AN ACTOR-CRITIC ALGORITHM FOR SEQUENCE PREDICTION

Dzmitry Bahdanau **Philemon Brakel**
Kelvin Xu **Anirudh Goyal**
Université de Montréal

Ryan Lowe **Joelle Pineau**[*]
McGill University

Aaron Courville[†]
Université de Montréal

Yoshua Bengio[*]
Université de Montréal

ABSTRACT

We present an approach to training neural networks to generate sequences using actor-critic methods from reinforcement learning (RL). Current log-likelihood training methods are limited by the discrepancy between their training and testing modes, as models must generate tokens conditioned on their previous guesses rather than the ground-truth tokens. We address this problem by introducing a *critic* network that is trained to predict the value of an output token, given the policy of an *actor* network. This results in a training procedure that is much closer to the test phase, and allows us to directly optimize for a task-specific score such as BLEU. Crucially, since we leverage these techniques in the supervised learning setting rather than the traditional RL setting, we condition the critic network on the ground-truth output. We show that our method leads to improved performance on both a synthetic task, and for German-English machine translation. Our analysis paves the way for such methods to be applied in natural language generation tasks, such as machine translation, caption generation, and dialogue modelling.

1 INTRODUCTION

In many important applications of machine learning, the task is to develop a system that produces a sequence of discrete tokens given an input. Recent work has shown that recurrent neural networks (RNNs) can deliver excellent performance in many such tasks when trained to predict the next output token given the input and previous tokens. This approach has been applied successfully in machine translation (Sutskever et al., 2014; Bahdanau et al., 2015), caption generation (Kiros et al., 2014; Donahue et al., 2015; Vinyals et al., 2015; Xu et al., 2015; Karpathy & Fei-Fei, 2015), and speech recognition (Chorowski et al., 2015; Chan et al., 2015).

The standard way to train RNNs to generate sequences is to maximize the log-likelihood of the "correct" token given a history of the previous "correct" ones, an approach often called *teacher forcing*. At evaluation time, the output sequence is often produced by an approximate search for the most likely candidate according to the learned distribution. During this search, the model is conditioned on its own guesses, which may be incorrect and thus lead to a compounding of errors (Bengio et al., 2015). This can become especially problematic for longer sequences. Due to this discrepancy between training and testing conditions, it has been shown that maximum likelihood training can be suboptimal (Bengio et al., 2015; Ranzato et al., 2015). In these works, the authors argue that the network should be trained to continue generating correctly given the outputs already produced by the model, rather than the *ground-truth* reference outputs from the data. This gives rise to the challenging problem of determining the target for the next network output. Bengio et al. (2015) use the token k from the ground-truth answer as the target for the network at step k, whereas Ranzato et al. (2015) rely on the REINFORCE algorithm (Williams, 1992) to decide whether or not the tokens

[*]CIFAR Senior Fellow
[†]CIFAR Fellow

from a sampled prediction lead to a high task-specific score, such as BLEU (Papineni et al., 2002) or ROUGE (Lin & Hovy, 2003).

In this work, we propose and study an alternative procedure for training sequence prediction networks that aims to directly improve their test time metrics (which are typically not the log-likelihood). In particular, we train an additional network called the *critic* to output the *value* of each token, which we define as the expected task-specific score that the network will receive if it outputs the token and continues to sample outputs according to its probability distribution. Furthermore, we show how the predicted values can be used to train the main sequence prediction network, which we refer to as the *actor*. The theoretical foundation of our method is that, under the assumption that the critic computes exact values, the expression that we use to train the actor is an unbiased estimate of the gradient of the expected task-specific score.

Our approach draws inspiration and borrows the terminology from the field of reinforcement learning (RL) (Sutton & Barto, 1998), in particular from the actor-critic approach (Sutton, 1984; Sutton et al., 1999; Barto et al., 1983). RL studies the problem of acting efficiently based only on weak supervision in the form of a reward given for some of the agent's actions. In our case, the reward is analogous to the task-specific score associated with a prediction. However, the tasks we consider are those of *supervised learning*, and we make use of this crucial difference by allowing the critic to *use the ground-truth answer as an input*. In other words, the critic has access to a sequence of expert actions that are known to lead to high (or even optimal) returns. To train the critic, we adapt the temporal difference methods from the RL literature (Sutton, 1988) to our setup. While RL methods with non-linear function approximators are not new (Tesauro, 1994; Miller et al., 1995), they have recently surged in popularity, giving rise to the field of 'deep RL' (Mnih et al., 2015). We show that some of the techniques recently developed in deep RL, such as having a *target network*, may also be beneficial for sequence prediction.

The contributions of the paper can be summarized as follows: 1) we describe how RL methodology like the actor-critic approach can be applied to supervised learning problems with structured outputs; and 2) we investigate the performance and behavior of the new method on both a synthetic task and a real-world task of machine translation, demonstrating the improvements over maximum-likelihood and REINFORCE brought by the actor-critic training.

2 BACKGROUND

We consider the problem of learning to produce an output sequence $Y = (y_1, \ldots, y_T)$, $y_t \in \mathcal{A}$ given an input X, where $\mathcal{A}$ is the alphabet of output tokens. We will often use notation $Y_{f \ldots l}$ to refer to subsequences of the form $(y_f, \ldots, y_l)$. Two sets of input-output pairs (X, Y) are assumed to be available for both training and testing. The trained predictor h is evaluated by computing the average task-specific score $R(\hat{Y}, Y)$ on the test set, where $\hat{Y} = h(X)$ is the prediction.

