Lecture 3: An Actor-Critic Algorithm For Sequence Predictions

Bahdanau et al.

Presenter: Zhihang Dong;
Adv. Natural Lang. Proc. Seminar

May 9, 2018

Table of contents

Motivation

Overarching Goal

In many cases of natural language processing such as machine translation, caption generation and speech recognition,

the **task** is to **develop a system that produces a sequence of discrete tokens given an input**.

Previous Approaches

Train recurrent neural networks (RNNs) to generate sequences

- Maximize the log-likelihood of the correct tokens given a history of previous 'correct ones'

Previous Approaches

Train recurrent neural networks (RNNs) to generate sequences

- Maximize the log-likelihood of the correct tokens given a history of previous 'correct ones'
- this is also called *teacher forcing*

Previous Approaches

Train recurrent neural networks (RNNs) to generate sequences

- Maximize the log-likelihood of the correct tokens given a history of previous 'correct ones'
- this is also called *teacher forcing*
- the output sequence is often produced by an approximate search for the most likely candidate according to the learned distribution

Problems

The model is conditioned on its own guesses (which may be correct or incorrect), hence a larger ratio of incorrect guessed tokens would lead to a large compound of errors

- with longer sequences, this is especially problematic...

Problems

The model is conditioned on its own guesses (which may be correct or incorrect), hence a larger ratio of incorrect guessed tokens would lead to a large compound of errors

- with longer sequences, this is especially problematic...
- the discrepancies between training and test set also leads to suboptimal predictions

Problems

The model is conditioned on its own guesses (which may be correct or incorrect), hence a larger ratio of incorrect guessed tokens would lead to a large compound of errors

- with longer sequences, this is especially problematic...
- the discrepancies between training and test set also leads to suboptimal predictions
- this leads to reinforcement learning, hence evaluations of them such as BLEU and ROUGE

Backgrounds

Question

consider the problem of learning to produce an output sequence $Y = (y_1, ..., y_T), y_t \in \mathcal{A}$ given an input $\mathcal{X}$, where $\mathcal{A}$ is the alphabet of output tokens. We will often use notation $Y_{f...l}$ to refer to subsequences of the form $(y_f, ..., y_l)$. Two sets of input-output pairs (X, Y) are assumed to be available for both training and testing.

- The trained predictor h is evaluated by computing the average task-specific score $R(\hat{Y}, Y)$ on the test set

Question

consider the problem of learning to produce an output sequence $Y = (y_1, ..., y_T), y_t \in \mathcal{A}$ given an input $\mathcal{X}$, where $\mathcal{A}$ is the alphabet of output tokens. We will often use notation $Y_{f...l}$ to refer to subsequences of the form $(y_f, ..., y_l)$. Two sets of input-output pairs (X, Y) are assumed to be available for both training and testing.

- The trained predictor h is evaluated by computing the average task-specific score $R(\hat{Y}, Y)$ on the test set
- where $\hat{Y} = h(X)$ is the prediction

Question

consider the problem of learning to produce an output sequence $Y = (y_1, ..., y_T), y_t \in \mathcal{A}$ given an input $\mathcal{X}$, where $\mathcal{A}$ is the alphabet of output tokens. We will often use notation $Y_{f...l}$ to refer to subsequences of the form $(y_f, ..., y_l)$. Two sets of input-output pairs (X, Y) are assumed to be available for both training and testing.

- The trained predictor h is evaluated by computing the average task-specific score $R(\hat{Y}, Y)$ on the test set
- where $\hat{Y} = h(X)$ is the prediction
- For simplicity, let's use use T to denote the length of an output sequence, ignoring the fact that the output sequences may have different length

RNNs: Set up

The RNN that produces a sequence of state vectors $(s_1, \cdots s_T)$ given a sequence of input vectors $(e_1, \cdots, e_T)$

- which starts from an initial s_0 state and applying T times the transition function $f : s_t = f(s_{t-1}, e_t)$.

