Theoretical Conditions and Empirical Failure of Bracket Counting on Long Sequences with Linear Recurrent Networks

Nadine El-Naggar **Pranava Madhyastha** **Tillman Weyde**
City, University of London
United Kingdom
{nadine.el-naggar,pranava.madhyastha,t.e.weyde}@city.ac.uk

Abstract

Previous work has established that RNNs with an unbounded activation function have the capacity to count exactly. However, it has also been shown that RNNs are challenging to train effectively and generally do not learn exact counting behaviour. In this paper, we focus on this problem by studying the simplest possible RNN, a linear single-cell network. We conduct a theoretical analysis of linear RNNs and identify conditions for the models to exhibit *exact* counting behaviour. We provide a formal proof that these conditions are necessary and sufficient. We also conduct an empirical analysis using tasks involving a Dyck-1-like Balanced Bracket language under two different settings. We observe that linear RNNs generally do not meet the necessary and sufficient conditions for counting behaviour when trained with the standard approach. We investigate how varying the length of training sequences and utilising different target classes impacts model behaviour during training and the ability of linear RNN models to effectively approximate the indicator conditions.

1 Introduction

Recurrent Neural Networks (RNNs) have a long history of being used to solve a wide range of tasks involving sequential data. They were the most common choice for natural language processing, but have since been largely replaced by transformers in recent years. However, there has been a recent resurgence of interest in the theoretical aspects of RNNs, as seen in studies such as Merrill et al. (2020). Another study found that RNNs with squashing and non-squashing (i.e. unbounded) activation functions exhibit qualitative differences in their counting abilities (Weiss et al., 2018). This, along with the findings of El-Naggar et al. (2022), suggests that even RNNs with unbounded activation functions struggle to learn accurate counting on very long sequences. It is therefore crucial to understand why RNNs, despite having the capacity, often fail to accurately count in practice.

In this study, we examine the behaviour of the simplest form of RNNs: a linear single-cell RNN. Our goals are to: a) theoretically identify the necessary conditions for a linear RNN to have the ability to count, and b) explore how these conditions relate to the empirical behaviour of trained linear RNN models. The primary contributions of this paper are: a) we identify two conditions that indicate counting behaviour in linear RNNs; b) we prove that these indicator conditions are necessary and sufficient for exact counting behaviour to be achieved in linear RNNs; c) we then show empirically that linear RNNs generally do not learn exact counting and do not meet the indicator conditions; and finally, d) we show empirical relationships between the length of the training sequences and the indicator value distributions.

2 Related Work

The success of deep learning sparked a renewed interest in research into the understanding of the theoretical properties of neural networks. It has been known for long that RNNs are Turing-complete (Siegelmann and Sontag, 1992). However, Weiss et al. (2018) showed that there are different classes of RNN architectures with respect to counting capacity when using finite precision states. The relationship between RNNs and automata and formal languages has been investigated by Merrill (2019) and a formal hierarchy of counter machines has been developed by Merrill et al. (2020). This analysis is often based on “saturating” the network, i.e. replacing sigmoids with step functions, so that neural activations become binary, allowing for simpler analysis.

In practice however, activations in neural networks use a wide variety of values and systematic behaviour like counting or even parsing is often not observed, which has been discussed for over 30 years since Fodor and Pylyshyn (1988). More recently, Lake and Baroni (2017) identified a specific lack of systematic behaviour and devised SCAN, a synthetic language processing task that standard RNNs fail on. A traditional approach for processing the hierarchical structure of language is to use a stack and a number of neural stack versions have been introduced, such as those by Joulin and Mikolov (2015), Grefenstette et al. (2015) and Chen et al. (2020), which performed well on SCAN. Suzgun et al. (2019b) use stack-augmented RNNs to learn Dyck-2 languages. Mali et al. (2021) developed methods to achieve more stable behaviour of neural stacks, achieving good, but not perfect, performance on longer sequences up to 160 tokens.

The generalisation of formal language tasks to very long sequences is not often addressed, as it requires an exact or near exact behaviour of the neural network in order avoid accumulation of errors, e.g. when counting. Suzgun et al. (2019a) claims that LSTMs can learn to count, but did not test for sequences of length greater than 100. Weiss et al. (2018) reported that ReLU RNNs, which were generally hard to train, and even LSTMs which are easier to train on counting tasks, did fail for sequences of several hundred tokens. Similarly, in our previous work (El-Naggar et al., 2022) we show that almost all ReLU RNNs and LSTMs fail on sequences of length 1000.