Recurrent neural networks A recurrent neural network (RNN) produces a sequence of state vectors $(s_1, \ldots, s_T)$ given a sequence of input vectors $(e_1, \ldots, e_T)$ by starting from an initial s_0 state and applying T times the transition function f: $s_t = f(s_{t-1}, e_t)$. Popular choices for the mapping f are the Long Short-Term Memory (Hochreiter & Schmidhuber, 1997) and the Gated Recurrent Units (Cho et al., 2014), the latter of which we use for our models.

To build a probabilistic model for sequence generation with an RNN, one adds a stochastic output layer g (typically a softmax for discrete outputs) that generates outputs $y_t \in \mathcal{A}$ and can feed these outputs back by replacing them with their embedding $e(y_t)$:

$$y_t \sim g(s_{t-1}) \tag{1}$$

$$s_t = f(s_{t-1}, e(y_t)). \tag{2}$$

Thus, the RNN defines a probability distribution $p(y_t|y_1, \ldots, y_{t-1})$ of the next output token y_t given the previous tokens $(y_1, \ldots, y_{t-1})$. Upon adding a special end-of-sequence token $\emptyset$ to the alphabet $\mathcal{A}$, the RNN can define the distribution $p(Y)$ over all possible sequences as $p(Y) = p(y_1)p(y_2|y_1) \ldots p(y_T|y_1, \ldots, y_{T-1})p(\emptyset|y_1, \ldots, y_T)$.

RNNs for sequence prediction To use RNNs for sequence prediction, they must be augmented to generate Y conditioned on an input X. The simplest way to do this is to start with an initial state $s_0 = s_0(X)$ (Sutskever et al., 2014; Cho et al., 2014). Alternatively, one can encode X as a variable-length sequence of vectors $(h_1, \dots, h_L)$ and condition the RNN on this sequence using an attention mechanism. In our models, the sequence of vectors is produced by either a bidirectional RNN (Schuster & Paliwal, 1997) or a convolutional encoder (Rush et al., 2015).

We use a *soft* attention mechanism (Bahdanau et al., 2015) that computes a weighted sum of a sequence of vectors. The attention weights determine the relative importance of each vector. More formally, we consider the following equations for RNNs with attention:

$$y_t \sim g(s_{t-1}, c_{t-1}) \tag{3}$$

$$s_t = f(s_{t-1}, c_{t-1}, e(y_t)) \tag{4}$$

$$\alpha_t = \beta(s_t, (h_1, \dots, h_L)) \tag{5}$$

$$c_t = \sum_{j=1}^{L} \alpha_{t,j} h_j \tag{6}$$

where β is the attention mechanism that produces the attention weights α_t and c_t is the context vector, or 'glimpse', for time step t. The attention weights are computed by an MLP that takes as input the current RNN state and each individual vector to focus on. The weights are typically (as in our work) constrained to be positive and sum to 1 by using the softmax function.

A conditioned RNN can be trained for sequence prediction by gradient ascent on the log-likelihood $\log p(Y|X)$ for the input-output pairs (X, Y) from the training set. To produce a prediction $\hat{Y}$ for a test input sequence X, an approximate beam search for the maximum of $p(\cdot|X)$ is usually conducted. During this search the probabilities $p(\cdot|\hat{y}_1, \dots, \hat{y}_{t-1})$ are considered, where the previous tokens $\hat{y}_1, \dots, \hat{y}_{t-1}$ comprise a candidate beginning of the prediction $\hat{Y}$.

Algorithm 1 Actor-Critic Training for Sequence Prediction

```
Require: A critic $\hat{Q}(a; \hat{Y}_{1...t}, Y)$ and an actor $p(a|\hat{Y}_{1...t}, X)$ with weights $\phi$ and $\theta$ respectively.
1: Initialize delayed actor $p^{'}$ and target critic $\hat{Q}'$ with same weights: $\theta' = \theta$, $\phi' = \phi$.
2: while Not Converged do
3:     Receive a random example $(X, Y)$.
4:     Generate a sequence of actions $\hat{Y}$ from $p^{'}$.
5:     Compute targets for the critic
           $$q_t = r_t(\hat{y}_t; \hat{Y}_{1...t-1}, Y) + \sum_{a \in \mathcal{A}} p^{'}(a|\hat{Y}_{1...t}, X)\hat{Q}'(a; \hat{Y}_{1...t}, Y)$$
6:     Update the critic weights $\phi$ using the gradient
           $$\frac{d}{d\phi}\left(\sum_{t=1}^{T}\left(\hat{Q}(\hat{y}_t; \hat{Y}_{1...t-1}, Y) - q_t\right)^2 + \lambda C\right)$$
7:     Update actor weights $\theta$ using the following gradient estimate
           $$\frac{d\widehat{V(X,Y)}}{d\theta} = \sum_{t=1}^{T}\sum_{a \in \mathcal{A}} \frac{dp(a|\hat{Y}_{1...t-1}, X)}{d\theta}\hat{Q}(a; \hat{Y}_{1...t-1}, Y)$$
8:     Update delayed actor and target critic, with a constant $\gamma \ll 1$:
           $$\theta' = \gamma_\theta\theta + (1-\gamma_\theta)\theta',\ \phi' = \gamma_\phi\phi + (1-\gamma_\phi)\phi'$$
9: end while
```

Value functions We view the conditioned RNN as a stochastic *policy* that generates *actions* and receives the task score (e.g., BLEU score) as the *return*. We furthermore consider the case when