RNNs: Set up

The RNN that produces a sequence of state vectors $(s_1, \cdots s_T)$ given a sequence of input vectors $(e_1, \cdots, e_T)$

- which starts from an initial s_0 state and applying T times the transition function $f : s_t = f(s_{t-1}, e_t)$.
- the choice of mapping is **Gated Recurrent Units**

RNNs: Set up

The RNN that produces a sequence of state vectors $(s_1, \cdots s_T)$ given a sequence of input vectors $(e_1, \cdots, e_T)$

- which starts from an initial s_0 state and applying T times the transition function $f : s_t = f(s_{t-1}, e_t)$.
- the choice of mapping is **Gated Recurrent Units**
- the authors added a stochastic output layer g to build a probabilistic sequence generation to feed back outputs by replacing them with their embedding denoted as $e(y_t)$

RNNs: Set up

The RNN that produces a sequence of state vectors $(s_1, \cdots s_T)$ given a sequence of input vectors $(e_1, \cdots, e_T)$

- which starts from an initial s_0 state and applying T times the transition function $f : s_t = f(s_{t-1}, e_t)$.
- the choice of mapping is **Gated Recurrent Units**
- the authors added a stochastic output layer g to build a probabilistic sequence generation to feed back outputs by replacing them with their embedding denoted as $e(y_t)$
-

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

$$s_t = f(s_{t-1}, e(y_t))$$

RNNs: Predictions

Therefore, RNN defines a probability distribution $p(y_t|y_1, ..., y_{t-1})$ of the next output token y_t given the previous tokens upon adding a special end-of-sequence token...

- augment to generate Y conditioned on an input X. The simplest way to do this is to start with an initial state $s_0 = s_0(X)$

RNNs: Predictions

Therefore, RNN defines a probability distribution $p(y_t|y_1, ..., y_{t-1})$ of the next output token y_t given the previous tokens upon adding a special end-of-sequence token...

- augment to generate Y conditioned on an input X. The simplest way to do this is to start with an initial state $s_0 = s_0(X)$
- the sequence of vectors is produced by either a bidirectional RNN or a convolutional encoder

RNNs: Attention Modeling

The author uses soft attention mechanism that computes a weighted sum of a sequence of vectors, then this mechanism determines the relative importance of each vector

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

RNNs: Attention Modeling

The author uses soft attention mechanism that computes a weighted sum of a sequence of vectors, then this mechanism determines the relative importance of each vector

- $y_t \sim g(s_{t-1}, ct-1)$
- $s_t = f(s_{t-1}, c_{t-1}, e(y_t)$

RNNs: Attention Modeling

The author uses soft attention mechanism that computes a weighted sum of a sequence of vectors, then this mechanism determines the relative importance of each vector

- $y_t \sim g(s_{t-1}, ct-1)$
- $s_t = f(s_{t-1}, c_{t-1}, e(y_t)$
- $\alpha_t = \beta(s_t, (h_1, ..., h_L)$

RNNs: Attention Modeling

The author uses soft attention mechanism that computes a weighted sum of a sequence of vectors, then this mechanism determines the relative importance of each vector

- $y_t \sim g(s_{t-1}, ct-1)$
- $s_t = f(s_{t-1}, c_{t-1}, e(y_t)$
- $\alpha_t = \beta(s_t, (h_1, ..., h_L)$
- $c_t = \sum_{j=1}^{L} \alpha_{t,j} h_j$

RNNs: Attention Modeling

The author uses soft attention mechanism that computes a weighted sum of a sequence of vectors, then this mechanism determines the relative importance of each vector

- $y_t \sim g(s_{t-1}, ct-1)$
- $s_t = f(s_{t-1}, c_{t-1}, e(y_t)$
- $\alpha_t = \beta(s_t, (h_1, ..., h_L)$
- $c_t = \sum_{j=1}^{L} \alpha_{t,j} h_j$
- Here, β is the attention mechanism, α_t is the attention weight, c_t is the context vector given step t

Value functions

Suggesting a conditioned RNN (by gradient ascent on the log-likelihood $p(Y|X)$) as a *stochastic policy* that generates *actions*, which result in a task score (which is also called *return*).

- return R is partially received at the intermediate steps in the form of rewards r_t : $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t, \hat{Y}_{1,\ldots,t-1}, Y)$

Value functions

Suggesting a conditioned RNN (by gradient ascent on the log-likelihood $p(Y|X)$) as a *stochastic policy* that generates *actions*, which result in a task score (which is also called *return*).