These studies generally indicate that RNNs do not reach a configuration that enables exact counting. It is not clear what the general conditions are for an RNN to perform exact counting, which is necessary for developing a deeper understanding of the behaviour of RNNs. We start to address this question here by studying the case of a linear single-cell RNN.

3 Counting Behaviour in Linear RNNs

In this section we formally define the Balanced Bracket Language, Balanced Bracket Counter and Linear Recurrent Network and we identify conditions for the network weights that indicate that the Linear Recurrent Network will behave as a Balanced Bracket Counter. We base our counter definitions on the General Counter Machine (GCM) as defined by Merrill (2020), which we also listed for convenience in Appendix A.

The GCM is defined by a vocabulary Σ, finite set of states Q, initial state q_0, counter update function u, state update function δ, and acceptance mask F. Some components, such as states, can be empty. The counter computation also uses a zero check function. An input string x is processed by the counter one token x_t at a time. The counter update function u is used to update the counter value $\mathbf{c}$ with integer increments $(+m)$. The counter updates are dependent on the current input token, the current state, and a finite mask of the current state. In our setting, a counter machine is said to accept a sequence if $\mathbf{c} = 0$ after the whole sequence is has been processed. A counter machine M is said to accept a language L if Maccepts $s \iff s \in L$.

We focus on sequences consisting of one type of bracket: Σ ={'(', ')'}.

Definition 1. (Balanced Bracket Language BB) The Balanced Bracket Language BB is defined as

$$BB = \{s \in \Sigma^* |\, \text{count}(\text{ `('}, s) = \text{count}(\text{ `)'}, s)\}.$$

The order of the opening and closing brackets does not matter for the BB language, only that the number of opening and closing brackets in a sequence is equal overall. Dyck-1 sequences are a special case of BB sequences where the number of closing brackets is never greater than the number of opening brackets at any point in the sequence, i.e. for all prefixes.

Our focus is on the counting abilities of single-cell linear RNNs. These networks do not have the capacity to accept Dyck-1 sequences from the universe of all possible sequences, because they would need to treat negative counts differently from positive activations to distinguish correctly ordered from incorrectly ordered bracket sequences. However, that is not possible with a single linear neuron, and additional mechanisms would be needed to fully accept a Dyck-1 language from the entire universe of possible sequences.

Previous work, such as Suzgun et al. (2019a), who train their single-cell RNN models to learn Dyck-1 languages only use valid Dyck-1 sequences in their datasets, where there are never any excess closing brackets at any point in the sequences. This seems unnecessarily limiting, however. Therefore, we use the BB language which can be fully accepted using a single-cell linear RNN.

Definition 2. (Balanced Bracket Counter) A General Counter Machine is a Balanced Bracket Counter iff it accepts BB.

Definition 3. (Linear Recurrent Network) A Linear Recurrent Network (LRN) is a network which receives an input x_t at every timestep t, which is used along with the activation from the previous timestep h_{t-1} and weights W, U, and W_b to produce activation h_t, which is then passed on to the next timestep with the update function:

$$h_t = Wx_t + Uh_{t-1} + W_b.$$

Here, x_t is a one-hot-encoded input token, an LRN is similar to a stateless counter machine if we apply a zero check z function to h_t after processing the last input. A counter based on the LRN deviates from the definition by Merrill (2020) in that:

- The counter value **c** corresponds to h_t (it is the only value that propagates from one time step to the next), which is real instead of integer,
- The results of the update function ($+m$ in the Counter Machine) are real numbers, specifically:

  $a = Wx_t + W_b$ if $x_t =$ '(' , and
  $b = Wx_t + W_b$ if $x_t =$ ')' .

We use a single-cell LRN for bracket counting. As a result, W is a vector of the same dimensionality as x_t and U and W_b are scalars, as well as m, **c**, and h_t.

In Theorem 1, we relate Balanced Bracket Counter behaviour of a LRN to specific conditions on its *weights*. We define two indicator conditions and show that they are necessary and sufficient for exact counting behaviour to be achieved in a LRN.

Theorem 1. (Linear RNN Counting Indicators) The following two indicator conditions are necessary and sufficient for a Linear Recurrent Network to accept the Balanced Bracket Language BB.

1. $\frac{a}{b} = -1$ (AB ratio)
2. $U = 1$ (recurrent weight).

Proof 1. We prove that the counting indicator conditions in Theorem 1 are necessary and sufficient to accept the Balanced Bracket Language with a Linear Recurrent Network. We first prove that the conditions are necessary (Part 1) and then that they are sufficient (Part 2).