Algorithm 2 Complete Actor-Critic Algorithm for Sequence Prediction

```
1: Initialize critic $\hat{Q}(a; \hat{Y}_{1...t}, Y)$ and actor $p(a|\hat{Y}_{1...t}, X)$ with random weights $\phi$ and $\theta$ respectively.
2: Pre-train the actor to predict $y_{t+1}$ given $Y_{1...t}$ by maximizing $\log p(y_{t+1}|Y_{1...t}, X)$.
3: Pre-train the critic to estimate $Q$ by running Algorithm 1 with fixed actor.
4: Run Algorithm 1.
```

the return R is partially received at the intermediate steps in the form of *rewards* r_t: $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t; \hat{Y}_{1...t-1}, Y)$. This is more general than the case of receiving the full return at the end of the sequence, as we can simply define all rewards other than r_T to be zero. Receiving intermediate rewards may ease the learning for the critic, and we use *reward shaping* as explained in Section 3. Given the policy, possible actions and reward function, the *value* represents the expected future return as a function of the current state of the system, which in our case is uniquely defined by the sequence of actions taken so far, $\hat{Y}_{1...t-1}$.

We define the value of an unfinished prediction $\hat{Y}_{1...t}$ as follows:

$$V(\hat{Y}_{1...t}; X, Y) = \mathop{\mathbb{E}}_{\hat{Y}_{t+1...T} \sim p(.|\hat{Y}_{1...t}, X)} \sum_{\tau=t+1}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1...\tau-1}, Y).$$

We define the value of a candidate next token a for an unfinished prediction $\hat{Y}_{1...t-1}$ as the expected future return after generating token a:

$$Q(a; \hat{Y}_{1...t-1}, X, Y) = \mathop{\mathbb{E}}_{\hat{Y}_{t+1...T} \sim p(.|\hat{Y}_{1...t-1}a, X)} \left(r_t(a; \hat{Y}_{1...t-1}, Y) + \sum_{\tau=t+1}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1...t-1} a \hat{Y}_{t+1...\tau}, Y) \right).$$

We will refer to the candidate next tokens as *actions*. For notational simplicity, we henceforth drop X and Y from the signature of p, V, Q, R and r_t, assuming it is clear from the context which of X and Y is meant. We will also use V without arguments for the expected reward of a random prediction.

3 Actor-Critic for Sequence Prediction

Let θ be the parameters of the conditioned RNN, which we will also refer to as the *actor*. Our training algorithm is based on the following way of rewriting the gradient of the expected return $\frac{dV}{d\theta}$:

$$\frac{dV}{d\theta} = \mathop{\mathbb{E}}_{\hat{Y} \sim p(\hat{Y}|X)} \sum_{t=1}^{T} \sum_{a \in \mathcal{A}} \frac{dp(a|\hat{Y}_{1...t-1})}{d\theta} Q(a; \hat{Y}_{1...t-1}). \tag{7}$$

This equality is known in RL under the names policy gradient theorem (Sutton et al., 1999) and stochastic actor-critic (Sutton, 1984). [1] Note that we use the probability rather than the log probability in this formula (which is more typical in RL applications) as we are summing over actions rather than taking an expectation. Intuitively, this equality corresponds to increasing the probability of actions that give high values, and decreasing the probability of actions that give low values. Since this gradient expression is an expectation, it is trivial to build an unbiased estimate for it:

$$\widehat{\frac{dV}{d\theta}} = \sum_{k=1}^{M} \sum_{t=1}^{T} \sum_{a \in \mathcal{A}} \frac{dp(a|\hat{Y}^k_{1...t-1})}{d\theta} Q(a; \hat{Y}^k_{1...t-1}) \tag{8}$$

where $\hat{Y}^k$ are M random samples from $p(\hat{Y})$. By replacing Q with a parameteric estimate $\hat{Q}$ one can obtain a biased estimate with relatively low variance. The parameteric estimate $\hat{Q}$ is called the *critic*.

The above formula is similar in spirit to the REINFORCE learning rule that Ranzato et al. (2015) use in the same context:

$$\widehat{\frac{dV}{d\theta}} = \sum_{k=1}^{M} \sum_{t=1}^{T} \frac{d \log p(\hat{y}^k_t|\hat{Y}^k_{1...t-1})}{d\theta} \left[\sum_{\tau=t}^{T} r_\tau(\hat{y}^k_\tau; \hat{Y}^k_{1...\tau-1}) - b_t(X) \right], \tag{9}$$

[1] We also provide a simple self-contained proof of Equation (7) in Supplementary Material.

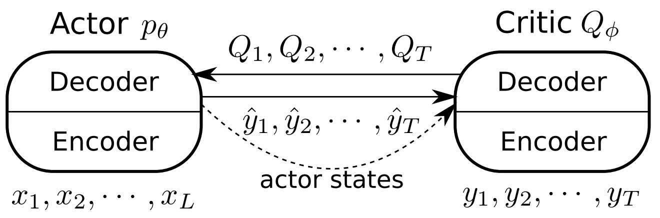

Figure 1: Both the actor and the critic are encoder-decoder networks. The actor receives an input sequence X and produces samples $\hat{Y}$ which are evaluated by the critic. The critic takes in the ground-truth sequence Y as input to the encoder, and takes the input summary (calculated using an attention mechanism) and the actor's prediction $\hat{y}_t$ as input at time step t of the decoder. The values $Q_1, Q_2, \cdots, Q_T$ computed by the critic are used to approximate the gradient of the expected returns with respect to the parameters of the actor. This gradient is used to train the actor to optimize these expected task specific returns (e.g., BLEU score). The critic may also receive the hidden state activations of the actor as input.