- return R is partially received at the intermediate steps in the form of rewards r_t : $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t, \hat{Y}_{1,\ldots,t-1}, Y)$
- Hence we have the value of an unfinished predictions $\hat{Y}_{1,\ldots,t}$:

Value functions

Suggesting a conditioned RNN (by gradient ascent on the log-likelihood $p(Y|X)$) as a *stochastic policy* that generates *actions*, which result in a task score (which is also called *return*).

- return R is partially received at the intermediate steps in the form of rewards r_t : $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t, \hat{Y}_{1,\ldots,t-1}, Y)$
- Hence we have the value of an unfinished predictions $\hat{Y}_{1,\ldots,t}$:
-

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

Value functions

Suggesting a conditioned RNN (by gradient ascent on the log-likelihood $p(Y|X)$) as a *stochastic policy* that generates *actions*, which result in a task score (which is also called *return*).

- return R is partially received at the intermediate steps in the form of rewards r_t : $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t, \hat{Y}_{1,\ldots,t-1}, Y)$
- Hence we have the value of an unfinished predictions $\hat{Y}_{1,\ldots,t}$:
- $$V(\hat{Y}_{1,\ldots,t}, X, Y_) = \mathbb{E}_{\hat{Y}_{t+1,\ldots T} \sim p(\cdot|\hat{Y},X)} \sum_{\gamma=t+1}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1,\ldots\tau-1}, Y)$$
- We define the value of a candidate next token a for an unfinished prediction $\hat{Y}_{1,\ldots,t-1}$ as the expected future return after generating token a

Value functions

Suggesting a conditioned RNN (by gradient ascent on the log-likelihood $p(Y|X)$) as a *stochastic policy* that generates *actions*, which result in a task score (which is also called *return*).

- return R is partially received at the intermediate steps in the form of rewards r_t : $R(\hat{Y}, Y) = \sum_{t=1}^{T} r_t(\hat{y}_t, \hat{Y}_{1,\ldots,t-1}, Y)$
- Hence we have the value of an unfinished predictions $\hat{Y}_{1,\ldots,t}$:
- $$V(\hat{Y}_{1,\ldots,t}, X, Y_) = \mathbb{E}_{\hat{Y}_{t+1,\ldots T} \sim p(\cdot|\hat{Y},X)} \sum_{\gamma=t+1}^{T} r_\tau(\hat{y}_\tau; \hat{Y}_{1,\ldots\tau-1}, Y)$$
- We define the value of a candidate next token a for an unfinished prediction $\hat{Y}_{1,\ldots,t-1}$ as the expected future return after generating token a
- We will refer to the candidate next tokens as *actions*

Actor-Critic Algorithm

Actor-Critic Algorithm: Overview

Algorithm 1 Actor-Critic Training for Sequence Prediction

Require: A critic $\hat{Q}(a; \hat{Y}_{1...t}, Y)$ and an actor $p(a|\hat{Y}_{1...t}, X)$ with weights ϕ and θ respectively.

1: Initialize delayed actor $p^{'}$ and target critic $\hat{Q}'$ with same weights: $\theta' = \theta$, $\phi' = \phi$.

2: **while** Not Converged **do**

3: Receive a random example (X, Y).

4: Generate a sequence of actions $\hat{Y}$ from $p^{'}$.

5: Compute targets for the critic

$$q_t = r_t(\hat{y}_t; \hat{Y}_{1...t-1}, Y) + \sum_{a \in \mathcal{A}} p^{'}(a|\hat{Y}_{1...t}, X)\hat{Q}'(a; \hat{Y}_{1...t}, Y)$$

6: Update the critic weights ϕ using the gradient

$$\frac{d}{d\phi}\left(\sum_{t=1}^{T}\left(\hat{Q}(\hat{y}_t; \hat{Y}_{1...t-1}, Y) - q_t\right)^2 + \lambda_C C_t\right)$$

$$\text{where } C_t = \sum_a \left(\hat{Q}(a; \hat{Y}_{1...t-1}) - \frac{1}{|\mathcal{A}|}\sum_b \hat{Q}(b; \hat{Y}_{1...t-1})\right)^2$$

