Reinforcement Learning for Sequence-to-Sequence Dialogue Systems

Master's Thesis

By

Fabian Retkowski

at the Interactive Systems Lab
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT)

September 28, 2020

Process Period	30. Mar 2020 - 29. Sep 2020
Matriculation Number	2157222
Reviewer	Prof. Dr. Waibel
Second Reviewer	Prof. Dr. Asfour
Advisor	Ngoc-Quan Pham

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, September 28, 2020

Location, Date

Fabian Retkowski

Abstract

English

Sequence-to-sequence (seq2seq) models offer great promise for sequence generation problems such as machine translation or dialogue generation. However, several issues have been identified, among them are exposure bias and search error. In addition, loss-evaluation mismatch emerged as a major problem. For example, conversational agents tend to generate dull and generic responses, because such sequences are of high frequency in common datasets. In recent works, reinforcement learning (RL) has been utilized to mitigate these issues. However, due to the high-dimensionality in natural language processing (NLP), the application of RL generally relies on well-trained fully supervised models and is restricted to fine-tune such models with policy gradient methods. Consequently, these approaches only lead to marginal improvements.

At the same time, recent RL methods applied to game playing achieve ground-breaking results. Ensemble methods like Rainbow, distributional RL algorithms such as IQN or FQF, and distributed agents like R2D2 are to be mentioned here. Nevertheless, they have not been applied to NLP in previous works. In this context, this thesis demonstrates the theoretical possibility of applying such advanced value-based RL methods on seq2seq models for the first time. Specifically, a seq2seq model is introduced, which is trained in a Rainbow-like setup. While such a model is practically still limited by its scalability, the thesis contributes towards a more generally applicable approach to RL in NLP which is beyond the scope of fine-tuning. For this, the thesis provides a theoretical and practical framework, a first baseline, and valuable insights by studying ablated models and different approaches for utilizing demonstration data.

Sequence-to-Sequence-Modelle (seq2seq) sind weitverbreitete Ansätze im Bereich der Sequenzgenerierung wie der maschinellen Übersetzung oder der Dialoggenerierung. Es wurden jedoch mehrere Probleme identifiziert, darunter der "Exposure Bias" und der "Search Error". Darüber zeigt sich die Diskrepanz zwischen Verlustfunktion und Testmetrik als besonders problematisch. Beispielsweise neigen Dialogsysteme dazu, uninteressante und generische Antworten zu generieren, da solche in den meisten Datensätzen hochfrequent auftreten. In neueren Arbeiten wurde Reinforcement Learning (RL) verwendet, um diesen Problemen zu begegnen. Aufgrund der hohen Dimensionalität im Bereich des Natural Language Processings (NLP) ist die Anwendung von RL jedoch im Allgemeinen auf gut-trainierte, vollständig überwachte Modelle angewiesen und beschränkt sich auf das Fine-Tuning solcher mit Policy-Gradient-Methoden. Folglich führen diese Ansätze nur zu marginalen Verbesserungen.

Gleichzeitig erzielen neuere RL-Methoden, angewandt auf Game Playing, wegweisende Ergebnisse. Ensemble-Methoden wie Rainbow, "Distributional RL"-Algorithmen wie IQN oder FQF und verteilte Agenten wie R2D2 sind hier zu nennen. Bisher wurden sie jedoch nicht auf NLP angewandt. In diesem Zusammenhang zeigt diese Arbeit erstmals die theoretische Möglichkeit auf, solche fortgeschrittenen wertbasierten RL-Methoden auf seq2seq-Modelle zu übertragen. Konkret wird ein seq2seq-Modell vorgestellt, das in einem vergleichbaren Aufbau trainiert wird wie Rainbow. Während ein solches Modell praktisch noch durch seine Skalierbarkeit begrenzt ist, ist sie ein Schritt zu einem allgemeineren Ansatz für RL in NLP, der über das Fine-Tuning von Modellen hinausgeht. Hierfür bietet die Arbeit ein theoretisches und praktisches Framework, eine erste Baseline und wertvolle Einsichten durch die Untersuchung abgetragener Modelle und verschiedener Ansätze zur Nutzung von Demonstrationsdaten.

Table of Contents

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AQL	Amortized Q-Learning
BLEU	Bilingual Evaluation Understudy
BPTT	Backpropagation Through Time
CE	Cross-Entropy
DAD	Data as Demonstrator
DQN	Deep Q-Network
DRQN	Deep Recurrent Q-Network
FQF	Fully Quantile Function
GRU	Gated Recurrent Unit
IQN	Implicit Quantile Network
IS	Importance Sampling
KL	Kullback-Leibler
LCS	Longest Common Subsequence
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MIXER	Mixed Incremental Cross-Entropy REINFORCE
MLE	Maximum Likelihood Estimation
MSE	Mean Squared Error
MTL	Multi-Task Learning
NLP	Natural Language Processing
PER	Prioritized Experience Replay
PG	Policy Gradient

QR	Quantile Regression
R2D2	Recurrent Replay Distributed DQN
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy For Gisting Evaluation
SC	Self-Critic
seq2seq	Sequence-to-Sequence
SGD	Stochastic Gradient Descent
SL	Supervised Learning
TD	Temporal Difference

List of Figures

List of Tables

1

Introduction

1.1. Motivation

With conversational agents on the rise, such as Amazon's Alexa, Google Assistant, or Microsoft's Cortana, dialogue systems and the language generation problem are receiving extensive attention. In addition, recent works have demonstrated the increasing popularity and excellent performance of sequence-to-sequence (seq2seq) models for sequence prediction. Such models can tackle a variety of problems, including machine translation, text summarization, image captioning, speech recognition, and language generation.

Nevertheless, fully supervised trained seq2seq models have several methodological weaknesses which have not been completely solved. First, such models suffer from *exposure bias* as they are usually trained with teacher forcing. In this method, the model is conditioned on ground-truth data as input instead of its own outputs. Secondly, word predictions by these models do not consider the whole sequence, which introduces *search error*. Thirdly and most importantly, the maximum likelihood estimation (MLE) objective is often used in

approximating the probability distribution $P(y|x)$, such that the likelihood of outputs given input is maximized. In many problems such as machine translation utilizing this distribution may be sufficient. However, for other problems, it differs substantially from the test objectives and the real-world goal.

It has been shown that when applied to the task of dialogue generation, the aforementioned models tend to generate highly generic, repetitive, and short-sighted responses, with "I don't know" among them. This outcome can be ascribed to the high frequency of such phrases in dialogue corpora, which are then favoured by the MLE objective.

Reinforcement learning (RL) addresses all of these issues. It allows using any function as a reward, which may include non-differentiable test metrics, human feedback, or other functions that are closer to the real-world goal. Furthermore, RL relies on its own outputs instead of ground-truth data, and it naturally incorporates future rewards, thus avoiding exposure bias and search error.

However, RL is considered sample inefficient, especially in the case of reward sparsity and large action spaces, and this is particularly true for most natural language processing (NLP) tasks, which require dealing with huge vocabulary sizes. Thus, most research in this area has focused on fine-tuning supervised models. It has, therefore, also been limited to methods that output softmax probabilities, which makes them easy to pretrain with supervised approaches. This category includes policy gradient (PG) methods and actor-critic setups. For instance, the easy-to-implement, and "plug'n'play" REINFORCE is still widely used in this area, although this algorithm is known to have severe issues, such as a time-consuming training and high variance. In contrast, value-based learning methods such as Q-learning have received little attention, since they need to predict future rewards for every single action and cannot be easily pretrained. However, value-based methods have made significant progress in recent years. Advanced methods such as Rainbow and fully quantile functions (FQFs) are state-of-the-art approaches in many fields (e.g., game-playing), significantly surpassing modern actor-critic agents, for instance asynchronous advantage actor-critic (A3C). They might overcome some caveats of Q-learning and make it a reasonable choice for the area of NLP.

1.2. Goals

The thesis' overall goal is to explore the theoretical possibilities of applying advanced value-based methods in the area of NLP. More specifically, this work investigates whether sequence-to-sequence models can be trained with state-of-the-art Rainbow, which is a Q-learning-based approach that seeks to combine several improvements made to deep Q-networks (DQNs) in recent years, including dueling nets and multi-step learning. With this objective in mind, this thesis strives to contribute to the long-term goal of applying RL in high-dimensional seq2seq problems beyond the limited scope of fine-tuning.

This overall goal encompasses the following subgoals:

- Definition of a reinforcement learning setting, which is compatible with both the sequence-to-sequence architecture and Q-learning.
- Transfer of DQN and its extensions to the seq2seq architecture.
- Evaluation of different-sized action spaces to assess the scalability.
- Ablation studies for DQN components to gain a deeper understanding of the underlying mechanisms.
- Exploration of techniques that allow utilization of demonstration data for this "Rainbow seq2seq" model.

1.3. Outline

The thesis is structured as follows. Chapter 1 introduces the subject matter and the motivation for the objective. Chapter 2 introduces the reader in dialogue systems, seq2seq architectures, and the field of reinforcement learning, especially value-based methods. Chapter 3 focuses on Rainbow, an advanced value-based RL method, and the DQN improvements it combines, which, in the context of the goal of this thesis, have to be implemented and transferred to the seq2seq architecture. Chapter 4 describes related research, while Chapter 5 presents two movie dialogue datasets commonly employed for language generation tasks. Chapter 6 works towards an RL setting, an RL method,

and pretraining techniques that allow transferring Q-learning and Rainbow's DQN extensions to seq2seq architectures. Since this research involves a non-trivial implementation, Chapter 7 points out a few selected aspects in that regard. In Chapter 8, the resulting model is evaluated with different vocabulary sizes, ablations, and different pretraining techniques on movie dialogue datasets. Finally, Chapter 9 presents the conclusions, offers a reflection on the work conducted, and proposes future work in this research area.

2

Foundations

2.1. Dialogue Systems

Dialogue systems refer to systems that can interact with humans via natural language. Gao et al. (2018) [1] describe three different problems such a system could handle:

- **Question answering:** The system needs to answer questions based on rich knowledge (e.g., texts or images) or knowledge bases (e.g., databases). These agents are sometimes called *Infobots*.
- **Task completion:** The agent should assist the user in reaching a goal through fairly short conversations.
- **Social chat or Chit-chat:** The agent needs to mimic human conversations (cf. Turing test), implying a demand for extensive, engaging, entertaining, and diverse dialogues.

Task-oriented systems are mostly implemented through modular systems consisting of units for language understanding, dialogue management, and language generation. Moreover, there is often a knowledge base to query. Conversely, *social bots* are usually unitary systems (e.g., sequence-to-sequence models, as described in Section 2.2).

2.2. Sequence-to-Sequence Models

2.2.1. Architecture

The classic *sequence-to-sequence* (seq2seq) architecture, also called encoder-decoder architecture (see Figure 2.1) was first proposed by Sutskever et al. (2014) [2]. The motivation behind this architecture is to map an input sequence (source) to an output sequence (target), both of which can be of arbitrary lengths. The architecture is composed of an encoder and a decoder. [3, pp. 390 sqq.]

- The encoder recurrent neural network (RNN) compresses the input sequence $x = (x_1, x_2, \ldots, x_{n_x})$ to a fixed-length vector C (thought vector or context vector), which is the final hidden state vector h_{n_x} of the RNN.
- The decoder's hidden state is initialized with the fixed-length vector C. The decoder RNN then generates the output sequence $y = (y_1, y_2, \ldots, y_{n_y})$.