where the scalar $b_t(X)$ is called *baseline* or *control variate*. The difference is that in REINFORCE the inner sum over all actions is replaced by its 1-sample estimate, namely $\frac{d \log p(\hat{y}_t|\hat{Y}_{1\ldots t-1})}{d\theta} Q(\hat{y}_t; \hat{Y}_{1\ldots t-1})$, where the log probability $\frac{d \log p(\hat{y}_t|\ldots)}{d\theta} = \frac{1}{p(\hat{y}_t|\ldots)} \frac{dp(\hat{y}_t|\ldots)}{d\theta}$ is introduced to correct for the sampling of $\hat{y}_t$. Furthermore, instead of the value $Q(\hat{y}_t; \hat{Y}_{1\ldots t-1})$, REINFORCE uses the cumulative reward $\sum_{\tau=t}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1\ldots\tau-1})$ following the action $\hat{y}_t$, which again can be seen as a 1-sample estimate of Q. Due to these simplifications and the potential high variance in the cumulative reward, the REINFORCE gradient estimator also has very high variance. In order to improve upon it, we consider the actor-critic estimate from Equation 8, which has a lower variance at the cost of significant bias, since the critic is not perfect and trained simultaneously with the actor. The success depends on our ability to control the bias by designing the critic network and using an appropriate training criterion for it.

To implement the critic, we propose to use a separate RNN parameterized by ϕ. The critic RNN is run in parallel with the actor, consumes the tokens $\hat{y}_t$ that the actor outputs and produces the estimates $\hat{Q}(a; \hat{Y}_{1\ldots t})$ for all $a \in \mathcal{A}$. A key difference between the critic and the actor is that the correct answer Y is given to the critic as an input, similarly to how the actor is conditioned on X. Indeed, the return $R(\hat{Y}, Y)$ is a deterministic function of Y, and we argue that using Y to compute $\hat{Q}$ should be of great help. We can do this because the values are only required during training and we do not use the critic at test time. We also experimented with providing the actor states s_t as additional inputs to the critic. See Figure 1 for a visual representation of our actor-critic architecture.

Temporal-difference learning A crucial component of our approach is *policy evaluation*, that is the training of the critic to produce useful estimates of $\hat{Q}$. With a naïve Monte-Carlo method, one could use the future return $\sum_{\tau=t}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1\ldots\tau-1})$ as a target to $\hat{Q}(\hat{y}_t; \hat{Y}_{1\ldots t-1})$, and use the critic parameters ϕ to minimize the square error between these two values. However, like with REINFORCE, using such a target yields to very high variance which quickly grows with the number of steps T. We use a temporal difference (TD) method for policy evaluation (Sutton, 1988). Namely, we use the right-hand side $q_t = r_t(\hat{y}_t; \hat{Y}_{1\ldots t-1}) + \sum_{a \in A} p(a|\hat{Y}_{1\ldots t})\hat{Q}(a; \hat{Y}_{1\ldots t})$ of the Bellman equation as the target for the left-hand $\hat{Q}(\hat{y}_t; \hat{Y}_{1\ldots t-1})$.

Applying deep RL techniques It has been shown in the RL literature that if $\hat{Q}$ is non-linear (like in our case), the TD policy evaluation might diverge (Tsitsiklis & Van Roy, 1997). Previous work has shown that this problem can be alleviated by using an additional *target network* $\hat{Q}'$ to compute q_t, which is updated less often and/or more slowly than $\hat{Q}$. Similarly to (Lillicrap et al., 2015), we update the parameters ϕ' of the target critic by linearly interpolating them with the parameters of the trained one. Attempts to remove the target network by propagating the gradient through q_t resulted in a lower square error $(\hat{Q}(\hat{y}_t; \hat{Y}_{1\ldots T}) - q_t)^2$, but the resulting $\hat{Q}$ values proved very unreliable as training signals for the actor.

The fact that both actor and critic use outputs of each other for training creates a potentially dangerous feedback loop. To address this, we sample predictions from a *delayed actor* (Lillicrap et al., 2015), whose weights are slowly updated to follow the actor that is actually trained.

Dealing with large action spaces One of the challenges of our work is that the action space is very large (as is typically the case in NLP tasks with large vocabularies). This can be alleviated by putting constraints on the critic values for actions that are rarely sampled. We found experimentally that shrinking the values of these rare actions is necessary for the algorithm to converge. Specifically, we add a term C to the optimization objective which drives all value predictions of the critic closer to their mean:

$$C = \sum_{a} \left(\hat{Q}(a; \hat{Y}_{1\ldots t-1}) - \frac{1}{|\mathcal{A}|} \sum_{b} \hat{Q}(b; \hat{Y}_{1\ldots t-1}) \right)^2 , \tag{10}$$

This corresponds to penalizing the *variance* of the outputs of the critic. Without this penalty the values of rare actions can be severely overestimated, which biases the gradient estimates and can cause divergence. A similar trick was used in the context of learning simple algorithms with Q-learning (Zaremba et al., 2015).