7: Update actor weights θ using the following gradient estimate

$$\widehat{\frac{dV(X,Y)}{d\theta}} = \sum_{t=1}^{T}\sum_{a \in \mathcal{A}} \frac{dp(a|\hat{Y}_{1...t-1}, X)}{d\theta}\hat{Q}(a; \hat{Y}_{1...t-1}, Y) + \lambda_{LL}\sum_{t=1}^{T}\frac{dp(y_t|Y_{1...t-1}, X)}{d\theta}$$

8: Update delayed actor and target critic, with constants $\gamma_\theta \ll 1$, $\gamma_\phi \ll 1$

$$\theta' = \gamma_\theta\theta + (1 - \gamma_\theta)\theta', \ \phi' = \gamma_\phi\phi + (1 - \gamma_\phi)\phi'$$

9: **end while**

Actor-Critic Algorithm: Complete one for generating sequence predictions

Algorithm 2 Complete Actor-Critic Algorithm for Sequence Prediction

1: Initialize critic $\hat{Q}(a; \hat{Y}_{1...t}, Y)$ and actor $p(a|\hat{Y}_{1...t}, X)$ with random weights ϕ and θ respectively.

2: Pre-train the actor to predict y_{t+1} given $Y_{1...t}$ by maximizing $\log p(y_{t+1}|Y_{1...t}, X)$.
3: Pre-train the critic to estimate Q by running Algorithm 1 with fixed actor.
4: Run Algorithm 1.

Introduction: Setting Up Parameters

Let θ be the parameters of the conditioned RNN, which is the *actor*. The training algorithm is to use *policy gradient theorem and stochastic actor-critic* to rewrite the graident of expected return...

- The set up sums over actions rather than taking an expectation

Introduction: Setting Up Parameters

Let θ be the parameters of the conditioned RNN, which is the *actor*. The training algorithm is to use *policy gradient theorem and stochastic actor-critic* to rewrite the graident of expected return...

- The set up sums over actions rather than taking an expectation
- To improve, we are looking for a lower variance at the cost of significant bias, since the critic is not perfect and trained simultaneously with the actor.

Introduction: Setting Up Parameters

Let θ be the parameters of the conditioned RNN, which is the *actor*. The training algorithm is to use *policy gradient theorem and stochastic actor-critic* to rewrite the graident of expected return...

- The set up sums over actions rather than taking an expectation
- To improve, we are looking for a lower variance at the cost of significant bias, since the critic is not perfect and trained simultaneously with the actor.
- The success depends on our ability to control the bias by designing the critic network and using an appropriate training criterion for it.

Introduction: Setting Up Parameters

Let θ be the parameters of the conditioned RNN, which is the *actor*. The training algorithm is to use *policy gradient theorem and stochastic actor-critic* to rewrite the graident of expected return...

- The set up sums over actions rather than taking an expectation
- To improve, we are looking for a lower variance at the cost of significant bias, since the critic is not perfect and trained simultaneously with the actor.
- The success depends on our ability to control the bias by designing the critic network and using an appropriate training criterion for it.
- To implement this critic, we use a separate RNN parameterized by ϕ, running parallel with the actor

Introduction: Setting Up Parameters

Let θ be the parameters of the conditioned RNN, which is the *actor*. The training algorithm is to use *policy gradient theorem and stochastic actor-critic* to rewrite the graident of expected return...

- The set up sums over actions rather than taking an expectation
- To improve, we are looking for a lower variance at the cost of significant bias, since the critic is not perfect and trained simultaneously with the actor.
- The success depends on our ability to control the bias by designing the critic network and using an appropriate training criterion for it.
- To implement this critic, we use a separate RNN parameterized by ϕ, running parallel with the actor
- The model consumes the tokens $\hat{y}_t$ that the actor outputs and produces the estimates $\tilde{Q}(a; \hat{Y}_{1,\ldots,t})$