Bahdanau et al. (2015) [4] observed a limitation of this approach: a fixed-length vector is incapable of remembering long sequences; instead, it may forget early parts. To combat this issue, the researchers introduced the attention mechanism, which was later refined by Luong et al. (2015) [5]. The general idea is to create shortcuts to the source sequence at every decoding time step, thus bypassing long in-between distances. [3, pp. 390 sqq.]

RNN encoders and decoders are typically implemented as long short-term memories (LSTMs) [6] or gated recurrent units (GRUs) [7], either unidirectional or bidirectional. The decoder usually adds another linear layer with softmax activation (the so-called "*generator*") to output a probability distribution over the vocabulary. [8]

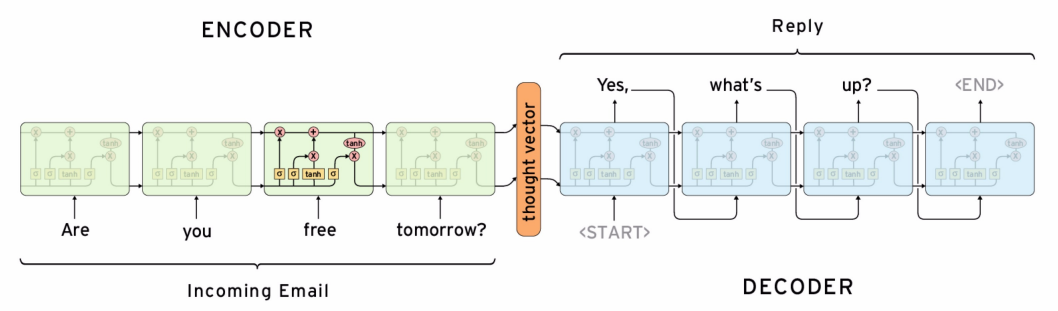


Figure 2.1: Sequence-to-sequence model, here: with LSTM units
Source: Diagram by Christopher Olah

More generally, encoder-decoder architectures may vary widely and include pointer networks [9], transformers [10] or may also be based on convolutions [11, 12, 13].

2.2.2. Training and Objective

As illustrated in Figure 2.1, the outputs of the decoder network are fed into the next sequential unit as input. Consequently, mistakes at the beginning of the sequence can lead to increasing erroneousness, which ultimately results in slow convergence. Thus, the most common algorithm for training the model is *teacher forcing*, which feeds the actual correct sequence (the targets) into the model. This algorithm allows parallelization, thereby avoiding backpropagation through time (BPTT), as it removes the necessity to wait for the sequential outputs to be used as inputs. Teacher forcing minimizes the maximum likelihood objective, which is the cross-entropy (CE) loss: [3, pp. 378 sqq.] [8, 14]

$$\begin{aligned} \mathcal{L}_{CE} = -\log p(y_1, \ldots, y_{n_y}) &= -\log \prod_{t=1}^{n_y} p(y_t | y_1, \ldots, y_{t-1}) \\ &= -\sum_{t=1}^{n_y} \log p(y_t | y_1, \ldots, y_{t-1}) \end{aligned} \tag{2.1}$$

During inference, the next output word is chosen by a greedy left-to-right process, $y_{t+1} = \text{argmax}_y \, p(y|y_t, h_t)$, without considering the complete sequence. This approach, however, might not produce the most likely sequence according to the abovementioned objective, an outcome known as *search error*. One way to reduce the search error is beam search, tracking k word candidates. [8, 14]

In addition, this setup suffers from *exposure bias* because of the distribution mismatch of ground-truth data and the model's predictions. Bengio et al. (2015) [15] proposed data as demonstrator (DAD) (also "scheduled sampling") slowly and randomly replacing the ground-truth labels with sampled actions from the model at every decoding step. However, in this approach, the target labels are still chosen by ground-truth data, which may lead to misalignments. [8, 14]

2.3. Reinforcement Learning

2.3.1. Markov Decision Process

Reinforcement learning (RL) is the problem of learning from interactions. The learner, referred to as the agent, is placed in an environment in which it sequentially chooses actions. The environment responds to these actions, potentially leading to a reward. This kind of framework is often formulated as a *Markov decision process* (MDP) and is defined by a tuple $M = (\mathcal{S}, \mathcal{A}, T, r, s_0, \gamma)$. [16, pp. 1, 37 sq.][8]

$\mathcal{S}$ describes the state space (e.g., pixel data from video-game screens), while $\mathcal{A}$ describes the action space. For games, the actions might be up, down, left, and right, and in NLP, the action space might be the vocabulary space. A transition function $T(s, a, s') = P\left[S_{t+1} = s' \,|\, S_t = s, A_t = a\right]$ returns the probability for stochastic transitions from one state to another conditioned on a specific action. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns a real number to an action, given the state. s_0 describes the distribution of the initial state space, for example, where the agent starts. [16, pp. 37 sq.][8]

The policy function $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$ chooses the agent's next action with a probability distribution over the actions, while the goal is to maximize the reward with its actions. As the criterion, the expected cumulative reward $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t)\right]$ is used, that is, all expected future rewards resulting from an action. The discount factor γ, which puts more weight on rewards in the near future, is also noteworthy here. [16, pp. 37 sq.][8]

2.3.2. Value Functions

Most reinforcement learning algorithms utilize value functions to estimate the "quality" or "goodness" of a state or an action. The state's value is defined as the expected reward when starting in s and following a policy π afterwards. Here, v^π is called the state-value function: [16, pp. 45 sq.]

$$v^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \,\middle|\, S_t = s\right] \tag{2.2}$$

Analogously, the action-value function q^π is defined. Again, state s is the starting point. However, unlike before, a particular action a is chosen first, only after which the policy is followed. [16, pp. 45 sq.]

$$q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t) \,\middle|\, S_t = s, A_t = a\right] \tag{2.3}$$

Value functions can be described using the *Bellman equation*, exploiting their recursive property: [16, pp. 46 sq.]

$$v^\pi(s) = \mathbb{E}_\pi[r(S_t, A_t) + \gamma v^\pi(S_{t+1}) \,|\, S_t = s] \tag{2.4}$$

$$q^\pi(s,a) = \mathbb{E}_\pi[r(S_t, A_t) + \gamma q^\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a] \tag{2.5}$$

The difference between the two sides is called the *Bellman error*.

It is possible to define the optimal value function, that is, the value function of the optimal policy, in a similar fashion to the Bellman equation (which is called the *Bellman optimality equation*): [16, pp. 49 sq.]

$$\begin{aligned} v^*(s) &= \max_\pi v^\pi(s) = \max_a q^*(s,a) \\ &= \max_a \mathbb{E}\left[r_t(s_t,a) + \gamma v^*(s_{t+1}) \,|\, S_t = s, A_t = a\right] \end{aligned} \tag{2.6}$$

Similarly, the Bellman optimality equation for q^* is obtained as follows: [16, pp. 49 sq.]

$$q^*(s,a) = \mathbb{E}\left[r(S_t,A_t) + \gamma \max_{a'} q^*(S_{t+1},a') \,\middle|\, S_t = s, A_t = a \right] \tag{2.7}$$

For notational convenience and ease of analysis, it is helpful to define an operator B, which maps a value function to another. Specifically, in this context, often a *Bellman operator* $\mathcal{T}^\pi$ and a *Bellman optimality operator* $\mathcal{T}$ is employed. They are both contraction mappings which means their repeated application to some value function q_0 leads to q^π respectively q^*.

$$\mathcal{T}^\pi q(s,a) = \mathbb{E}_\pi\left[r(S_t,A_t) + \gamma q(S_{t+1},A_{t+1}) \,|\, S_t = s, A_t = a\right] \tag{2.8}$$

$$\mathcal{T} q(s,a) = \mathbb{E}\left[r(S_t,A_t) + \gamma \max_{a'} q(S_{t+1},a') \,\middle|\, S_t = s, A_t = a \right] \tag{2.9}$$

2.3.3. Monte Carlo Methods and Temporal-Difference Learning

The most straightforward approach to learning such value functions is from experience via Monte Carlo methods. The key idea is to collect numerous random samples and their actual rewards, followed by averaging them for every state (or state-action pair). This notion is often implemented as the following update rule: [16, pp. 73, 95 sq.]

$$v^\pi(S_t) \leftarrow v^\pi(S_t) + \alpha\left[G_t - v^\pi(S_t)\right] \tag{2.10}$$

where G_t is the sampled discounted reward of a whole episode and $\alpha \in (0, 1]$ is the learning rate or step-size parameter determining to what extent prior information is overridden by newly acquired information (thus resulting in an *exponential recency-weighted average*). Essentially, the update rule approximates the state-value function v^π using estimates of Equation 2.2, notwithstanding their very high variance. [16, pp. 25, 73, 95 sq.]

Waiting for the completion of each episode may slow the training process. To address this issue, temporal difference (TD) learning, *TD(0)*, forms a target after every transition: [16, pp. 73, 95 sq.]

$$v^\pi(s) \leftarrow v^\pi(s) + \alpha\,[R_{t+1} + \gamma v^\pi(s_{t+1}) - v^\pi(s)] \tag{2.11}$$

where the term $R_{t+1} + \gamma v^\pi(s_{t+1}) - v^\pi(s)$ is known as *TD error* and denoted as δ_t. This term is only slightly different from the Bellman error (see Section 2.3.2), but without the expectation, instead being evaluated after just a transition.

Updating by utilizing the existing estimation of the value function is called *bootstrapping*, which introduces a substantial bias whilst reducing variance. There are *n-step TD* methods called *TD(n)* that trade off bias and variance, as discussed in Section 3.4. [16, pp. 73, 95 sq.]

2.3.4. Q-Learning

Q-learning was first described by Watkins (1989). It aims to approximate the optimal Q-function q^* directly. Thus, Q-learning is based on the Bellman optimality equation (see Equation 2.7). It is a TD(0) algorithm, which is why the update rule is similar to Equation 2.11. [16, p. 105]

$$q(s, a) \leftarrow q(s, a) + \alpha\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(s, a)\right] \tag{2.12}$$

Since Q-learning formulates the target based on the greedy policy by taking the max action, it is independent of the policy that is currently followed, which makes it an *off-policy* algorithm. [16, p. 105]

2.3.5. Deep Q-Network

In large state spaces, tabular algorithms grow exponentially in time and space complexity; some state spaces are even infinite. In such cases, it is possible to implement function approximators being able to generalize and interpolate between states. Mnih et al. (2015) [17] applied Q-learning to deep neural networks. This architecture is known as *deep Q-network* (DQN). In this setup, the authors minimized the squared TD error via RMSprop, a variant of stochastic gradient descent (SGD): [16, pp. 359 sq.][18]

$$\delta_t^2 = \left(R_{t+1} + \gamma \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t)\right)^2 \tag{2.13}$$

Here, q_θ represents the *online network* selecting actions and also being used to backpropagate the gradients. While $q_{\bar{\theta}}$ is the *target network*, a periodic copy of q_θ, which combats the issue of moving targets and stabilizes the training routine. [18]

To leverage the off-policy nature of DQN, an *experience replay buffer* is utilized. For this, at each time step, the agent's experience $(S_t, A_t, S_{t+1}, R_{t+1})$ is stored in the replay buffer. The collection process and the training process are performed repeatedly and in an alternating fashion. For training, mini-batches are uniformly sampled from the replay memory, and their gradient information is accumulated. As a result, consecutive samples are decorrelated. The ϵ*-greedy* strategy is employed for exploration, that is, choosing a random action with probability ϵ and otherwise picking actions greedily according to the q function q. [16, p. 362]

2.3.6. Policy Gradient Methods

Policy gradient (PG) methods aim to learn and parametrize the policy directly. This *parameterized policy* $\pi(a|s, \theta)$, with θ being the parameters, can select actions without consulting a value function. It employs *gradient ascent* to maximize the performance measure $J(\theta)$: [16, p. 321]

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{2.14}$$

In general, this performance measure is the expected reward $J(\theta) = \mathbb{E}_\pi[r(\tau)]$. However, the *policy gradient theorem* provides a useful reformulation of the derivative of the expected reward: [16, p. 326]

$$\nabla J(\theta) = \mathbb{E}_\pi[r(\tau)\nabla \log \pi(\tau|\theta)] \tag{2.15}$$

Intuitively expressed, the probability of a trajectory τ (likelihood of the observed data) is weighted by the given rewards, eventually leading to an increase of trajectories that issue higher rewards. Section 2.3.7 provides a practical approach of utilizing this theorem.