Reward shaping While we are ultimately interested in the maximization of the score of a complete prediction, simply awarding this score at the last step provides a very sparse training signal for the critic. For this reason we use potential-based reward shaping with potentials $\Phi(\hat{Y}_{1\ldots t}) = R(\hat{Y}_{1\ldots t})$ for incomplete sequences and $\Phi(\hat{Y}) = 0$ for complete ones (Ng et al., 1999). Namely, for a predicted sequence $\hat{Y}$ we compute score values for all prefixes to obtain the sequence of scores $(R(\hat{Y}_{1\ldots 1}), R(\hat{Y}_{1\ldots 2}), \ldots, R(\hat{Y}_{1\ldots T}))$. The difference between the consecutive pairs of scores is then used as the reward at each step: $r_t(\hat{y}_t; \hat{Y}_{1\ldots t-1}) = R(\hat{Y}_{1\ldots t}) - R(\hat{Y}_{1\ldots t-1})$. Using the shaped reward r_t instead of awarding the whole score R at the last step does not change the optimal policy (Ng et al., 1999).

Putting it all together Algorithm 1 describes the complete training details. Starting training with a randomly initialized actor and critic would be problematic, because neither the actor nor the critic would provide adequate training signals for one another. The actor would sample completely random predictions that receive very little reward, thus providing a very weak training signal for the critic. A random critic would be similarly useless for training the actor. Motivated by these considerations, we pre-train the actor using standard log-likelihood training. Furthermore, we pre-train the critic by feeding it samples from the pre-trained actor, while the actor's parameters are frozen. The complete training procedure including pre-training is described by Algorithm 2.

4 Related Work

In other recent RL-inspired work on sequence prediction, Ranzato et al. (2015) trained a translation model by gradually transitioning from maximum likelihood learning into optimizing BLEU or ROUGE scores using the REINFORCE algorithm. However, REINFORCE is known to have very high variance and does not exploit the availability of the ground-truth like the critic network does. The approach also relies on a curriculum learning scheme. Standard value-based RL algorithms like SARSA and OLPOMDP have also been applied to structured prediction (Maes et al., 2009). Again, these systems do not use the ground-truth for value prediction.

Imitation learning has also been applied to structured prediction (Vlachos, 2012). Methods of this type include the SEARN (Daumé Iii et al., 2009) and DAGGER (Ross et al., 2010) algorithms. These methods rely on an *expert policy* to provide action sequences that the policy learns to imitate. Unfortunately, it's not always easy or even possible to construct an expert policy for a task-specific score. In our approach, the critic plays a role that is similar to the expert policy, but is learned without requiring prior knowledge about the task-specific score. The recently proposed 'scheduled sampling' (Bengio et al., 2015) can also be seen as imitation learning. In this method, ground-truth tokens are occasionally replaced by samples from the model itself during training. A limitation is that the token k for the ground-truth answer is used as the target at step k, which might not always be the optimal strategy.

There are also approaches that aim to approximate the gradient of the expected score. One such approach is 'Direct Loss Minimization' (Hazan et al., 2010) in which the inference procedure is adapted to take both the model likelihood and task-specific score into account. Another popular approach is to replace the domain over which the task score expectation is defined with a small subset of it, as is done in Minimum (Bayes) Risk Training (Goel & Byrne, 2000; Shen et al., 2015; Och, 2003). This small subset is typically an n-best list or a sample (like in REINFORCE) that may or may not include the ground-truth as well. None of these methods provide intermediate targets for the actor during training.

Another recently proposed method is to optimize a global sequence cost with respect to the selection and pruning behavior of the beam search procedure itself (Wiseman & Rush, 2016). This method follows the more general strategy called 'learning as search optimization' (Daumé III & Marcu, 2005). This is an interesting alternative to our approach; however, it is designed specifically for the precise inference procedure involved.

5 Experiments

To validate our approach, we performed two experiments [2]. First, we trained the proposed model to recover strings of natural text from their corrupted versions. Specifically, we consider each character in a natural language corpus and with some probability replace it with a random character. We call this synthetic task *spelling correction*. A desirable property of this synthetic task is that data is essentially infinite and overfitting is no concern. Our second experiment is done on the task of automatic machine translation.

In addition to maximum likelihood and actor-critic training we implemented two versions of the REINFORCE gradient estimator. In the first version, we use a linear baseline network that takes the actor states as input, exactly as in (Ranzato et al., 2015). We also propose a novel extension of REINFORCE that leverages the extra information available in the ground-truth output Y. Specifically, we use the $\hat{Q}$ estimates produced by the critic network as the baseline for the REINFORCE algorithm. The motivation behind this approach is that using the ground-truth output should produce a better baseline that lowers the variance of REINFORCE, resulting in higher task-specific scores. We refer to this method as REINFORCE-critic.

5.1 Spelling Correction

We use text from the One Billion Word dataset for the spelling correction task (Chelba et al., 2013), which has pre-defined training and testing sets. The training data was abundant, and we never used any example twice. We evaluate trained models on a section of the test data that comprises 6075 sentences. To speed up experiments, we clipped all sentences to the first 10 or 30 characters.

For the spelling correction actor network, we use an RNN with 100 Gated Recurrent Units (GRU) and a bidirectional GRU network for the encoder. We use the same attention mechanism as proposed in (Bahdanau et al., 2015), which effectively makes our actor network a smaller version of the model used in that work. For the critic network, we employed a model with the same architecture as the actor. We provide the states of the actor as additional inputs for the critic.

We use character error rate (CER) to measure performance on the spelling task, which we define as the ratio between the total of Levenshtein distances between predictions and ground-truth outputs and the total length of the ground-truth outputs. This is a corpus-level metric for which a lower value is better. We use it as the return by negating per-sentence ratios. At the evaluation time greedy search is used to extract predictions from the model.