Part 1: We prove that the counting indicator conditions in Theorem 1 are satisfied if a Linear Recurrent Network accepts the Balanced Bracket Language by using different input sequences (Table 1), from which we derive the indicator conditions. If a Linear Recurrent Network accepts the Balanced Bracket Language, then equal numbers of opening and closing brackets result in an output activation $h_t = 0$, otherwise, $h_t \neq 0$. This is equivalent to zero check function $z(h_t)$ yielding 0 or 1. Therefore, we will not include the zero-check function in the following derivation.

Case	Input	Output (h)	Findings
1	'('	$h_1 \neq 0$	$a \neq 0$
2	')'	$h_1 \neq 0$	$b \neq 0$
3	'()'	$h_2 = 0$	$b = -Ua$ and $U \neq 0$
4	'(('	$h_2 \neq 0$	$U \neq -1$
5	'(())'	$h_4 = 0$	$b + Ub + U^2a + U^3a = 0$
6	'()()'	$h_4 = 0$	$b + Ua + U^2b + U^3a = 0$

Table 1: Input sequences used to derive the indicator conditions from Theorem 1.

Case 1: $seq =$'(', $h_0 = 0, h_1 \neq 0$
$h_1 = a + Uh_0 = a$
$\therefore a \neq 0$

Case 2: $seq =$')', $h_0 = 0, h_1 \neq 0$
$h_1 = b + Uh_0 = b$
$\therefore b \neq 0$

Case 3: $seq =$ '()', $h_2 = 0$
$h_2 = b + Ua$
$b + Ua = 0$
From cases 1,2: $a \neq 0$ and $b \neq 0$
$\therefore b = -Ua$, and $U \neq 0$

Case 4: $seq =$'((', $h_2 \neq 0$
$h_2 = a + Ua$
$a + Ua \neq 0$
$\therefore U \neq -1$

Case 5: $seq =$'(())', $h_4 = 0$
$h_3 = b + Uh_2 = b + U(a + Ua)$
$h_4 = b + Uh_3 = b + U(b + U(a + Ua))$
$\therefore h_4 = b + Ub + U^2a + U^3a = 0$

Case 6: $seq =$'()()', $h_4 = 0$
$h_3 = a + Uh_2 = a + U(b + Ua)$
$h_4 = b + Uh_3 = b + U(a + U(b + Ua))$
$\therefore h_4 = b + Ua + U^2b + U^3a = 0$

Combine the findings from cases 5 and 6.
$b+Ub+U^2a+U^3a = b+Ua+U^2b+U^3a$

Subtract $b + U^3a$ from both sides and divide both sides by U
$b + Ua = a + Ub$

Rearrange and factorise
$b - a = U(b - a)$

As a result, we get 2 possible situations:

(a) $U = 1$, which implies $a = -b$ by case 3

(b) $a = b$, where by cases 1 and 2 we know $a \neq 0$ and $b \neq 0$, and $U = -1$ follows from case 3, which contradicts case 4

$\therefore$ $U = 1$ and $a = -b$, i.e., the counter indicator conditions listed in Theorem 1 hold, if a Linear Recurrent Network accepts the Balanced Bracket Language.

Part 2: We prove by induction that if the counting indicator conditions listed in Theorem 1 hold, a Linear Recurrent Network accepts the Balanced Bracket Language.

Each sequence consists of n opening brackets and m closing brackets, and the input token x_k is either '(' or ')'.

Base Case: $k = 1$

- $x_1 =$'(', $n = 1$ and $m = 0$:
  $h_1 = a + Uh_0$
  $h_1 = a$
- $x_1 =$')', $n = 0$ and $m = 1$:
  $h_1 = -a + Uh_0$
  $h_1 = -a$

For n opening and m closing brackets, the following equation satisfies the base case, and is therefore our induction hypothesis:

$$h_k = (n - m) \times a$$

We assume that this is true for sequences of length k consisting of n opening brackets and m closing brackets. We prove by induction that if this holds for sequences of length k tokens, it holds for sequences of length $k + 1$ tokens. In our induction step, we use once $x_{k+1} =$'(' and once $x_{k+1} =$')'.