2.3.7. REINFORCE

The remaining expectation of the policy gradient theorem (see Equation 2.14) can be unbiasedly approximated by Monte Carlo methods, that is, by sampling complete trajectories. This notion is the premise of the REINFORCE [19] algorithm: [16, pp. 326 sqq.]

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_\pi[G_t \nabla \log \pi(A_t|S_t,\theta)] \\ &\approx \sum_{t=1}^{T} \gamma^{t-1} r(S_t,A_t) \sum_{t=1}^{T} \nabla \log \pi(A_t|S_t,\theta) \end{aligned} \tag{2.16}$$

This approach has a substantial variance, which can be reduced using a baseline $b(s)$ for the reward as comparison:

$$\nabla J(\theta) \approx \left(\left(\sum_{t=1}^{T} \gamma^{t-1} r(S_t,A_t)\right) - b(S_t)\right) \sum_{t=1}^{T} \nabla \log \pi(A_t|S_t,\theta) \tag{2.17}$$

The simplest baseline is the average sampled reward of a batch of N trajectories:

$$\nabla J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\left(G_{i,t} - b(S_{i,t}) \right) \sum_{t=1}^{T} \nabla \log \pi(A_{i,t} | S_{i,t}, \theta) \right) \tag{2.18}$$

with $b(s) = \frac{1}{N} \sum_{i=1}^{N} G_t$.

However, more sophisticated approaches for the baseline include self-critic (SC) [20] and learned value functions as in *actor-critic* methods like advantage actor-critic (A2C) [21].

2.3.8. Comparison between Policy Gradient and Value-Based Approaches

Compared to value-based approaches policy gradient methods (including actor-critics) have both advantages and disadvantages.

- Value-based approaches are (typically) only able to learn deterministic policies, which makes them unsuitable for certain problems such as poker or rock-paper-scissors. For instance, for rock-paper-scissors a uniform random policy is optimal (i.e., Nash equilibrium). PG methods are capable of learning such stochastic policies. Deterministic policies can be easily exploited.
- Actions in PG methods can be continuous. Value-based approaches are bound to a discrete action space.
- PG methods do entirely without bootstrapping and instead rely on sampling alone, which is why they have a very high variance. This caveat still applies to actor-critic methods, though to a lesser extent. In contrast, value-based approaches have a high bias, even though there are ways to trade off bias and variance via n-step returns.
- In many cases, policy gradients converge to a local rather than the global maximum. This outcome contrasts sharply to value-based approaches, which always try to reach the optimum.
- PG methods are sensitive to the absolute value of the rewards. Therefore, constant additions can substantially change their convergence properties.

- Being on-policy makes the algorithm quite sample inefficient as it needs to generate new samples at every iteration. Conversely, off-policy algorithms are able to store and replay samples. However, this does not necessarily result in reduced wall-clock time.

It should be noted that there is a theoretical link between both approaches. Schulman et al. [22] presume that "Q-learning methods are secretly implementing policy gradient updates" and prove the precise equivalence between entropy-regularized Q-learning and policy gradient. A similar connection between actor-critic and Q-learning algorithms is proposed by Nachum et al. [23].

3

DQN Improvements

The classic DQN approach outlined in Section 2.3.5 suffers from many limitations that have been addressed in recent years. *Rainbow* by Hessel et al. [18] seeks to combine six different extensions to improve overall performance. These improvements are presented in this chapter.

3.1. Double Q-Learning

DQN suffers from *maximization bias*. In general, the model's estimated expectations of the action values are noisy and uncertain. Consequently, some action values are distributed above, whereas others are distributed below their true values. That said, such maximization in the action selection engenders a constant overestimation. [16, pp. 108 sq.] This issue is solved by the decorrelation of noise from action selection and noise in values by having two different Q-networks. In practice, however, the two existing networks, the online and target networks, are utilized. [18]

$$\delta_t^2 = \left(R_{t+1} + \gamma q_{\bar{\theta}} \left(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, q_\theta(S_{t+1}, a') \right) - q_\theta(S_t, A_t) \right)^2 \tag{3.1}$$

3.2. Prioritized Experience Replay

Classic experience replay samples transitions from the buffer according to a uniform distribution. In contrast, *prioritized experience replay* (PER) aims to sample transitions more effectively and from which is much to learn [24]. One means to this goal is to prioritize transitions whose last encountered TD error is high. Because greedy prioritization is prone to over-fitting, a stochastic prioritization method defines the probability to sample transition i:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{3.2}$$

where p_i is the priority value, and α determines the weighting between pure greedy prioritization and uniform random sampling. For proportional prioritization, p_i equals $|\delta_i| + \epsilon$.

In this way, PER over-samples high-priority transitions. Naturally, this approach introduces a bias, which can be corrected by employing importance sampling (IS) to downweight these transitions.

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{3.3}$$

The bias is fully corrected if $\beta = 1$. These unbiased updates are most important near convergence, which is why β is usually annealed from β_0 to 1 over the time of training.

3.3. Dueling Networks

When considering large action spaces in which only a small number of actions are suitable, it seems unnecessary to estimate the values for every action. Therefore Wang et al. [25] proposed decomposing the action value into the state value and the advantage:

$$q_\theta(S_t, A_t) = v_\eta(S_t) + a_\psi(S_t, A_t) \tag{3.4}$$

Calculating q in this way makes the equation unidentifiable. Thus, $\max_a a_\psi(S_t, a)$ is usually subtracted from the advantage estimation to have a zero at the selected action. For stabilization, the max operator is often replaced with the average, resulting in the following equation:

$$q_\theta(S_t, A_t) = v_\eta(S_t) + \left(a_\psi(S_t, A_t) - \frac{1}{|\mathcal{A}|} \sum_a a_\psi(S_t, a) \right) \tag{3.5}$$

Dueling nets are implemented as two-stream networks with a shared encoder for the state.

3.4. Multi-Step Learning

As hinted in Section 2.3.3, there are TD(n) approaches that trade off between Monte Carlo methods (with high variance) and TD(0) methods (with high bias) by using n-step returns. For DQN, a multi-step variant can be implemented with the following loss function:

$$\delta_t^2 = \left(R_t^{(n)} + \gamma^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t) \right)^2 \tag{3.6}$$

where $R_t^{(n)}$ defines the summed and discounted reward for n steps. Sutton et al. states that "an intermediate amount of bootstrapping" typically performs better than either extreme. [16, pp. 141 sqq.]

3.5. Distributional Reinforcement Learning

Thus far, a scalar value, the expectation value of rewards, is considered, that is, $q^\pi = \mathbb{E}[z^\pi]$ where z^π is a random variable of discounted future rewards: $z^\pi(S_t, A_t) = \sum_{t=0}^{\infty} \gamma^t r(S_t, A_t)$. However, it is also possible to approximate the distributions over returns z directly. Analogous to the Bellman equation and the Bellman operator (see Equation 2.4 and 2.8), there are the *distributional Bellman equation* and the *distributional Bellman operator*:

$$z^{\pi}(S_t, A_t) \overset{\mathrm{D}}{=} r(S_t, A_t) + \gamma z^{\pi}(S_{t+1}, A_{t+1}) \tag{3.7}$$

$$\mathcal{T}^{\pi} z(S_t, A_t) \overset{\mathrm{D}}{=} r(S_t, A_t) + \gamma z(S_{t+1}, A_{t+1}) \tag{3.8}$$

Categorical DQNs. Bellemare et al. (2017) [26] have demonstrated that the return distribution $z(s, a)$ can be modelled as a categorical distribution over a fixed set of equidistant points $z_1 \dots z_N$. In this setup, the probabilities p_i associated with the values z_i are learned. The authors introduced a projection step ϕ to map the target $\mathcal{T}^{\pi} z$ onto these atomic supports via linear interpolation, followed by minimization of the KL divergence using the cross-entropy loss. To approximate the action-value distributions iteratively, they introduced the novel *C51 algorithm*. However, their approach has a sharp, theoretical disconnect as it relies on a projection step to fix disjoint supports and does not consider a true probability metric.

Quantile Regression DQNs. While C51 is applied in Rainbow DQN, Dabney et al. (2018) [27] suggest modelling the return distribution by a discrete set of quantiles $q_1 \dots q_N$ whose locations are adjusted using quantile regression. Their QR-DQN algorithm has shown to minimize the Wasserstein distance. In contrast to categorical DQNs, the probabilities are fixed while the supports (i.e., the values or the centres of mass in respective quantiles) are learned. This approach does not require domain knowledge to define the range of values. Moreover, no projection step is necessary. Aside from the different sized output layer, only minimal changes are needed to apply this approach to the classic DQN. Most importantly, the loss, commonly mean squared error (MSE) or the Huber loss, is replaced with the *quantile Huber loss* which penalizes overestimations in lower quantiles (i.e., with weight τ), and underestimations in upper quantiles (i.e., with weight $1-\tau$). The quantile Huber loss is given by:

$$\rho_{\tau}^{\kappa}(\delta_{ij}) = \left|\tau - \mathbb{I}(\delta_{ij} < 0)\right| \mathcal{L}_{\kappa}(\delta_{ij}) = \begin{cases} \tau \mathcal{L}_{\kappa}(\delta_{ij}), & \text{if } \delta_{ij} \geq 0 \\ (1-\tau)\mathcal{L}_{\kappa}(\delta_{ij}), & \text{if } \delta_{ij} < 0 \end{cases} \tag{3.9}$$

where quantile $\tau_i = i/N$ is the cumulative probability and $\mathcal{L}_\kappa$ is the Huber loss. Moreover, $\delta_{ij} = R(S_t, A_t) + \gamma\theta_j(S_{t+1}, A_{t+1}) - \theta_i(S_t, A_t)$ is the TD error with θ_i being the predicted support at a fixed quantile target $\hat{\tau}_i = \frac{\tau_{i-1}+\tau_i}{2}$. This loss is calculated for all pairs of $(\theta_i(S_t), \theta_j(S_{t+1}))$.

It should be noted that both algorithms introduce a hyperparameter N, which is the number of equidistant points or the number of quantiles, respectively.