We use the ADAM optimizer (Kingma & Ba, 2015) to train all the networks with the parameters recommended in the original paper, with the exception of the scale parameter α. The latter is first set to 10^{-3} and then annealed to 10^{-4} for log-likelihood training. For the pre-training stage of the actor-critic, we use $\alpha = 10^{-3}$ and decrease it to 10^{-4} for the joint actor-critic training. We pretrain the actor until its score on the development set stops improving. We pretrain the critic until its

[2] The source code is available at `https://github.com/rizar/actor-critic-public`

TD error stabilizes[3]. We used $M = 1$ sample for both actor-critic and REINFORCE. For exact hyperparameter settings we refer the reader to Appendix A.

We start REINFORCE training from a pretrained actor, but we do not use the curriculum learning employed in MIXER. The critic is trained in the same way for both REINFORCE and actor-critic, including the pretraining stage. We report results obtained with the reward shaping described in Section 3, as we found that it slightly improves REINFORCE performance.

Table 1 presents our results on the spelling correction task. We observe an improvement in CER over log-likelihood training for all four settings considered. In three out of 4 cases the actor-critic training results in a better CER than REINFORCE with critic. In the 4th case, actor-critic and REINFORCE with critic have similar performance.

5.2 Machine Translation

For the machine translation experiment, we use data from the German-English machine translation track of the IWSLT 2014 evaluation campaign (Cettolo et al., 2014), as used in Ranzato et al. (2015), and closely follow the pre-processing described in that work. The training data comprises about 153,000 German-English sentence pairs, and the sizes of validation and tests set were 6,969 and 6,750, respectively. We limited the number of words in the English and German vocabularies to the 22,822 and 32,009 most frequent words, respectively, and replaced all other words with a special token. The maximum sentence length in our dataset was 50.

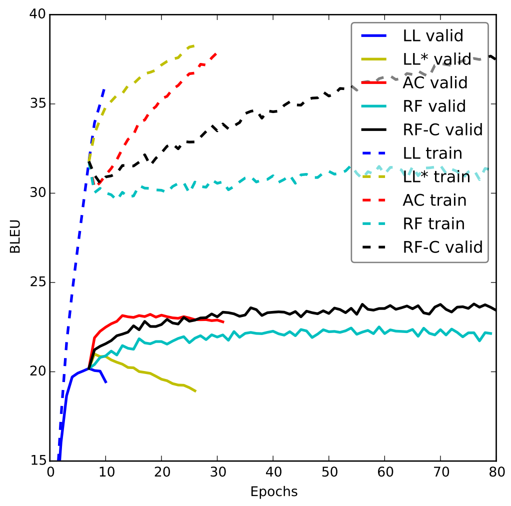

Figure 2: Progress of log-likelihood (LL), REINFORCE (RF) and actor-critic (AC) training in terms of BLEU score on the training (train) and validation (valid) datasets. LL* stands for the annealing phase of log-likelihood training. The curves start from the epoch of log-likelihood pretraining from which the parameters were initialized.

We use a convolutional encoder in the actor to make our results more comparable with Ranzato et al. (2015). For the same reason, we use 256 hidden units in all machine translation networks. For the machine translation critic, we replaced the convolutional network with a bidirectional GRU network. In preliminary experiments we found that for machine translation providing actor states as inputs to the critic has a negative impact on the results.

The return is defined as a smoothed and rescaled version of the BLEU score. Specifically, we start all n-gram counts from 1 instead of 0, and multiply the resulting score by the length of the ground-truth translation. Smoothing is a common practice when sentence-level BLEU score is considered, and it has been used to apply REINFORCE in similar settings (Ranzato et al., 2015).

For training this model we mostly used the same hyperparameter values as in the spelling correction experiments, with a few difference highlighted in Appendix A.

For decoding we used greedy search and beam search with a beam size of 10. We found that penalizing candidate sentences that are too short is required to obtain the best results. Similarly to (Hannun et al., 2014), we subtracted ρT from the negative log-likelihood of each candidate sentence, where T is the candidate's length, and ρ is a hyperparameter tuned on the validation set.

The results are summarized in Table 2. We report a significant improvement of 2.3 BLEU points over the log-likelihood baseline when greedy search is used for decoding. Surprisingly, the best performing method is REINFORCE with critic, with an additional 0.6 BLEU point advantage over

[3]A typical behaviour for TD error was to grow at first and then start decreasing slowly. We found that stopping pretraining shortly after TD error stops growing leads to good results.

Table 1: Character error rate of different methods on the spelling correction task. In the table L is the length of input strings, η is the probability of replacing a character with a random one.

Setup	Character Error Rate		
	Log-likelihood	Actor-Critic	REINFORCE with critic
$L = 10, \eta = 0.3$	18.6	**17.2**	17.8
$L = 30, \eta = 0.3$	18.5	**17.3**	18.2
$L = 10, \eta = 0.5$	38.2	**35.9**	**35.8**
$L = 30, \eta = 0.5$	41.3	**37.0**	37.6

Table 2: Our machine translation results compared to the previous work by Ranzato et al. The abbreviations LL, RF, RF-C, AC stand for log-likelihood, REINFORCE, REINFORCE with critic and actor-critic training respectively. The asterisk identifies results from (Ranzato et al., 2015). The numbers reported with $\leq$ were approximately read from Figure 6 of (Ranzato et al., 2015)

Decoding method	Model					
	LL*	MIXER*	LL	RF	RF-C	AC
greedy search	17.74	20.73	19.33	20.92	**22.24**	21.66
beam search	$\leq$ 20.3	$\leq$ 21.9	21.46	21.35	**22.58**	22.45

the actor-critic. When beam-search is used, the ranking of the compared approaches is the same, but the margin between the proposed methods and log-likelihood training becomes smaller. The final performances of the actor-critic and the REINFORCE-critic with greedy search are also 0.7 and 1.3 BLEU points respectively better than what Ranzato et al. (2015) report for their MIXER approach. This comparison should be treated with caution, because our log-likelihood baseline is 1.6 BLEU points stronger than its equivalent from (Ranzato et al., 2015). The performance of REINFORCE with a simple baseline matches the score reported for MIXER in Ranzato et al. (2015).