Illustration

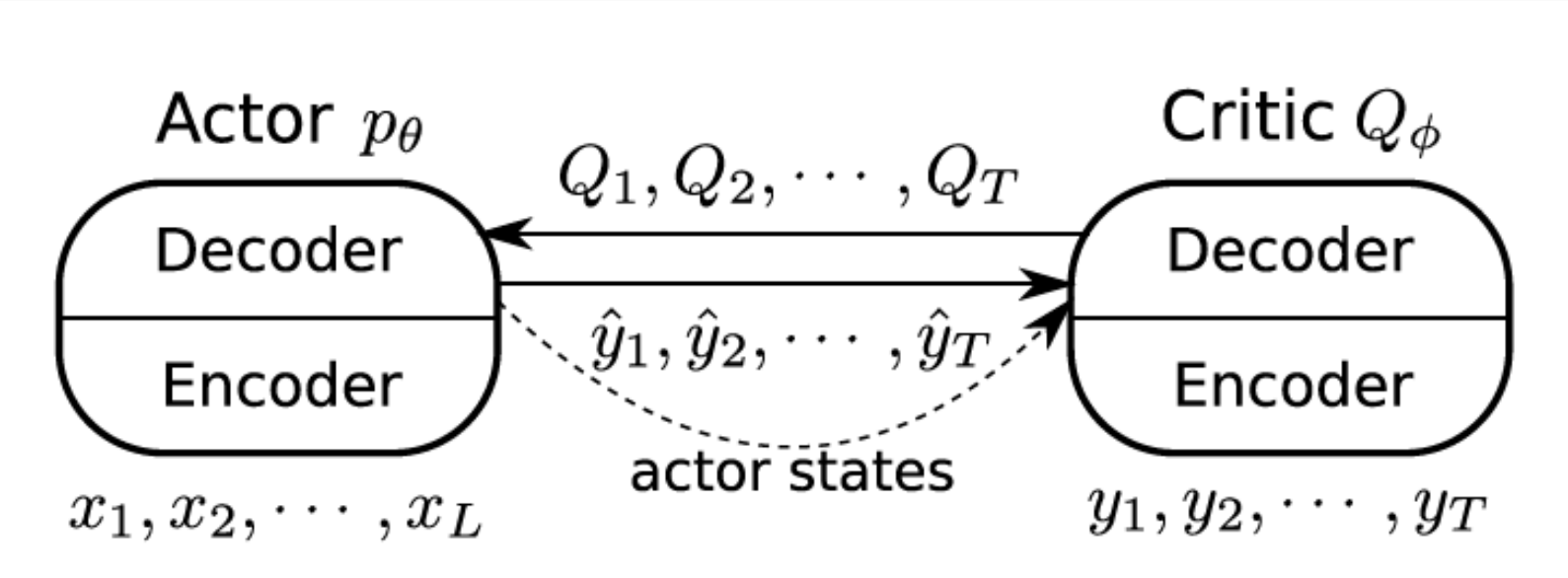

Figure 1: key difference between the critic and the actor is that the correct answer Y is given to the critic as an input, similarly to how the actor is conditioned on *X*

Temporal-Difference Learning

A crucial component of our approach is policy evaluation, that is the training of the critic to produce useful estimates of $\hat{Q}$

- With a naive Monte-Carlo method, one is likely to yield very high variance which quickly grows with the number of steps T

Temporal-Difference Learning

A crucial component of our approach is policy evaluation, that is the training of the critic to produce useful estimates of $\hat{Q}$

- With a naive Monte-Carlo method, one is likely to yield very high variance which quickly grows with the number of steps T
- Using such a target yields to very high variance which quickly grows with the number of steps T,

Temporal-Difference Learning

A crucial component of our approach is policy evaluation, that is the training of the critic to produce useful estimates of $\hat{Q}$

- With a naive Monte-Carlo method, one is likely to yield very high variance which quickly grows with the number of steps T
- Using such a target yields to very high variance which quickly grows with the number of steps T,
- Authors use a temporal difference method for policy evaluation, namely the right-hand side of the Bellman equation as the target for the left-hand

Deep RL

The RL literature that if $\hat{Q}$ is non-linear (like in our case), the TD policy evaluation might diverge

- the authors update the parameters ϕ' of the target critic by linearly interpolating them with the parameters of the trained one

Deep RL

The RL literature that if $\hat{Q}$ is non-linear (like in our case), the TD policy evaluation might diverge

- the authors update the parameters ϕ' of the target critic by linearly interpolating them with the parameters of the trained one
- Attempts to remove the target network by propagating the gradient through qt resulted in a lower square error

Deep RL

The RL literature that if $\hat{Q}$ is non-linear (like in our case), the TD policy evaluation might diverge

- the authors update the parameters ϕ' of the target critic by linearly interpolating them with the parameters of the trained one
- Attempts to remove the target network by propagating the gradient through qt resulted in a lower square error
- The fact that both actor and critic use outputs of each other for training creates a potentially dangerous feedback loop. To address this, we sample predictions from a *delayed actor*, whose weights are slowly updated to follow the actor that is actually trained

Dealing with Large Action Spaces

One of the challenges is that the action space is very large (as is typically the case in NLP tasks with large vocabularies). This can be alleviated by putting constraints on the critic values for actions that are rarely sampled.