In both cases, the policy is still determined by the mean of the distribution, just as in classic Q-learning. However, the empirical results indicate that "learning distributions matters in the presence of approximation" [26]. Bellemare et al. suggest that distributions are a more stable target and that the agent benefits from auxiliary predictions. Their view is based on a similar principle to auxiliary tasks in RL or multitask learning in general. Most recent approaches, such as implicit quantile networks (IQNs) [28] or fully quantile functions (FQFs) [29], go even further by considering the whole distribution in the action selection process, which leads to increased risk-awareness and risk-aversion.

3.6. Noisy Nets

In order to support exploration, ϵ-greedy policies (random steps with some probability ϵ) or entropy regularization (penalizing highly confident predictions) are ordinarily applied. In [30], the authors address the issue by adding Gaussian noise to the last fully connected layers. The parameters of the noise can be adjusted by the model itself; that is, the noisy layer is self-annealing. Essentially, the agent decides itself what amount of exploration it applies to which part of the state space. The noisy layer with factorized Gaussian noise is defined by

$$y = (b + Wx) + \left(b_{\text{noisy}} \odot \varepsilon^b + \left(W_{\text{noisy}} \odot \varepsilon^w\right) x\right) \tag{3.10}$$

where ε^b and ε^w are random variables, and $\odot$ is the Hadamard product.

4

Related Work

Encoder-decoder models suffer from problems that remain only partially resolved, such as the exposure bias, which refers to the fact that the model has not been exposed to its own predictions [14, 15]. Moreover, it is questionable whether the (word-level) training objective, the maximum likelihood objective, is capable of approximating the real-world objective and the evaluation metrics. When this training objective is applied to dialogue systems, it has been observed that the system tends to respond with dull and short-sighted answers [31].

Sequence generation using RL. Therefore, and as one of the first, Ranzato et al. (2015) [14] applied reinforcement learning on different sequence generation problems to align training and test measurements. The researchers utilized supervised training to initialize the policy, and they then introduced the mixed incremental cross-entropy REINFORCE (MIXER) algorithm which provides an annealing schedule between supervised training (i.e., CE loss and teacher forcing) and reinforcement learning, using REINFORCE [19] and the model's own predictions.

Due to the disadvantages of REINFORCE, which include the time-consuming training and its high variance, Bahdanau et al. (2017) [32] took this idea one step further by employing an actor-critic setup to train encoder-decoder models. To speed up the training process and deal with the ample action space, the authors used several techniques such as penalizing the variance of the value predictions and reward shaping to provide rewards for the whole sequence.

Rennie et al. [20] (2017) introduced another variant, a self-critic baseline for PG methods employed in sequence generation. The algorithm avoids estimating a normalization (cf. REINFORCE) or training a value function (cf. actor-critics) for the baseline by utilizing the inference outputs.

However, all above mentioned approaches rely on well-trained supervised models, and the application of RL is limited to fine-tune such models. Consequently, this line of research focuses on policy-gradient methods as the provided outputs are alike (i.e., token probabilities based on softmax activations).

Advanced applications of RL in NLP, and reward engineering. Based on the results of Ranzato et al. [14], the authors of [31] have proposed applying RL to chit-chat dialogue systems in a three-stage training process. First, the system is pretrained using the maximum likelihood objective. Secondly, it is refined applying MIXER with mutual information between source and target as the reward function, that is $P(x, y)$. [33] has demonstrated that mutual information can significantly reduce the probability of dull responses. In the final stage, self-play learning is deployed, simulating dialogues between two agents, similar to the strategies applied for TD-Gammon [34] or AlphaGo [35]. The plain REINFORCE algorithm is utilized, and the reward is defined as a weighted sum of three factors, namely ease of answering, information flow, and semantic coherence.

Similarly, Kreutzer et al. [36] (2018) defined a three-staged training process in the field of machine translation, initializing the model with parameters based on a fully supervised training on a large out-of-domain corpus, then refining such model with a supervised training on in-domain data. Eventually, they applied RL using REINFORCE, while also exploring an off-policy version of the algorithm and different rewards: "simulated rewards" (e.g., BLEU), direct user rewards, and estimated rewards (based on direct user rewards).

Paulus et al. (2017) [37] adopted the SC approach to learn the task of generating abstractive summaries. The authors utilized a mixed training objective function weighting the MLE and RL loss components with a scaling factor γ. They compared a supervised trained model, a fine-tuned RL model and a model with such a mixed training objective. The researchers found the fine-tuned RL model to score the highest ROUGE value while the mixed model was able to improve on relevance and readability over both compared models.

[38] seeks to employ adversarial learning to approximate a reasonable reward function, thus avoiding an exhaustive reward engineering (cf. inverse RL). In this setup, a generative model generates dialogues, while a discriminative model needs to distinguish between machine-generated dialogues and actual human dialogues. The output of the discriminator can then serve as a reward.

Dialogue systems with restricted domains. A different approach for dialogue systems is defining verbal actions instead of equating the vocabulary space with the action space, thus avoiding high-dimensionality. This redefinition of the action space can be applied to restricted domains, for instance, movie booking (cf. [39]), restaurant search (cf. DTSC2 [40]), or some question-answering problems (cf. bAbi dataset [41]). Such systems are typically modular, separating natural language understanding, dialogue state tracking and natural language generation. The verbal actions are templates or dialogue acts. Examples of such approaches include [42], [43], and [44]. Classic PG methods, as well as value-based RL methods such as DQN or DRQN, has been utilized for this kind of problems.

Recurrence and memories in RL. Another related research direction seeks to equip DQN agents with a memory. The first notable work in this area was conducted by Hausknecht et al. in 2017, with their proposed DRQN [45]. The authors argue that most real-world applications fail to meet the Markov property; that is, their true states are only partially observable. Consequently, they propose replacing the first fully-connected layer with an LSTM layer. With this goal in mind, Hausknecht et al. consider two possibilities: sequential updates (replaying whole episodes while violating DQN's random sampling policy) and random updates (with zeroing out hidden states). The paper concludes that both updates work similarly well.

A more recent approach is recurrent replay distributed DQN (R2D2), developed by Kapturowski et al. (2019) [46]. They authors store and replay fixed-length sequences ($m = 80$) using a mix of mean and max for prioritization of the samples: $p = \eta \max_i \delta_i + (1 - \eta)\overline{\delta}$ with $\eta = 0.9$. They hypothesize two strategies for the hidden state: storing and replaying such states or applying a burn-in period, which involves using a part of the replay sequence to produce a start state.

5

Datasets

5.1. OpenSubtitles

The OpenSubtitles dataset [47] is a large-scale parallel corpus of movie and TV subtitles for more than 60 languages. Regarding the English language, it contains more than 337 million utterances for over 106,000 distinct movies and TV episodes. No information about dialogue structure and no author annotations are available, however, which significantly reduces the data quality.[1] To address this shortcoming, additional corpus building was undertaken by Jean Senellart. He has extracted pairs of utterances (single-turn dialogues) where the first sentence ends with a question mark, and the consecutive utterance does not, leaving around 14 million sentences overall.[2]

[1] `http://opus.nlpl.eu/OpenSubtitles2016.php`
[2] `http://forum.opennmt.net/t/english-chatbot-model-with-opennmt/`

5.2. Cornell Movie Dialogues Corpus

The Cornell Movie Dialogues Corpus [48] contains over 220,000 fictional conversations from 617 unique movies involving nearly 10,000 characters, which amounts to around 304,000 utterances in total. In contrast to most corpora, including the OpenSubtitles dataset (Section 5.1), this dataset is a metadata-rich collection. It not only contains the utterances but also specifies the movie character who uttered any given line. Moreover, this corpus is enriched with information about each movie title (e.g., year and rating) and character (e.g., gender and name).[3]

[3] https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

6

Methodology

As explained in Section 1.2, this research aims to apply the Rainbow DQN setup to the sequence-to-sequence architecture. Therefore, this chapter addresses the question of how the classic Q-learning approach can be transferred to this architecture. With the fulfilment of this requirement, most of the DQN extensions that have been used in Rainbow are also straightforwardly transferable. Specifically, the following extensions are included:

- Double Q-learning (see Section 3.1).
- Prioritized experience replay (see Section 3.2), which is the only extension that is not transferable without significant methodological changes (see Section 6.3).
- Dueling networks (see Section 3.3).
- Multi-step learning (see Section 3.4).
- Distributional RL; for this thesis, the more recent QR-DQN is chosen over categorical DQNs because it is easier to implement and has also been shown to yield better results, though categorical DQNs have been used for Rainbow (see Section 3.5).
- Noisy nets (see Section 3.6).

6.1. Reinforcement Learning Setting

In this section, the reinforcement learning environment is described as it varies from one task to another. In general, the setting is similar to those in other sequence prediction tasks such as [14], [32], and [31], although the perspective on the state differs substantially as these works use policy gradient approaches.

Action space. In the context of this work, the action space $\mathcal{A}$ is the vocabulary space. At each time step t, the decoder of the seq2seq model chooses the next action A_t, which is a token in a sequence.

State space. The state S_t at a specific time step t includes all the input data that is required to produce the next action A_t. Since the decoder generates its output depending on the previous hidden state h_{t-1} and the previously chosen action y_{t-1}, $[h_{t-1}, y_{t-1}]$ may be used as the state. Alternatively, the input sentence and all previous actions $[x, y_{1:t-1}]$ can be viewed as the state, as it is possible to reproduce the hidden states with this information. This perspective is relevant for Section 6.4.

Reward. The reward function r can be any user-defined function. Advanced dialogue generation models, [31] for example, utilize rather complex reward functions such as combinations of information flow, semantic coherence, and ease of answering. A unique characteristic in NLP settings compared to other RL tasks is that the reward is always and only collected at the end of the sequence.

6.2. Rewards

To keep the thesis' work as simple, comparable and interpretable as possible, and to focus on the feasibility of transferring Q-learning to sequence-to-sequence models, BLEU and ROUGE are selected as exemplary reward functions. In addition, with such rewards, it will be possible to provide a

strong baseline for the model. The CE objective is known to approximate these metrics quite well. Nevertheless, more sophisticated and suitable reward functions, in the context of dialogue generation, are also conceivable. Both metrics evaluate a generated sentence against a reference sentence. Thus, a dataset with sources and targets is required. However, reward functions like those in [31] have no dependence on references, which enables self-playing training techniques.

6.2.1. BLEU

Bilingual evaluation understudy (BLEU) is a metric first presented by Papineni et al. in [49]. It is primarily used in machine translation and other language generation problems. The values range from 0 to 1. BLEU calculates a modified n-gram precision p_n, for which it counts the number of matches between the n-grams of the candidate and the n-grams of the reference divided by the total number of n-grams in the candidate. In order to prevent abundances of high-frequency words, the number of matches for a word is clipped after its maximum reference count. However, because this measure still enables very short candidates to achieve high-scoring results, a brevity penalty as a multiplicative factor is introduced to mimic some kind of "recall".

$$\mathrm{BP} = \begin{cases} 1, & \text{if } c > r \\ e^{(1-\frac{r}{c})}, & \text{if } c \leq r \end{cases} \tag{6.1}$$

It is possible to combine the scores of different n-gram sizes by calculating the geometric mean. The final equation is given by:

$$\mathrm{BLEU} = \mathrm{BP} \cdot \left(\sum_{n=1}^{N} w_n \log(p_n) \right) \tag{6.2}$$

where the weight w_n is usually the uniform distribution $1/N$.