To better understand the machine translation results we provide the learning curves for the considered approaches in Figure 2. We can clearly see that the training methods that use generated predictions have a strong regularization effect — that is, better progress on the validation set in exchange for slower or negative progress on the training set. The effect is stronger for both REINFORCE varieties, especially for the one without a critic. The actor-critic training does a much better job of fitting the training set than REINFORCE and is the only method except log-likelihood that shows a clear overfitting, which is a healthy behaviour for such a small dataset.

Finally, we performed an ablation study for the actor-critic on the machine translation task. We found that using a target network was crucial; while the joint actor-critic training was still progressing with $\gamma_\theta = 0.1$, with $\gamma_\theta = 1.0$ it did not work at all. Similarly important was the value penalty described in Equation (10). We found that good values of the λ coefficient were in the range $[10^{-3}, 10^{-6}]$. Other techniques, such as reward shaping and a delayed actor, brought moderate performance gains. We refer the reader to Appendix A for more details.

6 Discussion

We proposed an actor-critic approach to sequence prediction. Our method takes the task objective into account during training and uses the ground-truth output to aid the critic in its prediction of intermediate targets for the actor. We showed that our method leads to significant improvements over maximum likelihood training on both a synthetic task and a machine translation benchmark. Compared to REINFORCE training on machine translation, actor-critic fits the training data much faster, although we were able to significantly reduce the gap in the training speed and achieve a better test error using our critic network as the baseline for REINFORCE.

One interesting observation we made from the machine translation results is that the training methods that use generated predictions have a strong regularization effect. Our understanding is that conditioning on the sampled outputs effectively increases the diversity of training data. This phenomenon makes it harder to judge whether the actor-critic training meets our expectations, because a noisier gradient estimate yielded a better test set performance. We argue that the strong spelling correction results obtained on a virtually infinite dataset in conjuction with better machine translation performance on the training set provide convincing evidence that the actor-training can be effective. In future work we will consider larger machine translation datasets.

We ran into several optimization issues. The critic would sometimes assign very high values to actions with a very low probability according to the actor. We were able to resolve this by penalizing the critic's variance. Additionally, the actor would sometimes have trouble to adapt to the demands of the critic. We noticed that the action distribution tends to saturate and become deterministic, causing the gradient to vanish. While these issues did not prevent us from training our models, we might obtain better results by addressing them.

ACKNOWLEDGMENTS

We thank the developers of Theano (Theano Development Team, 2016) and Blocks (van Merriënboer et al., 2015) for their great work. We thank NSERC, Compute Canada, Calcul Quebéc, Canada Research Chairs, CIFAR and Samsung Institute of Advanced Techonology for their financial support.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the ICLR 2015*, 2015.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):834–846, 1983.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*, 2015.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign. In *Proc. of IWSLT*, 2014.

William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, KyungHyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *CoRR*, abs/1506.07503, 2015. URL `http://arxiv.org/abs/1506.07503`.

Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd international conference on Machine learning*, pp. 169–176. ACM, 2005.

Hal Daumé Iii, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634, 2015.

Vaibhava Goel and William J Byrne. Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135, 2000.

Awni Y Hannun, Andrew L Maas, Daniel Jurafsky, and Andrew Y Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*, 2014.

Tamir Hazan, Joseph Keshet, and David A McAllester. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pp. 1594–1602, 2010.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3128–3137, 2015.

Diederik P Kingma and Jimmy Ba. A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.

Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*, 2014.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Chin-Yew Lin and Eduard Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 71–78. Association for Computational Linguistics, 2003.

Francis Maes, Ludovic Denoyer, and Patrick Gallinari. Structured prediction with reinforcement learning. *Machine learning*, 77(2-3):271–301, 2009.

W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pp. 160–167. Association for Computational Linguistics, 2003.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics, 2002.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv preprint arXiv:1011.0686*, 2010.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. *arXiv preprint arXiv:1512.02433*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pp. 1057–1063, 1999.

Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv:1506.00619 [cs, stat]*, June 2015.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2015.

Andreas Vlachos. An investigation of imitation learning algorithms for structured prediction. In *EWRL*, pp. 143–154. Citeseer, 2012.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 2048–2057, 2015.

Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. *arXiv preprint arXiv:1511.07275*, 2015.

Table 3: Results of an ablation study. We tried varying the actor update speed γ_θ, the critic update speed γ_ϕ, the value penalty coefficient λ, whether or not reward shaping is used, whether or not temporal difference (TD) learning is used for the critic. Reported are the best training and validation BLEU score obtained in the course of the first 10 training epochs. Some of the validation scores would still improve with longer training. Greedy search was used for decoding.