- Experimentally that shrinking the values of these rare actions is necessary for the algorithm to converge

Dealing with Large Action Spaces

One of the challenges is that the action space is very large (as is typically the case in NLP tasks with large vocabularies). This can be alleviated by putting constraints on the critic values for actions that are rarely sampled.

- Experimentally that shrinking the values of these rare actions is necessary for the algorithm to converge
- To achieve this, the authors add a term C_t for every step t to the critic's optimization objective which drives all value predictions of the critic closer to their mean

Dealing with Large Action Spaces

One of the challenges is that the action space is very large (as is typically the case in NLP tasks with large vocabularies). This can be alleviated by putting constraints on the critic values for actions that are rarely sampled.

- Experimentally that shrinking the values of these rare actions is necessary for the algorithm to converge
- To achieve this, the authors add a term C_t for every step t to the critic's optimization objective which drives all value predictions of the critic closer to their mean
-

$$C_t = \sum_a \left(\hat{Q}(a; \hat{Y}_{1,\ldots,t-1}) - \frac{\sum_b \hat{Q}(b; \hat{Y}_{1,\ldots,t-1})}{|\mathcal{A}|} \right)^2$$

Dealing with Large Action Spaces

One of the challenges is that the action space is very large (as is typically the case in NLP tasks with large vocabularies). This can be alleviated by putting constraints on the critic values for actions that are rarely sampled.

- Experimentally that shrinking the values of these rare actions is necessary for the algorithm to converge
- To achieve this, the authors add a term C_t for every step t to the critic's optimization objective which drives all value predictions of the critic closer to their mean
- $$C_t = \sum_a \left(\hat{Q}(a; \hat{Y}_{1,\ldots,t-1}) - \frac{\sum_b \hat{Q}(b; \hat{Y}_{1,\ldots,t-1})}{|\mathcal{A}|} \right)^2$$
- which penalizes the variance of the outputs of the critic

Reward Shaping

While we are ultimately interested in the maximization of the score of a complete prediction, simply awarding this score at the last step provides a very sparse training signal for the critic.

- the authors use potential-based reward shaping with potentials $\Phi(\hat{Y}_{1,\dots t}) = R(\hat{Y}_{1,\dots t})$ for incomplete sequences and $\Phi(\hat{Y}) = 0$

Reward Shaping

While we are ultimately interested in the maximization of the score of a complete prediction, simply awarding this score at the last step provides a very sparse training signal for the critic.

- the authors use potential-based reward shaping with potentials $\Phi(\hat{Y}_{1,\dots t}) = R(\hat{Y}_{1,\dots t})$ for incomplete sequences and $\Phi(\hat{Y}) = 0$
- Namely, for a predicted sequence $\hat{Y}$ we compute score values for all prefixes to obtain the sequence of scores

Reward Shaping

While we are ultimately interested in the maximization of the score of a complete prediction, simply awarding this score at the last step provides a very sparse training signal for the critic.

- the authors use potential-based reward shaping with potentials $\Phi(\hat{Y}_{1,\dots t}) = R(\hat{Y}_{1,\dots t})$ for incomplete sequences and $\Phi(\hat{Y}) = 0$
- Namely, for a predicted sequence $\hat{Y}$ we compute score values for all prefixes to obtain the sequence of scores
- The difference between the consecutive pairs of scores is then used as the reward at each step

Experiment

Introduction to the Experiment Settings

To validate their approach, the authors performed two sets of experiments. First, they trained the proposed model to recover strings of natural text from their corrupted versions.

Specifically, they consider each character in a natural language corpus and with some probability replace it with a random character.

Their second series of experiments is done on the task of automatic machine translation using different models and datasets.