The original definition of the brevity penalty, as indicated in Equation 6.1, has no solution at $c = 0$. This is particularly problematic in this RL setting with Q-learning, as the model tends to generate empty candidates in the early stages of training. Nevertheless, the generated candidates require evaluation in order to add them to the experience replay buffer. Hence, as part of this work, the brevity penalty is defined as zero if $c = 0$, which is tantamount to BLEU $= 0$.

$$\mathrm{BP} = \begin{cases} 0, & \text{if } c = 0 \\ 1, & \text{if } c > r \\ e^{(1-\frac{r}{c})}, & \text{if } 0 < c \leq r \end{cases} \tag{6.3}$$

6.2.2. ROUGE

Another metric to be evaluated with the model is recall-oriented understudy for gisting evaluation (ROUGE), which was presented in [50]. While ROUGE has been developed especially for text summarization tasks, it can be applied to all kinds of language generation problems.

Here, more precisely, ROUGE-W functions as the reward and as an evaluation metric. In contrast to BLEU (Section 6.2.1) and other versions of ROUGE, the longest common subsequence (LCS) is determined instead of n-gram overlaps. This approach means consecutive matches are not required, as it allows in-sequence matches on sentence-level order. Moreover, no predefined n-gram length needs to be specified, and it works for any sequence length. One drawback, however, is that consecutive matches are assigned the same score as non-consecutive matches. To address this issue, with ROUGE-W weights are introduced. The F1 metric is applied to take recall and precision equally into account.

6.3. Experience Replay for Sequence-to-Sequence Models

This section describes the implications of using (prioritized) experience replay in a sequential setup. Figure 6.1 presents a visualization of both classic approaches and the thesis' modified versions.

6.3.1. Experience Replay

Typically, Q-learning approaches with deep neural nets (i.e., DQNs as described in Section 2.3.5) store transitions experienced by the agent in a buffer called experience replay. These transitions are reiterated during the training process. A transition is defined by its state S_t and action A_t at time step t, the next state S_{t+1}, and the reward R_{t+1} received by the agent: $(S_t, A_t, S_{t+1}, R_{t+1})$. For seq2seq models, however, this approach has to be adjusted. As explained in Section 6.1, the state is defined by the previous hidden state of the decoder and the previous action $[h_{t-1}, y_{t-1}]$. However, given that the hidden state representation is not static, but learned during the training process, it is not suitable to be replayed in later phases of training. Here, the alternative state representation $[x, y_{1:t-1}]$ can provide a solution. Rather than storing

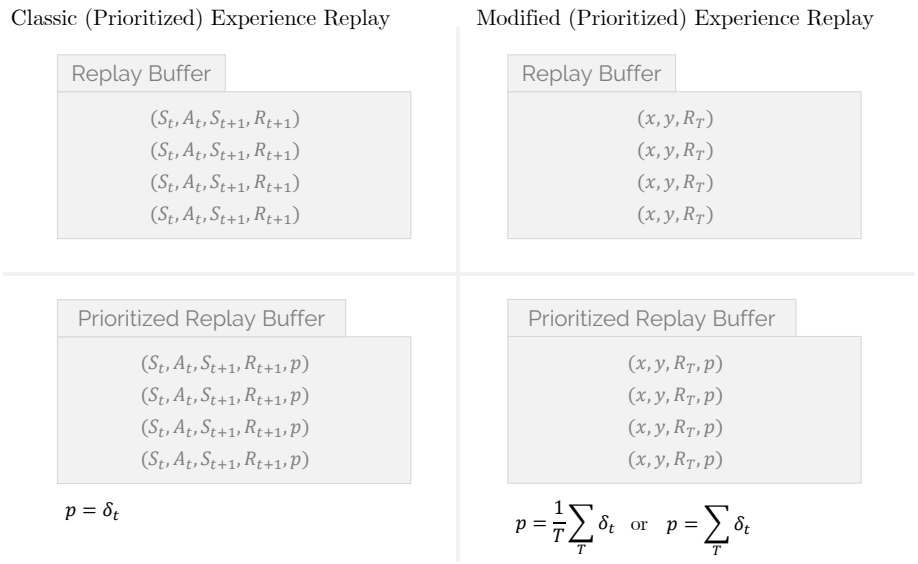


Figure 6.1: Classic and modified versions of (prioritized) experience replay

single transitions, it allows for storing the entire input and output sequence (x, y, R_T) to represent the states and actions of the whole episode. The entry is completed by a scalar reward R_T, as only the final transition issues a reward for the full sequence.

6.3.2. Prioritized Experience Replay

As noted in Section 6.3.1, experience replay is modified to store episodes instead of transitions. This decision has some implications, especially for prioritized experience replay, one of the extensions, which has been combined with others in [18]. In the original paper [24] by Schaul et al., the absolute TD error δ (see Section 2.3.5) is utilized as the criterion of importance for the transitions in the buffer. However, here it is necessary to deal with whole episodes, which consist of many transitions. Consequently, there is a need to aggregate the TD errors of the steps in episode e. For this approach, there are at least two options:

- Summing the errors: $p_e = \sum_i^T \delta_i$
- Averaging the errors: $p_e = \frac{1}{T} \sum_i^T \delta_i$

It may be expected that the summation of errors leads to an advantage of longer sequences at the expense of shorter ones; this outcome could be disadvantageous for the overall success. In fact, early experiments have indicated that averaging is superior.

6.4. Teacher Forcing

As discussed in Section 6.3.1, entire episodes must be stored in the experience replay buffer to combine it with sequence-to-sequence models. However, as the name suggests, it is necessary to replay the episodes. This is where a technique that is widely used in supervised learning for seq2seq models comes into play: teacher forcing (see Section 2.2.2). Instead of feeding the decoder's output to the input of the next sequential unit (as in the inference stage), the ground-truth sequence is fed into the network. The same idea can be applied to replay

episodes, but in place of the ground-truth sequence, the stored output sequence is inputted into the recurrent units of the decoder. The essential difference is that the output sequence does not necessarily have to be one of the "good examples". The examples in the experience replay buffer are usually collected by the model itself, substantially reducing the exposure bias caused by the distributional mismatch of decoder inputs in the training and inference stages. Here, the model's own predictions are replayed in the training stage, syncing the input distributions. Consequently, while the algorithm applied is the same for supervised learning and the RL approach taken in this study, its aim and motivation is entirely different. Moreover, teacher forcing has computational benefits since it allows parallelization.

6.5. Utilization of Demonstration Data

Although it is highly flexible in defining its goals and rewards, reinforcement learning also has some downsides: it is usually exceedingly sample inefficient and converges much slower than supervised learning. Furthermore, data collection is time-consuming. This is why, as part of this work, methods are explored with which available information can be utilized to accelerate convergence.

6.5.1. Preloading Replay Buffer

DQNs learn from transitions being collected by the agent and stored in the experience replay buffer. In the case of this work, however, human demonstration data is already at hand, as there are some corpora available to use (see Chapter 5). The simplest way to leverage such data is to preload it into the replay buffer instead of filling the buffer with random experiences in the beginning. For practical reasons, the experience replay buffer typically has a size limit. This is why older transitions get replaced by more recent experiences. However, to prevent the displacement of exemplary data, such data is excluded from the "first in, first out" replacement policy and instead is permanently stored in the buffer.

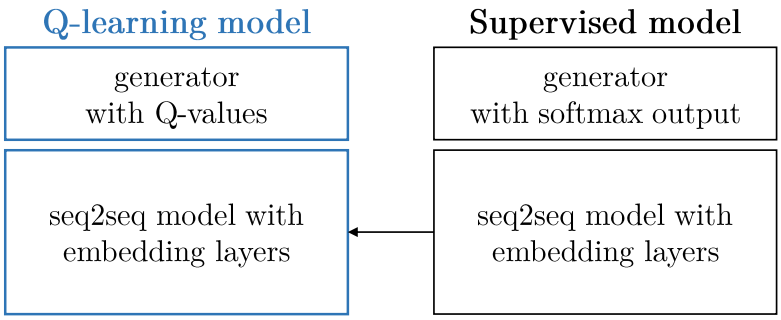


Figure 6.2: Transfer learning: initialization with the supervised model

6.5.2. Transfer Learning

This approach is inspired by classic transfer learning. As Yosinski et al. [51] have shown, layers in a deep neural network for image classification, that were trained on a specific task can be transferred to others to varying degrees. The new model is then able to train faster. In particular, early layers in the network are rather general, agnostic regarding the specifics of the input, and therefore easily transferable. In the context of this work, this method entails pretraining a typical seq2seq network using supervised learning (as described in Section 2.2). The parameters of this model, or specifically, the parameters of the encoder, decoder, and the embedding layers while discarding those of the generator, are utilized to initialize the Q-learning model, which has its own randomly initialized generator (cf. Figure 6.2). Thus, the recurrence and embeddings may not have to be learned from scratch. This approach is similar in conception to those in [14], [32], and [31]. However, for PG methods, it is not necessary to replace the generator because both generators produce probabilities for the defined set of tokens. On the contrary, in DQNs, the output layer utilizes a linear activation function. Additionally, the number of neurons in the output layer differ when employing distributional RL (cf. Section 3.5).

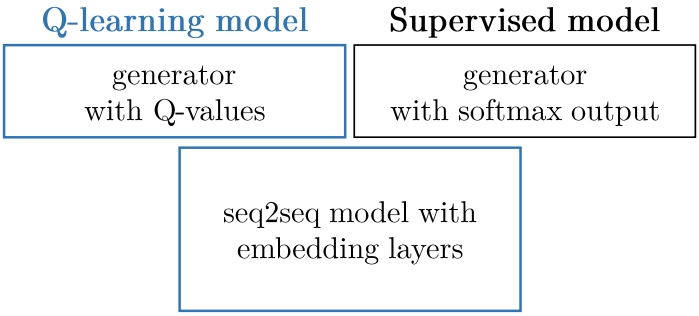


Figure 6.3: Multitask learning: sharing a common feature extractor

6.5.3. Multitask Learning

Transfer learning works optimally when the training data and training objective of both tasks are similar. In this study, however, the objectives differ substantially, as [31] suggests. On the one hand, there is the MLE criterion; on the other hand, Q-values, the estimated future rewards, are to be predicted. Thus, instead of using transfer learning, it would be possible to treat these objectives as two different tasks, but to employ a shared "feature extractor", which, in this case, is the encoder RNN, the decoder RNN and the embedding layers (cf. Figure 6.3). The general idea is known as multitask learning, and it has been successfully applied to a broad range of applications, including NLP [52] and computer vision [53]. Originally, multitask learning was described by [54]: it is usually implemented by sharing hidden layers between several tasks while having task-specific output layers. These tasks are learned jointly by alternating the optimization steps for each. The different tasks benefit from each other as they introduce regularization and reduce the hypothesis space.

7

Implementation

This chapter focuses on the implementation. It presents an overview of the development environment, the libraries, and the reference implementations (see Section 7.1) while also highlighting some of the non-trivial implementation issues (see Section 7.2, 7.3, and 7.4). The implementation of the work conducted is available online.[1]

7.1. Environment and Tools

The implementation was realized with the Python framework PyTorch and is generally based on OpenNMT (Open Neural Machine Translation) [55], as the library provides efficient implementations of sequence-to-sequence architectures. On top of that, there are some reference implementations for the Rainbow approach[2,3]. The modified prioritized experience replay buffer is based on the high-quality baseline implementation by OpenAI[4]. BLEU is borrowed from TensorFlow[5], while for ROUGE, there is a Python package available[6].