Induction Step:

- If $x_{k+1} =$'(':
  $h_{k+1} = ((n + 1) - m) \times a$
- If $x_{k+1} =$')':
  $h_{k+1} = (n - (m + 1)) \times a$

From the premise, we can derive that:
$h_k = a+Uh_{k-1}$ if $x_k =$'(', and $h_k = -a+Uh_{k-1}$ if $x_k =$')'

- If $x_{k+1} =$'(':
  $h_{k+1} = a + h_k$
  Substitute $h_k = (n - m) \times a$
  $h_{k+1} = a + ((n - m) \times a)$
  $\therefore h_{k+1} = ((n + 1) - m) \times a$
- If $x_{k+1} =$')':
  $h_{k+1} = -a + h_k$
  Substitute $h_k = (n - m) \times a$
  $h_{k+1} = -a + ((n - m) \times a)$
  $\therefore h_{k+1} = (n - (m + 1)) \times a$

Therefore, we prove that if the counting indicator conditions listed in Theorem 1 are satisfied in a Linear Recurrent Network, it accepts the Balanced Bracket Language. □

4 Counting in Linear RNNs in Practice

We conduct experiments to analyse the models and whether or not they satisfy the conditions defined in the previous section in training. We use 2 classification tasks to evaluate our models.

4.1 Task 1: Binary Classification

We use the same task and model as in our previous work (El-Naggar et al., 2022), i.e., a linear RNN without biases with a single output neuron with sigmoid activation to classify the bracket difference of the sequence as > 0 or ≤ 0 (binary). The absence of a trainable bias reduces the degrees of freedom in the model, and is equivalent to having a bias (W_b) value that is fixed to 0, hence simplifying the learning task. We also use the same models with trainable biases. The models are trained with sequences of lengths 2, 4 and 8 tokens for 100 epochs in 10 runs. The initial count value (h_0) has a value of 0 for every sequence. We inspect the weights of our models and plot the distribution of the indicator values (a/b,U) of the trained models for each training set size in Figure 1. We observe that the models do not fulfill the indicator conditions, but they do approach the conditions as the length of the training sequences increases. We observe that the distributions of the a/b indicator have mean values above -1 but less so for longer training sequences.

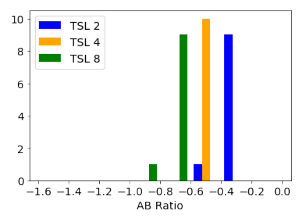
4.2 Task 2: Ternary Classification

We also apply a ternary classification: > 0, $= 0$ or < 0. We use the same model as in Task 1, except that instead of a single output neuron with a sigmoid output activation, we use 3 output neurons and a softmax output layer with bias, which is the minimal configuration that can achieve this task. We also use the same models with trainable biases.

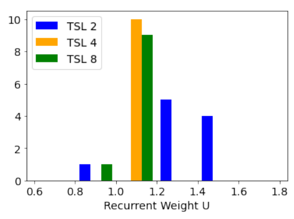
The initial count value (h_0) has a value of 0 for every sequence and the models are trained in the same manner as the models from Task 1. The ternary classification accuracy is slightly lower as

Classification Experiment	Train Length	Train Avg(Min/Max)	20 Tokens Avg(Min/Max)	50 Tokens Avg(Min/Max)
Binary (without bias)	2	100 (100/100)	69.2 (6.04/77.3)	69.0 (66.7/72.7)
Binary (without bias)	4	100 (100/100)	94.8 (94.7/95.3)	89.3 (88.7/90.0)
Binary (without bias)	8	100 (100/100)	96.9 (94.0/100)	92.7 (78.7/98.0)
Ternary (without bias)	2	90 (33.3/100)	55.6 (33.3/64.4)	51.4 (33.3/60.0)
Ternary (without bias)	4	100 (100/100)	79.5 (65.8/94.7)	67.2 (66.7/68.0)
Ternary (without bias)	8	100 (100/100)	94.4 (67.1/100)	85.7 (66.7/100)
Binary (with bias)	2	100 (100/100)	73.4 (63.3/100)	72.4 (60.0/93.3)
Binary (with bias)	4	100 (100/100)	95.3 (92.7/98.0)	86.0 (77.3/90.7)
Binary (with bias)	8	100 (100/100)	95.2 (85.3/100)	87.9 (70.0/98.0)
Ternary (with bias)	2	88.3 (66.7/100)	58.0 (38.2/67.6)	54.4 (43.6/67.5)
Ternary (with bias)	4	97.9 (79.2/100)	81.5 (64.4/100)	68.0 (65.3/73.3)
Ternary (with bias)	8	100 (100/100)	95.9 (83.6/100)	76.5 (65.3/100)

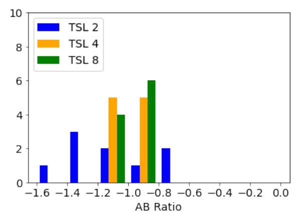
Table 2: Accuracy metrics of our previous binary classification experiments without bias (El-Naggar et al., 2022), ternary classification experiments without bias, and binary and ternary classification experiments with bias.