Experiment Results I

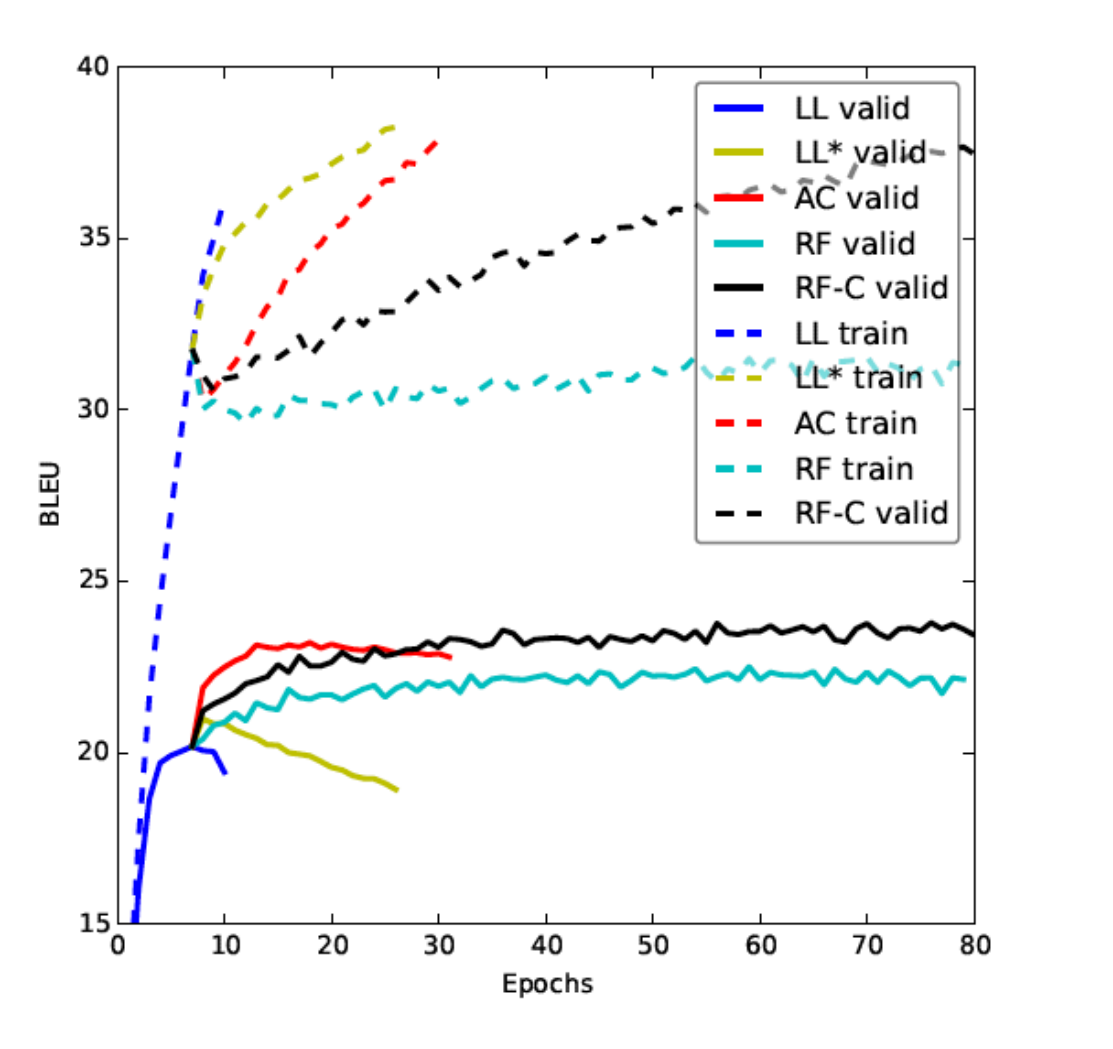

Figure 2: Data: One Billion Word dataset for the spelling correction task;The curves start from the epoch of log-likelihood pretraining from which the parameters were initialized

Other Data

- **IWSLT 2014 with a convolutional encoder**: German-English machine translation track of the IWSLT 2014 evaluation campaign

Other Data

- **IWSLT 2014 with a convolutional encoder**: German-English machine translation track of the IWSLT 2014 evaluation campaign
- **IWSLT 2014 with a bidirectional GRU encoder**: repeated the IWSLT 2014 investigation with a different encoder, a bidirectional RNN with 256 GRU units