[1] https://github.com/ScientiaEtVeritas/rainbow-dialogues
[2] https://github.com/qfettes/DeepRL-Tutorials
[3] https://github.com/higgsfield/RL-Adventure
[4] https://github.com/openai/baselines
[5] https://github.com/tensorflow/nmt/blob/master/nmt/scripts/bleu.py
[6] https://pypi.org/project/rouge/

7.2. Episodes

Most of the implementation effort is required because the model presented in Chapter 6 does not work with batches of single transitions as in Rainbow reference implementations, but rather with batches of whole episodes (i.e., sequences of transitions). This difference adds another dimension to the tensors and calculations.

The additional complexity can be observed, for instance, when calculating the target $\hat{q}_{\bar{\theta}}$ for multi-step learning (cf. Equation 3.6):

$$\hat{q}_{\bar{\theta}} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a') \tag{7.1}$$

This procedure requires the combination of the reward tensor R and the tensor for estimates of optimal future values $\max_{a'} q_{\bar{\theta}}(S_{t+n}, a')$, as visualized in Figure 7.1. Due to taking multi-steps, a window of n rewards must be considered at a specific time step t while there is a $n-1$ shift for the value estimate tensor. To obtain the target via simple addition of the tensors, the estimates tensor can be transformed to be of the same shape, discarding the first $n-1$ steps while zero-padding n final states. The reward tensor, conversely, can be "dewindowed" utilizing convolutions (see Section 7.4).

Furthermore, the differences in length of the episodes necessitate the careful application of sequence padding and masking as one proceeds.

7.3. Normalization

In many implementations of sequence-to-sequence models, bucketing and padding of sequences is applied. However, bucketing cannot be used in conjunction with an experience replay buffer which samples whole episodes (i.e., sequences instead of single transitions; see Section 6.3 for details). The lengths

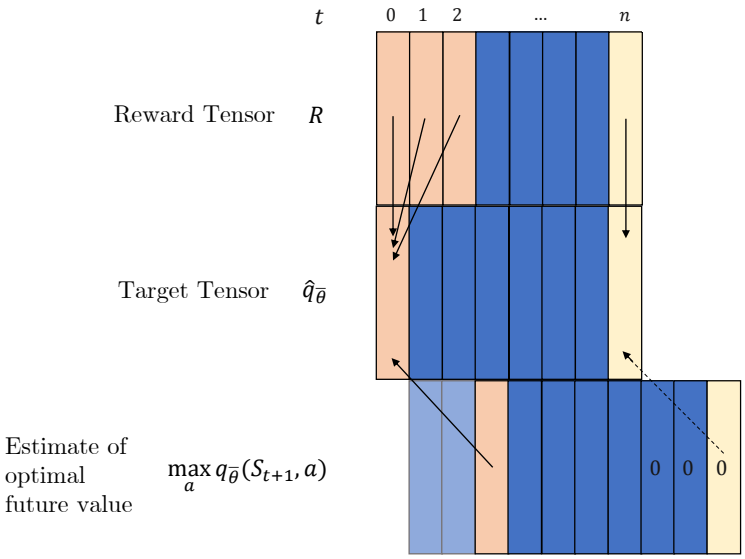


Figure 7.1: Calculating targets for multi-step Q-learning in a seq2seq setup

of sequences in a batch are completely randomized, which may lead to a considerable gap between minimum and maximum sequence length in a batch. In early experiments, this led to the effect that normalization of the loss by the number of tokens is superior to normalization by the batch size or no normalization at all.

7.4. Multi-Step Learning as Convolution

For the multi-step learning case (see Section 3.4), the target is obtained by $R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{(n)} \max_{a'} q_{\bar{\theta}}(S_{t+n}, a')$. Thus, instead of utilizing a single reward, the next n steps are summed while being exponentially decayed using the discount factor γ. However, in sharp contrast to most applications of multi-step learning, this thesis works with batches of sequences, which allows viewing the term as a convolution. Specifically, it is a one-dimensional, axis-aligned convolution whose input array's values beyond the edge are filled with a constant value of 0. The kernel can be calculated in advance with $[\gamma^{n-1}, \gamma^{n-2}, \ldots, \gamma, 1]$.

8

Results

8.1. Experimental Setup

8.1.1. Models

In the following experiments, four models are investigated:

- As comparative model, a supervised trained seq2seq model conditioned on the maximum likelihood objective is employed (cf. Section 2.2.2). Such a model can be considered a strong baseline because the CE loss is known to approximate BLEU and ROUGE quite well.
- A seq2seq network trained purely with reinforcement learning, based on the Rainbow method introduced in Chapters 2 and 3, including the methodological modifications needed presented in Chapter 6. In this setup, the replay buffer is preloaded with demonstration data, as described in Section 6.5.1.
- A transfer learning model, as presented in Section 6.5.2, is not fully evaluated because early experiments have shown it to usually converge to suboptimal solutions (for further analysis, see Section 8.2.4).

Vocabulary Size	Dataset Size	Word Minimum Frequency
111	1,311	900
201	3,484	680
401	9,230	130
806	18,523	50

Table 8.1: Dataset vocabulary sizes

- A multitask network (see Section 6.5.3), which jointly trains the supervised learning and reinforcement learning models described above.

8.1.2. Dataset

For subsequent experiments, the Cornell Movie Dialogue dataset is utilized (see Section 5.2). As part of this work, only smaller vocabulary sizes are evaluated and a clean comparison between the models is striven for. Accordingly, the chosen dataset's high quality outweighs the relatively small quantity of data, in contrast to the OpenSubtitles dataset which has a higher quantity but lower quality.

8.1.3. Hyperparameters

All four models (see Section 8.1.1) share some architectural hyperparameters, which are presented in Table 8.2. Moreover, the SL model and the RL model have additional hyperparameters, shown in Table 8.3 and Table 8.4, respectively. Since both models are part of the multitask learning and transfer learning model, all hyperparameters are also applied to it. Furthermore, a mixing ratio between those two models is introduced with multitask learning, along with a number of supervised lead iterations for transfer learning settings (see Table 8.5).

Parameter	Value
Optimizer	Ranger [56] [57]
Learning rate	$1 \cdot 10^{-3}$
Batch size	32
RNN unit	Bidirectional GRU
RNN unit size	500
Number of RNN layers	1
Embedding layer size	100
Dropout rate	0.0

Table 8.2: Shared seq2seq hyperparameters

Parameter	Value
Beam search size	3

Table 8.3: Supervised learning hyperparameters

Parameter	Value
Discount factor γ	0.99
TD step size n	4
Target network update period τ	$10,000$
Number of quantiles N	5/21
Experience replay buffer size	$1,000,000$
PER α	0.6
PER β	0.4
PER β max iterations	$1,000,000$
Noisy nets σ_0	0.4
Sample generation period	8

Table 8.4: Reinforcement learning hyperparameters

Parameter	Value
# SL iterations for transfer learning	$100,000$
MTL mixing ratio (SL:RL)	$25:100$

Table 8.5: Pretraining hyperparameters

8.2. Evaluation and Discussion

In this section, the different models (see Section 8.1.1) and their aspects are evaluated and discussed. This task includes assessing the scalability in Section 8.2.1, ablating several DQN improvements in Section 8.2.3, and exploring different techniques to utilize demonstration data (see Section 8.2.4). On top of that, exemplary outputs of the various models are discussed in Section 8.2.2.

8.2.1. Scalability

Metric / Reward	**Model**		**Vocabulary Size**			
			111	201	401	806
BLEU	SL		0.71	0.70	0.77	**0.77**
	RL		**0.74**	**0.71**	**0.81**	0.71
	MTL	SL Layer	0.73	**0.71**	**0.81**	0.67
		RL Layer	0.73	**0.71**	0.75	0.42
ROUGE	SL		**0.6**	0.60	0.63	0.60
	RL		**0.6**	**0.63**	**0.65**	**0.61**
	MTL	SL Layer	0.58	0.61	0.64	0.49
		RL Layer	0.57	0.60	0.60	0.36

Table 8.6: Evaluation of presented models (supervised learning [SL], reinforcement learning [RL], multitask learning [MTL]) in different setups to assess the scalability

Multiple experiments have been conducted to assess the scalability of the presented model. Generally, four settings are considered. In each case, the action space is approximately doubled, resulting in vocabulary sizes of 111, 201, 401 and 806. The dataset size exhibits disproportionate growth ranging from 1,311 to 18,523 examples.

The results, which are displayed in Table 8.6, demonstrate that it is possible to train a seq2seq network with the methods of value-based reinforcement learning. For limited problem sizes, these methods are able to match or even surpass ambitious baselines such as supervised trained models in their stronghold settings. While conducting the experiments, however, it became

evident that the model is subject to scalability constraints. With the initial parameter setting of $N = 21$ for the number of quantiles, it was not possible to scale to an action space size of 401 without a drop in performance. Instead, the hyperparameter had to be reduced to 5, which seems to lift the upper limit of its scalability to the 806 setting.

8.2.2. Exemplary Outputs

The evaluation in this section refers to the 401 setting, as this is the setting for which both RL models are still able to match or surpass the supervised trained model on the respective evaluation metric. Generally, the trained models produce largely similar results. The RL^{BLEU} model outputs the same sequences as the supervised model for 80.92 per cent of unique sources in the corpus. A similar outcome holds for the RL^{ROUGE} model with 80.80%. Even among themselves, they share outputs in a similar order of magnitude (i.e., 81.69%).

Nevertheless, in Table 8.7, ten random examples are presented for which the models returned different sequences. From studying these examples, possible reasons for the differences include, that the target is still incompletely or partly learned or, given the nature of the dataset, in which a source can have several targets, the models simply approach different targets. Interestingly, there are also a few examples that reveal the very distinct way of learning. For instance, the RL^{ROUGE} model outputs "*What Why?*" for "*What time is it?*" and scores a medium reward on the "*Why?*" and "*What?*" target with it. However, this output sequence ultimately yields a higher combined reward on ROUGE-W (1.51) than predicting either of the targets (1.396). This example highlights the importance of choosing an adequate reward and the necessity that such a reward encompasses "semantic coherence" as in [31].

Source	Target(s)	RL^{BLEU}	RL^{ROUGE}	SL
Of course we are. Of course.	Then you'll be my best man.	Then you'll be my best man.	Then you'll be my best, go.	Then you'll be fine.
And if I didn't want to come?	I guess I'd ask you.	I guess I'd ask you to ask me.	I guess I'd ask you why.	I guess I'd ask you to ask me.
So did you do it?	Did I kill him ? Of course not.	Did I kill him?	Did I kill him? Of course not!	
What time is it?	– Why? – What?	Why?	What Why?	What?
You believe that?	You don't believe me? Okay, fine.	I do if you do.	No.	You think bad, and bad is what you get.
Are you okay?	– I'm fine. – I'll be fine. – She'll be fine. – I think so. – … (*8 more targets*)	I'll be fine.	I'm fine.	She'll be fine.
Yeah.	*132 targets*	Yeah.?	I like nothing I bad?. say	You're with me now.
She's not at home?	No … they said she was here …	No … they said she was it.	No … they said she want, that's!	No … they didn't < u > here –
She's out with a friend.	Oh … do you know when she'll be home?	Oh … do you know when she'll be back?	Oh … do you know when she'll be?	Oh … do you know when she'll be home?
Now what?	– We just don't stop. – Now what what? – Don't go. – … (*2 more targets*)	Don't go.	We don't know it.	Now what what?

