig. 4: Architecture of the experimental setup

6.1 Transformer Models

BERT [5] and *GPT-2* [10] are both representatives of transfer learning models in NLP which are further discussed in section 4. They are both based on the transformer architecture [13]. Despite the fact that both are very well applicable to numerous tasks, GPT-2 is more often used in text-generation task such as continuing given inputs. BERT is one of the most used models for processing queries on data for instance classifying text by its sentiment. In this paper GPT-2 is the basis for the *generator* component and BERT for the *classifier*.

6.2 Value Head

[15] introduced the value head. It is an additional head on the GPT-2 model that estimates a scalar for each output token. These estimates on the current states value are require in order for PPO to work.

6.3 KL-controllers

To prohibit the learned policy from moving to far away from the original language model [15] used the KL divergence between the new policy and the reference policy, which in this case is the language model before PPO training. This KL divergence serves an additional reward signal. It was further described in section 3.2. By this additional reward signal a large KL-divergences gets punished and staying close to the reference is rewarded.

[15] used two KL-controllers which are also presented in [18]: an *adaptive log-space proportional controller* and a *fixed controller*. [18] found out that models trained with different seeds and the same KL penalty β are sometimes hard to compare because they resulted in quite values of $KL(\pi, \rho)$. To overcome this

problem Ziegler et. al. used the log-space proportional controller in some experiments in [18] to dynamically vary β to target a particular value of $KL(\pi, \rho)$

$$e_t = clip\left(\frac{KL(\pi_t, \rho - KL_{target}}{KL_{target}}, -0.2, 0.2\right)$$

$$\beta_{t+1} = \beta_t(1 + K_\beta e_t)$$

Where [18] used $K_\beta = 0.1$

7 Experimental Setup

In the following section a few experiments will be conducted to find out which hyperparameter setup is best suited to generate convincing news. It will start from the experimental setup described in [15] as a baseline.

7.1 Data

For the experiments in this paper Clément Bisaillon's "Fake and real news dataset" [1] was used. It contains in total 38729 news articles with 46% (17903) fake and 54% (20826) real news articles. All articles are written in English. The majority of them is dealing with politics and/or with topics related to the USA. Thus making the data quite homogeneous. This homogeneity makes it easier for the classifier and for the generator to determine nuances that distinguishes fake from true news. Originally the fake and real news where stored in two separated tables each table had four columns:

title: The title or headline of the article
text: The text or content of the article
subject: The subject or category of the article
date: The date the article was originally published

In the process of data preparation both tables where joined together with an additional column `label` which has the value 1 if the article is real and 0 if the article is fake news. Tab. 1 shows example entries. For the experiments only the title and the label were used. Instead of the title the text could be used but in most the cases the title itself indicates fake news in a much shorter way than the complete content text. So the generators output can also be shorter to influence the classifiers estimation of the modified fake news.

[1] https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

label	title	text	subject	date
0	Sheriff David Clarke Becomes An Internet Joke For Threatening To Poke People 'In The Eye'	On Friday, it was revealed that former Milwaukee Sheriff David Clarke, who was being considered for ...	News	December 30, 2017
1	As U.S. budget fight looms, Republicans flip their fiscal script	WASHINGTON (Reuters) - The head of a conservative Republican faction in the U.S. Congress, who voted...	politicsNews	December 31, 2017

Table 1: Example items from the fake and real news data set

7.2 Results

The classifier is a binary classification model based on the pre-trained `bert-base-cased`. It was trained with a learning rate of $1e-5$ and for 1 epoch on the data. The generator is based on `gpt2` and was trained for 6400 steps with a batch size of 256, a forward batch size of 16, a text input length of 5, a text output length of 15, a target of 6, 10000 as horizon, gamma 1, a range for clipping in PPO policy gradient loss of 0.2, a range clipping values in loss calculation of 0.2 and a scaling factor for value loss of 0.1 on the data. The following paragraphs show experiments on the learning rate, the initial kl coefficient value and the lambda value with the purpose of tuning these hyperparameters to further improve results.

Lambda: Lambda and gamma are the two factors that determine the degree to which the previous advantage influences the current advantage. Gamma is set to 1 and the baseline value of lambda is 0.95. Fig. 5 shows that a value between $0.95-1.01$ perform well. Values greater than 0.95 seem to slightly increase the performance. Decreasing lambda slowed the convergence of the mean rewards towards the maximum value. This maximum seems independent to lambda. So a high value lambda value speeds up the training process. The baseline value of 0.95 seems sufficient.