Other Data

- **IWSLT 2014 with a convolutional encoder**: German-English machine translation track of the IWSLT 2014 evaluation campaign
- **IWSLT 2014 with a bidirectional GRU encoder**: repeated the IWSLT 2014 investigation with a different encoder, a bidirectional RNN with 256 GRU units
- **WMT14**: WMT14 English-French dataset (Cho et al., 2014)

Experiment Results II

Table 1: Character error rate of different methods on the spelling correction task. In the table L is the length of input strings, η is the probability of replacing a character with a random one. LL stands for the log-likelihood training, AC and RF-C and for the actor-critic and the REINFORCE-critic respectively, AC+LL and RF-C+LL for the combinations of AC and RF-C with LL.

Setup	Character Error Rate				
	LL	AC	RF-C	AC+LL	RF-C+LL
$L = 10, \eta = 0.3$	17.81	17.24	17.82	**16.65**	16.97
$L = 30, \eta = 0.3$	18.4	17.31	18.16	**17.1**	17.47
$L = 10, \eta = 0.5$	38.12	35.89	35.84	**34.6**	35
$L = 30, \eta = 0.5$	40.87	37.0	37.6	**36.36**	36.6

Table 2: Our IWSLT 2014 machine translation results with a convolutional encoder compared to the previous work by Ranzato et al. Please see 1 for an explanation of abbreviations. The asterisk identifies results from (Ranzato et al., 2015). The numbers reported with $\leq$ were approximately read from Figure 6 of (Ranzato et al., 2015)

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

out of four settings. In the fourth case, actor-critic and REINFORCE-critic have similar performance. Adding the log-likelihood gradient with a cofficient $\lambda_{LL} = 0.1$ helps both of the methods, but actor-critic still retains a margin of improvement over REINFORCE-critic.

Experiment Results III

Table 3: Our IWSLT 2014 machine translation results with a bidirectional recurrent encoder compared to the previous work. Please see Table 1 for an explanation of abbreviations. The asterisk identifies results from (Wiseman & Rush, 2016).

Decoding method	Model						
	LL*	BSO*	LL	RF-C	RF-C+LL	AC	AC+LL
greedy search	22.53	23.83	25.82	27.42	**27.7**	27.27	27.49
beam search	23.87	25.48	27.56	27.75	28.3	27.75	**28.53**

Table 4: Our WMT 14 machine translation results compared to the previous work. Please see Table 1 for an explanation of abbreviations. The apostrophy and the asterisk identify results from (Bahdanau et al., 2015) and (Shen et al., 2015) respectively.

Decoding method	Model					
	LL'	LL*	MRT *	LL	AC+LL	RF-C+LL
greedy search	n/a	n/a	n/a	29.33	**30.85**	29.83
beam search	28.45	29.88	**31.3**	30.71	31.13	30.37

Take Home Message

- The current methodstakes the task objective into account during training and uses the ground-truth output to aid the critic in its prediction of intermediate targets for the actor.

Take Home Message

- The current methodstakes the task objective into account during training and uses the ground-truth output to aid the critic in its prediction of intermediate targets for the actor.
- Led some (the author claimed 'significant', which the presenter suspects) improvement on both a synthetic task and a machine translation benchmark

Take Home Message

- The current methodstakes the task objective into account during training and uses the ground-truth output to aid the critic in its prediction of intermediate targets for the actor.
- Led some (the author claimed 'significant', which the presenter suspects) improvement on both a synthetic task and a machine translation benchmark
- "Actor-critic fits the training data much faster, although in some of our experiments we were able to significantly reduce the gap in the training speed and achieve a better test error using our critic network as the baseline for " (Read: It is faster, though in many cases generating a bit more error)

Take Home Message

- The current methodstakes the task objective into account during training and uses the ground-truth output to aid the critic in its prediction of intermediate targets for the actor.
- Led some (the author claimed 'significant', which the presenter suspects) improvement on both a synthetic task and a machine translation benchmark
- "Actor-critic fits the training data much faster, although in some of our experiments we were able to significantly reduce the gap in the training speed and achieve a better test error using our critic network as the baseline for " (Read: It is faster, though in many cases generating a bit more error)
- At least it leaves a lot of room for future studies :))

Questions?