Table 8.7: Examples for which the presented models generate different responses; the models included are the reinforcement learning models conditioned on either BLEU or ROUGE (RL^{BLEU}, RL^{ROUGE}) and the supervised trained model (SL).

8.2.3. Ablation Study

The presented model, which is based on Rainbow, combines several DQN improvements. In order to reach a better understanding of how these extensions behave in this work's high-dimensional sequence-to-sequence setup, an ablation study was conducted. In this study, the influence on performance and convergence of some algorithmic ingredients was assessed by removing them one at a time from the complete setup. The results are displayed in Figure 8.1.

# Quantiles N	# Parameters (401)	# Parameters (806)
No Distributional RL	4.617,305	5,306,820
2	5,020,109	6,115,434
5	6,228,521	8,541,276
21	12,673,385	21,479,100
51	24,757,505	45,737,520

Table 8.8: Influence of the number of quantiles N on the number of parameters

Dueling nets. Although the dueling architecture is motivated by problems with many similar-valued actions and larger action spaces, the ablation study does not indicate a significant impact of the extension on the model. No effect is observed, either in terms of performance or in light of convergence speed. These results are in line with findings by [18]. It is notable, however, that the performance of the dueling architecture lags behind in the early stages of training, although it catches up later.

Prioritized experience replay. This extension is an essential contribution to scalability and performance, given that the results unambiguously and significantly worsen for the ablated model. To a certain extent, these results can also validate the modifications made to prioritized experience replay in Section 6.3.2. Presumably, PER is especially effective in this setup because the memory is preloaded with demonstration data (see Section 6.5.1).

Multi-step learning. Based on the ablation study, it is clear that the most influential extension is multi-step learning. The ablated model did not learn at all for a vocabulary size of 806, while it performed only slightly better in a setup with 401. [18] also found it to be the most important extension, with ablation resulting in a substantial drop in early and final performance. However, the adverse effects of ablation are much more strongly reflected in the thesis' setup. Presumably, the reasons for this outcome are to be found in the different nature of the problem. Here, rewards are issued only at the very end of the sequence, which is cataclysmic in combination with the last action always being the end token. In this particular case, only $q(s, \texttt{</s>})$ is able to obtain immediate, unbiased targets. Conversely, all the other actions' targets can rely solely on the model itself, via bootstrapping.

Distributional reinforcement learning. The quantile regression extension exercises a noticeable effect on the performance. That said, the model is highly sensitive to the number of quantiles N chosen. In the original paper, [27], the authors suggested N to be 32. However, the researchers only probed their models on small action spaces, which are different in magnitude compared to this thesis' setup. In general, distributional RL requires the model

to learn more and make auxiliary predictions, increasing the difficulty of the task while introducing synergy effects and easing approximation. Moreover, the output layer size is defined by $|\mathcal{A}| \cdot N$, which will dominate the model's size and complexity for larger action spaces, introducing a disproportion between the problem's complexity and the model's complexity. This relationship is shown in Table 8.8. While $N = 21$ works well on vocabulary sizes of 111 and 201, it already slightly hurts performance for 401, and it fails for 806. For the latter two sizes, a value between 2 and 5 seems to be a reasonable choice. By means of these parameters, distributional reinforcement learning contributes to scalability and final performance. To conclude, there are presumably two opposite effects resulting from QR-DQN: the model benefits from learning auxiliary tasks while a larger quantile number leads to a larger model and more difficult prediction task, necessitating a careful trade-off.

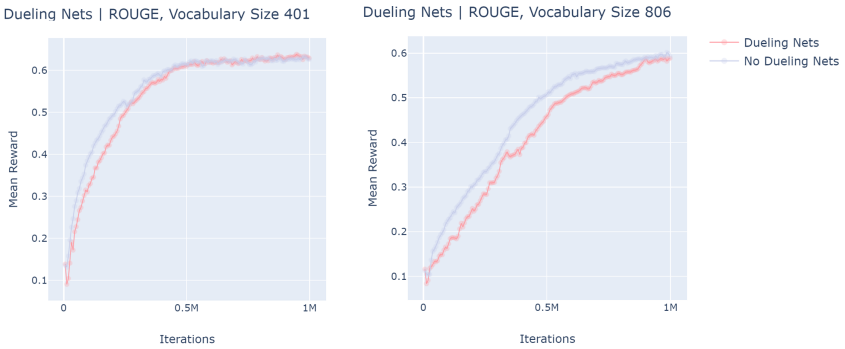
8.2.4. Utilization of Demonstration Data

This section deals with techniques intended to utilize the provided demonstration data. These techniques are introduced in Section 6.5. The learning curves for the different models are depicted in Figure 8.2.

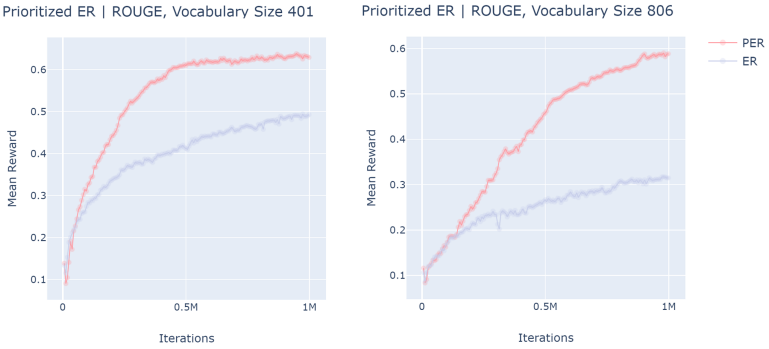
Preloading PER. Preloading the prioritized replay buffer is demonstrated to be a key element in all presented settings. With preloading, the model does not need to rely on random sampling only but can utilize demonstration data. The number of potential sequences grows exponentially with the vocabulary size, which is why, without preloading, the model is practically unable to learn at all in a setting with a vocabulary size of 806, while it seems to converge to a suboptimal solution in a 401 setting. However, this result is notable, because in other settings like [58] preloading had less of an impact. Moreover, in a replay buffer with up to 1 million entries, the amount of demonstration data is vanishingly small.

Transfer learning. Transferring the weights of the supervised learning model to the GRU and embedding layers of the RL model worsens performance significantly. While these models tend to start slightly better compared to pure RL settings, their learning curve quickly flattens, and they converge to suboptimal solutions. Through transfer learning, the hypothesis space seems to be narrowed in a disadvantageous way, hinting that predicting token probabilities based on the CE loss and predicting Q-values based on BLEU or ROUGE as a reward are very different tasks. This outcome suggests the problem is more suitable to be framed in a multitask learning than a transfer learning setup.

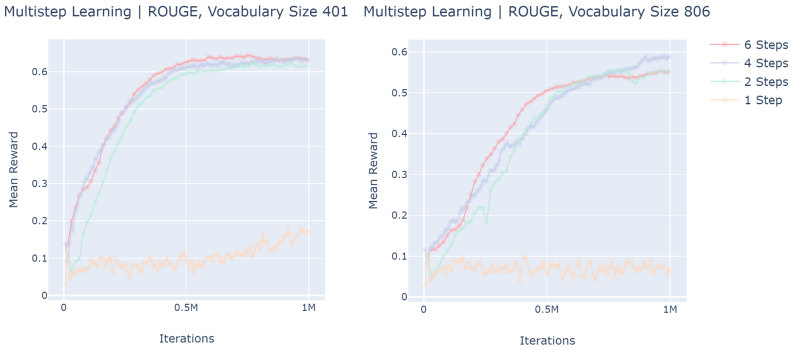
Multitask learning. The experiments conducted show multitask learning models to have a strong early performance (i.e., a comparably quite steep increase of the average reward yielded in early training stages). However, these models reach a premature plateau before they eventually diverge. Nevertheless, this result may suggest there is potential in this approach, while further exploration of multitask learning setups may help overcome the caveats in this specific setting.



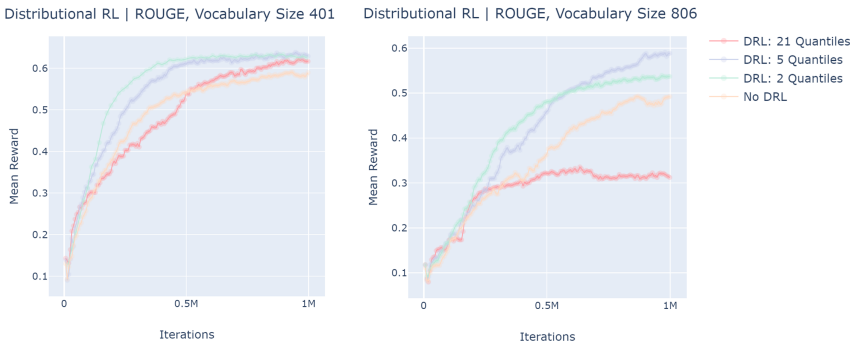
(a) Ablated dueling nets



(b) Ablated prioritized experience replay

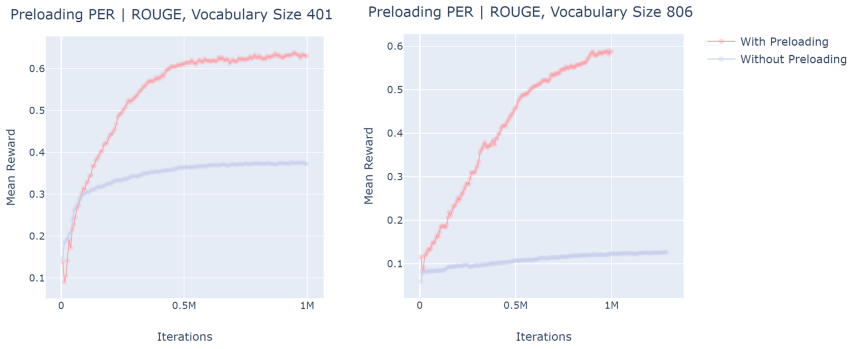


(c) Ablated multi-step learning

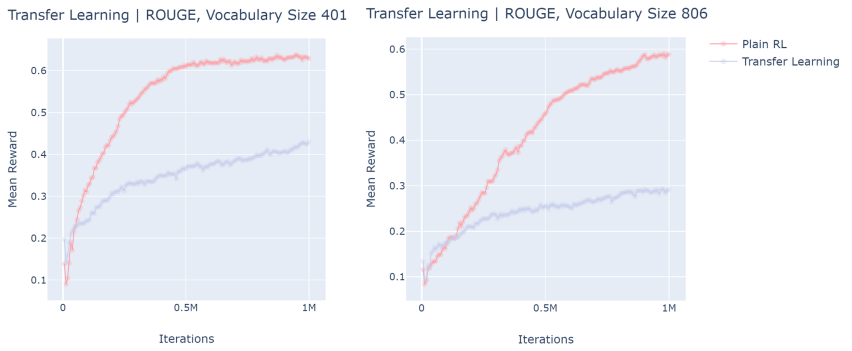


(d) Ablated distributional reinforcement learning

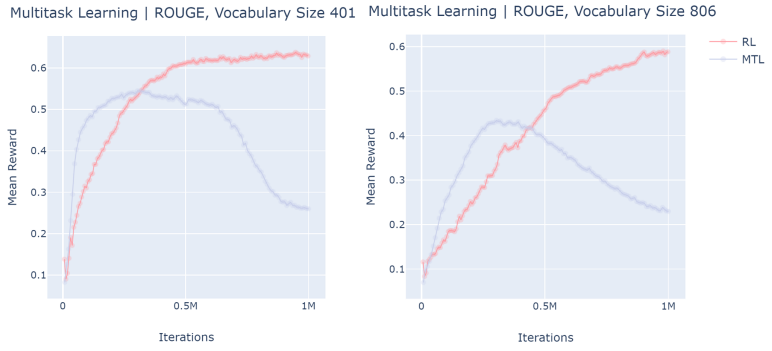
Figure 8.1: Ablation for different DQN improvements



(a) Preloading the prioritized experience replay buffer



(b) Utilizing transfer learning



(c) Multitask learning

Figure 8.2: Evaluating pretraining techniques

9

Conclusion

9.1. Summary

In this thesis, a framework was developed, allowing the application of value-based reinforcement learning methods to sequence-to-sequence models for the first time. This framework contrasts sharply with existing approaches which focus solely on policy-gradient methods and actor-critic setups because they are easy to pretrain. However, this thesis follows a long-term goal of making reinforcement learning approaches usable in the area of natural language processing beyond the fine-tuning of supervised trained models.