Leaning rate: The learning rate determines the step size of the gradient descent. A too small learning rate slows down the the process of finding a minimum. In this case very small improvements of the policy over time so that the initial policy is almost unchanged. A too big learning rate results in drastic changes of the policy. This could destroy an effortfully trained policy. A big part of PPO is prohibiting too large changes of the policy. So it seems plausible that Fig. 6 indicates that the model's performance is very sensitive to changes of the learning rate. A small reduction of the learning rate decreased the steps needed to converge and raised the maximum achieved mean reward. In this setup 0.00001 performed the best.

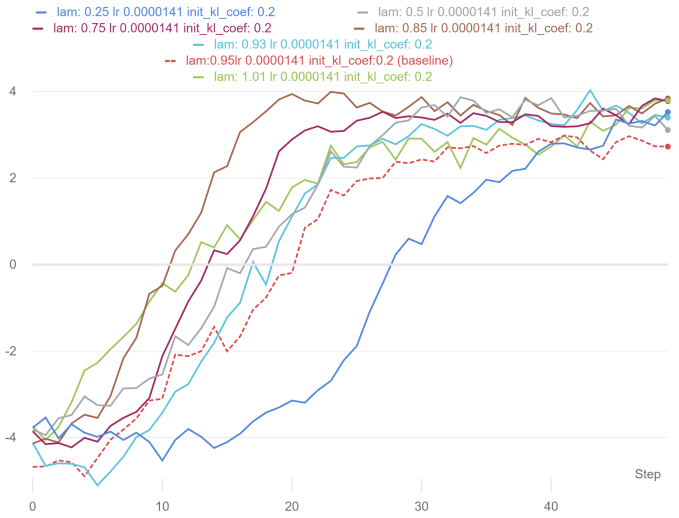


Fig. 5: mean reward with lambda $\in \{0.25, 0.5, 0.75, 0.93, 0.95, 1.01\}$

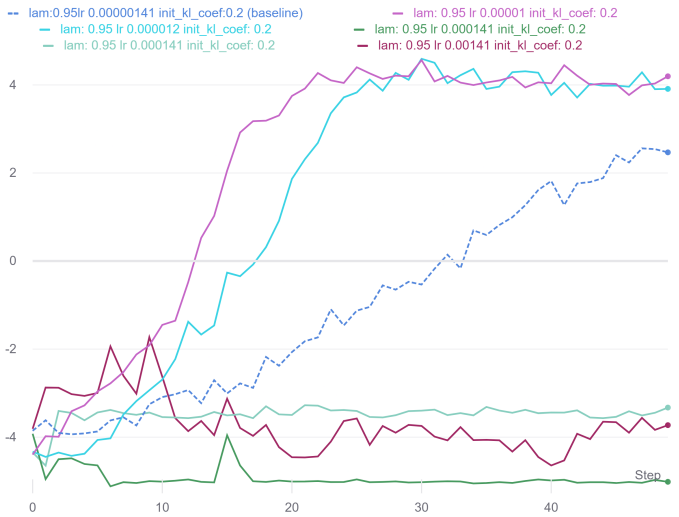


Fig. 6: mean reward with learning rate $\in \{0.00001, 0.000012, 0.0000141, 0.000141, 0.00141\}$

Initial KL-Coefficient: The Initial KL penalty coefficient is used for the adaptive and the linear control. It is the initial the value and determines the factor of the KL penalty. Fig. 7 indicates that values smaller than the baseline value of 0.25 increase model's performance. In this particular setup 0.15 performed the best.

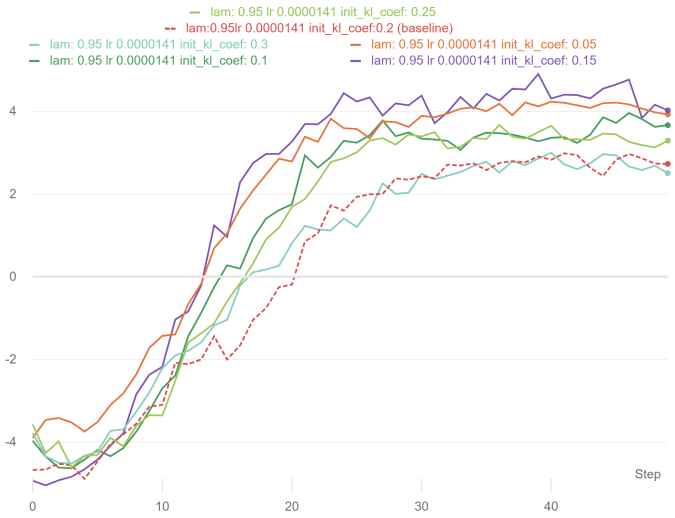


Fig. 7: mean reward with an initial KL-coefficient $\in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$