γ_θ	γ_ϕ	λ	reward shaping	TD	train BLEU	valid BLEU
			baseline			
0.001	0.001	10^{-3}	yes	yes	33.73	23.16
			with different γ_ϕ			
0.001	0.01	10^{-3}	yes	yes	33.52	23.03
0.001	0.1	10^{-3}	yes	yes	32.63	22.80
0.001	1	10^{-3}	yes	yes	9.59	8.14
			with different γ_θ			
1	0.001	10^{-3}	yes	yes	32.9	22.88
			without reward shaping			
0.001	0.001	10^{-3}	no	yes	32.74	22.61
			without temporal difference learning			
0.001	0.001	10^{-3}	yes	no	23.2	16.36
			with different λ			
0.001	0.001	$3 * 10^{-3}$	yes	yes	32.4	22.48
0.001	0.001	10^{-4}	yes	yes	34.10	23.15
0.001	0.001	10^{-6}	yes	yes	35.00	23.10
0.001	0.001	10^{-8}	yes	yes	33.6	22.72
0.001	0.001	0	yes	yes	27.41	20.55

A HYPERPARAMETERS

For machine translation experiments the variance penalty coefficient λ was set to 10^{-4}, and the delay coefficients γ_θ and γ_ϕ were both set to 10^{-4}. For REINFORCE with the critic we did not use a delayed actor, i.e. γ_θ was set to 1. For the spelling correction task we used the same γ_θ and γ_ϕ but a different $\lambda = 10^{-3}$. All initial weights were sampled from a centered uniform distribution with width 0.1.

For decoding with beam search we substracted the length of a candidate times ρ from the log-likelihood cost. The exact value of ρ was selected on the validation set and was equal to 0.8 for models trained by log-likelihood and REINFORCE and to 1.0 for models trained by actor-critic and REINFORCE-critic.

For some of the hyperparameters we performed an ablation study. The results are reported in Table 3.

B GENERATED Q-VALUES

In Table B we provide an example of value predictions that the critic outputs for candidate next words. One can see that the critic has indeed learnt to assign larger values for the appropriate next words. While the critic does not always produce sensible estimates and can often predict a high return for irrelevant rare words, this is greatly reduced using the variance penalty term from Equation (10).

Figure 3: The best 3 words according to the critic at intermediate steps of generating a translation. The numbers in parentheses are the value predictions $\hat{Q}$. The German original is "über eine davon will ich hier erzählen ." The reference translation is "and there's one I want to talk about".

Word	Words with largest $\hat{Q}$
one	and(6.623) there(6.200) but(5.967)
of	that(6.197) one(5.668) &apos;s(5.467)
them	that(5.408) one(5.118) i(5.002)
i	that(4.796) i(4.629) ,(4.139)
want	want(5.008) i(4.160) &apos;t(3.361)
to	to(4.729) want(3.497) going(3.396)
tell	talk(3.717) you(2.407) to(2.133)
you	about(1.209) that(0.989) talk(0.924)
about	about(0.706) .(0.660) right(0.653)
here	.(0.498) ?(0.291) –(0.285)
.	.(0.195) there(0.175) know(0.087)
$\emptyset$	.(0.168) $\emptyset$ (-0.093) ?(-0.173)

C Proof of Equation (7)

$$\frac{dV}{d\theta} = \frac{d}{d\theta} \mathop{\mathbb{E}}_{\hat{Y}\sim p(\hat{Y})} R(\hat{Y}) = \sum_{\hat{Y}} \frac{d}{d\theta} \left[p(\hat{y}_1)p(\hat{y}_2|\hat{y}_1)\dots p(\hat{y}_T|\hat{y}_1\dots\hat{y}_{T-1})\right] R(\hat{Y}) =$$

$$\sum_{t=1}^{T}\sum_{\hat{Y}} p(\hat{Y}_{1\dots t-1})\frac{dp(\hat{y}_t|\hat{Y}_{1\dots t-1})}{d\theta} p(\hat{Y}_{t+1..T}|\hat{Y}_{1\dots t})R(\hat{Y}) =$$

$$\sum_{t=1}^{T}\sum_{\hat{Y}_{1\dots t}} p(\hat{Y}_{1\dots t-1})\frac{dp(\hat{y}_t|\hat{Y}_{1\dots t-1})}{d\theta} \sum_{\hat{Y}_{t+1\dots T}} p(\hat{Y}_{t+1..T}|\hat{Y}_{1\dots t}) \sum_{\tau=1}^{T} r_\tau(\hat{y}_\tau;\hat{Y}_{1\dots\tau-1}) =$$

$$\sum_{t=1}^{T}\sum_{\hat{Y}_{1\dots t}} p(\hat{Y}_{1\dots t-1})\frac{dp(\hat{y}_t|\hat{Y}_{1\dots t-1})}{d\theta}$$

$$\left[r_t(\hat{y}_t;\hat{Y}_{1\dots t-1}) + \sum_{\hat{Y}_{t+1\dots T}} p(\hat{Y}_{t+1..T}|\hat{Y}_{1\dots t}) \sum_{\tau=t+1}^{T} r_\tau(\hat{y}_\tau;\hat{Y}_{1\dots\tau-1})\right] =$$

$$\sum_{t=1}^{T} \mathop{\mathbb{E}}_{\hat{Y}_{1\dots t-1}\sim p(\hat{Y}_{1\dots t-1})} \sum_{a\in A} \frac{dp(a|\hat{Y}_{1\dots t-1})}{d\theta} Q(a;\hat{Y}_{1\dots t-1}) =$$

$$\mathop{\mathbb{E}}_{\hat{Y}\sim p(\hat{Y})} \sum_{t=1}^{T}\sum_{a\in\mathcal{A}} \frac{dp(a|\hat{Y}_{1\dots t-1})}{d\theta} Q(a;\hat{Y}_{1\dots t-1})$$