The presented model demonstrates the theoretical possibility of training a sequence-to-sequence model in a Rainbow setup, a state-of-the-art single-actor DQN agent. In practice, such a model is still highly limited by its scalability. However, it is the first step towards a generally applicable approach and an important baseline for future improvements. Furthermore, the ablation study included here provides valuable insights into the behaviour of several DQN improvements in a high-dimensional NLP setup. More specifically, multi-step learning, prioritized experience replay and distributional reinforcement learning were found to be essential components enabling the model to learn in

the investigated settings. Additionally, the thesis explored how demonstration data can be utilized. In this context, the preloading of the replay buffer with such data was identified as an indispensable prerequisite for learning in higher-dimensional spaces.

9.2. Future Work

There are several directions in which this research can be furthered. Particular attention should be given to the question of how scalability can be improved.

Recent improvements on single-actor DQNs. While Rainbow is still considered to be state-of-the-art, there have recently been some major improvements in the area of distributional reinforcement learning. Models such as IQN [28] and FQF [29] already match or even surpass the performance of Rainbow, even without combining orthogonal enhancements. Both papers encourage using their approaches to distributional RL in a Rainbow-like setup. Also, they might be especially effective for high-dimensional spaces as they avoid the excessive growth of the output layer with the number of quantiles, which was presumed to be especially hurtful in this thesis.

Distributed DQNs. A potential approach for significantly improving the model's scalability is to switch to a distributed architecture. By decoupling data collection and learning, models such as ApeX [59], R2D2 [46], and Agent57 [60] are able to increase final performance substantially while reducing wall-clock learning speed against all single-actor agents.

Dealing with high dimensionality. Several tricks and methods are primarily motivated by vast action spaces. Most recently, a promising contribution was made with amortized Q-learning (AQL) in [61], whose method relies on a proposal network to suggest potential actions. With value penalties, as used in [4] or [62], an additional loss component is added, which penalizes variance

on outputs helping with rare actions. Another idea is action branching proposed by [63], who architecturally divide the action space into smaller chunks. However, it has not yet been examined if and in which way these approaches can be applied in the context of a Rainbow seq2seq network.

Representation of the action space. It might be beneficial for RL problems, especially value-based methods, if actions are not formed at word-level, but at byte-level or character-level, resulting in a significant reduction of the action space. There is a chance that DQNs are able to cope better with longer action sequences than an increased action space.

Besides scalability, there are other interesting research questions: for instance, the application of reinforcement learning on state-of-the-art NLP architectures such as transformers. Moreover, it might be explored how the presented model behaves with rewards, that are more different from the maximum likelihood objective.

Bibliography

[1] Jianfeng Gao, Michel Galley, and Lihong Li. "Neural Approaches to Conversational AI". In: *CoRR* abs/1809.08267 (2018). arXiv: 1809.08267.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2015.

[5] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025.

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. issn: 0899-7667.

[7] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078.

[8] Yaser Keneshloo et al. "Deep Reinforcement Learning For Sequence to Sequence Models". In: *CoRR* abs/1805.09461 (2018). arXiv: 1805.09461.

[9] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2692–2700.

[10] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762.

[11] Alexander H. Waibel et al. “Phoneme recognition using time-delay neural networks”. In: *IEEE Trans. Acoust. Speech Signal Process.* 37 (1989), pp. 328–339.

[12] Yann Lecun and Yoshua Bengio. “Convolutional networks for images, speech, and time-series”. English (US). In: *The handbook of brain theory and neural networks*. Ed. by M.A. Arbib. MIT Press, 1995.

[13] Jonas Gehring et al. “Convolutional Sequence to Sequence Learning”. In: *CoRR* abs/1705.03122 (2017). arXiv: 1705.03122.

[14] Marc'Aurelio Ranzato et al. “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*. 2016.

[15] Samy Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *CoRR* abs/1506.03099 (2015). arXiv: 1506.03099.

[16] Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning - An Introduction*. Cambridge: MIT Press, 2018. isbn: 978-0-262-03924-6.

[17] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602.

[18] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *CoRR* abs/1710.02298 (2017). arXiv: 1710.02298.

[19] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn.* 8.3-4 (May 1992), pp. 229–256. issn: 0885-6125.

[20] Steven J. Rennie et al. “Self-critical Sequence Training for Image Captioning”. In: *CoRR* abs/1612.00563 (2016). arXiv: 1612.00563.

[21] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783.

[22] John Schulman, Pieter Abbeel, and Xi Chen. "Equivalence Between Policy Gradients and Soft Q-Learning". In: *CoRR* abs/1704.06440 (2017). arXiv: 1704.06440.

[23] Ofir Nachum et al. "Bridging the Gap Between Value and Policy Based Reinforcement Learning". In: *CoRR* abs/1702.08892 (2017). arXiv: 1702.08892.

[24] Tom Schaul et al. "Prioritized Experience Replay". In: *CoRR* abs/1511.05952 (2015).

[25] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581.

[26] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *CoRR* abs/1707.06887 (2017). arXiv: 1707.06887.

[27] Will Dabney et al. "Distributional Reinforcement Learning with Quantile Regression". In: *CoRR* abs/1710.10044 (2017). arXiv: 1710.10044.

[28] Will Dabney et al. "Implicit Quantile Networks for Distributional Reinforcement Learning". In: *CoRR* abs/1806.06923 (2018). arXiv: 1806.06923.

[29] Derek Yang et al. "Fully Parameterized Quantile Function for Distributional Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 6190–6199. arXiv: 1911.02140 [cs.LG].

[30] Meire Fortunato et al. "Noisy Networks for Exploration". In: *CoRR* abs/1706.10295 (2017). arXiv: 1706.10295.

[31] Jiwei Li et al. "Deep Reinforcement Learning for Dialogue Generation". In: *CoRR* abs/1606.01541 (2016). arXiv: 1606.01541.

[32] Dzmitry Bahdanau et al. "An Actor-Critic Algorithm for Sequence Prediction". In: *CoRR* abs/1607.07086 (2016). arXiv: 1607.07086.

[33] Jiwei Li et al. "A Diversity-Promoting Objective Function for Neural Conversation Models". In: *CoRR* abs/1510.03055 (2015). arXiv: 1510.03055.

[34] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon". In: *Commun. ACM* 38.3 (Mar. 1995), pp. 58–68. issn: 0001-0782.

[35] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (jan 2016), pp. 484–489. issn: 0028-0836.

[36] Julia Kreutzer, Joshua Uyheng, and Stefan Riezler. "Reliability and Learnability of Human Bandit Feedback for Sequence-to-Sequence Reinforcement Learning". In: *arXiv e-prints*, arXiv:1805.10627 (May 2018), arXiv:1805.10627. arXiv: 1805.10627 [cs.CL].

[37] Romain Paulus, Caiming Xiong, and Richard Socher. "A Deep Reinforced Model for Abstractive Summarization". In: *CoRR* abs/1705.04304 (2017). arXiv: 1705.04304.

[38] Jiwei Li et al. "Adversarial Learning for Neural Dialogue Generation". In: *arXiv e-prints*, arXiv:1701.06547 (Jan. 2017), arXiv:1701.06547. arXiv: 1701.06547 [cs.CL].

[39] Pararth Shah et al. "Building a Conversational Agent Overnight with Dialogue Self-Play". In: *arXiv preprint arXiv:1801.04871* (2018).

[40] Matthew Henderson, Blaise Thomson, and Jason D. Williams. "The Second Dialog State Tracking Challenge". In: *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Philadelphia, PA, U.S.A.: Association for Computational Linguistics, June 2014, pp. 263–272.

[41] Antoine Bordes and Jason Weston. "Learning End-to-End Goal-Oriented Dialog". In: *CoRR* abs/1605.07683 (2016). arXiv: 1605.07683.

[42] Tiancheng Zhao and Maxine Eskénazi. "Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning". In: *CoRR* abs/1606.02560 (2016). arXiv: 1606.02560.

[43] Xiujun Li et al. "End-to-End Task-Completion Neural Dialogue Systems". In: *CoRR* abs/1703.01008 (2017). arXiv: 1703.01008.

[44] Bing Liu et al. "Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems". In: *CoRR* abs/1804.06512 (2018). arXiv: 1804.06512.

[45] Matthew J. Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *CoRR* abs/1507.06527 (2015). arXiv: 1507.06527.

[46] Steven Kapturowski et al. "Recurrent Experience Replay in Distributed Reinforcement Learning". In: *International Conference on Learning Representations*. 2019.

[47] Jörg Tiedemann and Pierre Lison. "OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles". In: *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. 2016.

[48] Cristian Danescu-Niculescu-Mizil and Lillian Lee. "Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs." In: *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*. 2011.

[49] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318.

[50] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81.

[51] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792.

[52] Ronan Collobert and Jason Weston. "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 160–167. isbn: 978-1-60558-205-4.

[53] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083.

[54] Rich Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: *ICML*. 1993.

[55] Guillaume Klein et al. “OpenNMT: Open-Source Toolkit for Neural Machine Translation”. In: *Proc. ACL*. 2017.

[56] Liyuan Liu et al. *On the Variance of the Adaptive Learning Rate and Beyond*. Aug. 2019.

[57] Michael R. Zhang et al. “Lookahead Optimizer: k steps forward, 1 step back”. In: *CoRR* abs/1907.08610 (2019). arXiv: 1907.08610.

[58] Emil Larrson. *Evaluation of Pretraining Methods for Deep Reinforcement Learning*. 2018.

[59] Dan Horgan et al. “Distributed Prioritized Experience Replay”. In: *CoRR* abs/1803.00933 (2018). arXiv: 1803.00933.

[60] Adrià Badia et al. *Agent57: Outperforming the Atari Human Benchmark*. Mar. 2020.

[61] Tom Van de Wiele et al. “Q-Learning in enormous action spaces via amortized approximate maximization”. In: *CoRR* abs/2001.08116 (2020). arXiv: 2001.08116.

[62] Wojciech Zaremba et al. “Learning Simple Algorithms from Examples”. In: *CoRR* abs/1511.07275 (2015). arXiv: 1511.07275.

[63] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. “Action Branching Architectures for Deep Reinforcement Learning”. In: *CoRR* abs/1711.08946 (2017). arXiv: 1711.08946.