8 Discussion

The goal of this paper was to generate convincing news. The experiments in section 7 indicated that the classifier could be convinced to some degree, that the generated news are true news. The high rewards show that the PPO-based fine-tuning indeed increase the degree to which the classifier is convinced. The baseline for the experiments is the hyperparameter setup proposed in the "Tune GPT2 to generate positive reviews" experiment [15] which itself if based on the findings in [18]. Their proposed learning rate of 0.0000141 can be slightly decreased to 0.00001 the achieve a better performance as seen in Fig. 6. The model's performance seems very sensitive to this hyperparameter so future experiments could aim to find even a better value for the learning rate. Additionally the experiments in this paper support the baseline value of 0.95 for lambda. Values between $0.9 - 1.01$ all performed similarly (see Fig. 5). Tuning lambda seems to have only little influence on the model's performance. Experiments on the Initial KL-Coefficient (Fig. 7) showed that values below the initial value of 0.2 performed well. In this particular setup 0.15 performed the best.

query	**response**	**reward**
POWERFUL! FOR,	reasons that are constitutional as well as practical, the fight lacked a sink of	1.456
WOW! STER,	who is said to be similarly unduroured by the pedestrian (07	4.069
MUSLIMS	conditions."" President Trump's recent decision to seize a certain for instance, one	4.910
WATCH: ESTABL	ISHMENT requested floor space for emergency maintenance to relieve residents from excessive vehicle load	3.439
EPIC TUCKER	, who said that after announcement of residency, no body is actually required to	3.939
This Pill Is 25,	stating," "Less tanks deaths caused from infighting among vehicles taking sides,	4.291
US Spends Over $	100 on car related city council ""concept proposal"" of electric transportation commuters	3.800
DIAMOND and SIL	VER, a district general, the police said in the revelation that he is	3.529
WOW! MAJ	political president, has said the place of signing of the agreement with the defence	5.143
GAME OVER SNOWFL	OWER, said noted that: the court packet was displayed due to a request	2.092

Table 2: A few example queries from the last batch of the baseline model, the response or output generated by the model and its achieved reward (truncated after third position after decimal point)

Despite the fact that the generator convinced the classifier, human readers would easily spot the results in Table 2 as computer-generated fake news. Most of the generated news make sense in a syntactical way but some highly lack quality when it comes to semantics. Increasing the syntactical-performance could be done by adding a grammar and spellchecker such as the Stanford Parser [2] to the reward function or training an additional grammar-discriminator. Increasing the semantical performance of language models is an unsolved problem in NLP. They are plenty of leverages that could be applied. One could be the use of knowledge graphs as in [17]. Another could be the improvement of the language representation as in [7]. Nonetheless all these heuristics or machine learning approaches are currently far from reaching the quality of a feedback from a human labeler. So future works could further investigate how human judges can be more efficiently involved in the process of training NP models based on the findings in [18].

List of Abbreviations

CSO Clipped Surrogate Objective
NLP Natural Language Processing
PPO Proximal Policy Optimization
SGD stochastic gradient descent
TRPO Trust Region Policy Optimization

References

1. Huggingface's Transformers, https://huggingface.co/transformers/
2. PPO2 — Stable Baselines 2.10.2a1 documentation, https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html
3. Bolukbasi, T., Chang, K.W., Zou, J., Saligrama, V., Kalai, A.: Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings, http://arxiv.org/abs/1607.06520
4. Clark, K., Luong, M.T., Le, Q.V., Manning, C.D.: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators, http://arxiv.org/abs/2003.10555
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, http://arxiv.org/abs/1810.04805
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Nets **27**, https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html
7. He, P., Liu, X., Gao, J., Chen, W.: DeBERTa: Decoding-enhanced BERT with Disentangled Attention, http://arxiv.org/abs/2006.03654
8. Langford, J.: Approximately Optimal Approximate Reinforcement Learning
9. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning, http://arxiv.org/abs/1602.01783

[2] https://nlp.stanford.edu/software/lex-parser.shtml

10. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language Models are Unsupervised Multitask Learners p. 24
11. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms, http://arxiv.org/abs/1707.06347
12. Sutton, R.S., Barto, A.G.: Reinforcement Learning, Second Edition: An Introduction. Bradford Books, second edition edn.
13. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, u., Polosukhin, I.: Attention is All you Need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 5998–6008. Curran Associates, Inc., http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
14. Wei, J., Zou, K.: EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks, http://arxiv.org/abs/1901.11196
15. von Werra, L.: Transformer Reinforcement Learning (trl), https://lvwerra.github.io/trl//
16. Wilson, M.: What is the way to understand Proximal Policy Optimization Algorithm in RL?, https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl
17. Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M., Liu, Q.: ERNIE: Enhanced Language Representation with Informative Entities, http://arxiv.org/abs/1905.07129
18. Ziegler, D.M., Stiennon, N., Wu, J., Brown, T.B., Radford, A., Amodei, D., Christiano, P., Irving, G.: Fine-Tuning Language Models from Human Preferences, http://arxiv.org/abs/1909.08593