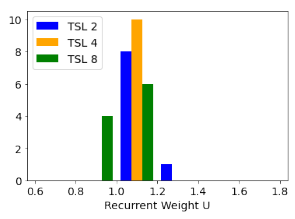
(a) AB ratio binary (no bias)



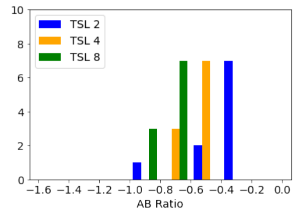
(b) U value binary (no bias)



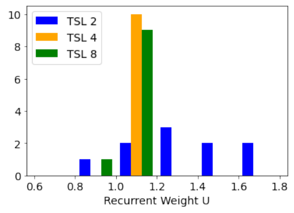
(c) AB ratio ternary (no bias)



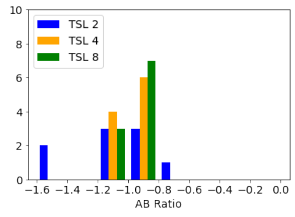
(d) U value ternary (no bias)



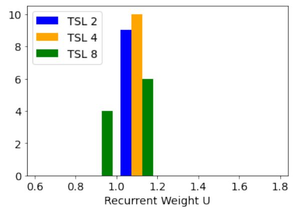
(e) AB ratio binary (bias)



(f) U value binary (bias)



(g) AB ratio ternary (bias)



(h) U value ternary (bias)

Figure 1: Indicators after training on binary and ternary classification without biases (top) and with biases (bottom) with different Training Sequence Lengths (TSL).

can be expected with more classes. For shorter training sequences, this may be related to the larger number of model parameters relative to the data points. The accuracy improves with longer training sequences. Ternary classification does not show a mean of a/b above -1 as can be seen in Figure 1.

5 Conclusions and Future Work

Although linear RNNs have the theoretical capacity for counting behaviour, previous research has shown that these models often struggle to effectively generalise counting behaviour to long sequences. In this study, we present a set of necessary and sufficient conditions that indicate counting behaviour in linear RNNs and provide proof that meeting these conditions is equivalent to counting behaviour. To investigate the extent to which these conditions are met, we examine the parameters of models trained on sequences of varying lengths for classification tasks. We use both binary and ternary classification tasks and find that the models do not fully meet these conditions, but do approach them and get closer as the sequence length increases.

There are several potential areas for future work based on these findings. One possible research direction is to extend this approach to ReLU RNNs, and LSTMs as far as possible. Another option is to devise methods to ensure that the indicator conditions we have identified are met during training in order to improve the generalisation abilities of our models.

References

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. *arXiv preprint arXiv:2008.06662*.

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. 2022. Experiments in learning dyck-1 languages with recurrent neural networks. In *Proceedings of the 3rd Human-Like Computing Workshop (HLC 2022) co-located with the 2nd International Joint Conference on Learning and Reasoning (IJCLR 2022), Windsor, United Kingdom, September 28-30th, 2022*, volume 3227 of *CEUR Workshop Proceedings*, pages 24–28. CEUR-WS.org.

Nadine El-Naggar, Pranava Madhyastha, and Tillman Weyde. 2022. Exploring the long-term generalization of counting behavior in rnns. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*.

Jerry A Fodor and Zenon W Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, pages 1828–1836.

Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28.

Brenden M Lake and Marco Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*.

Ankur Mali, Alexander Ororbia, Daniel Kifer, and Lee Giles. 2021. Investigating backpropagation alternatives when learning to dynamically count with recurrent neural networks. In *International Conference on Grammatical Inference*, pages 154–175. PMLR.

William Merrill. 2019. Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13.

William Merrill. 2020. On the linguistic capacity of real-time counter automata. *CoRR*, abs/2004.06866.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A Smith, and Eran Yahav. 2020. A formal hierarchy of rnn architectures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459.

Hava T. Siegelmann and Eduardo D. Sontag. 1992. On the computational power of neural nets. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 440–449, New York, NY, USA. Association for Computing Machinery.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. 2019a. Lstm networks can perform dynamic counting. *arXiv preprint arXiv:1906.03648*.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. 2019b. Memory-augmented recurrent neural networks can learn generalized dyck languages. *arXiv preprint arXiv:1911.03329*.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 740–745. Association for Computational Linguistics.

A General Counter Machine

This definition is from Merrill (2020).

Definition 4. (General Counter Machine) A k-Counter is a tuple $\langle \Sigma, Q, q_0, u, \delta, F \rangle$ with:

1. A finite alphabet Σ
2. A finite set of states Q
3. An initial state q_0
4. A counter update function

$$u : \Sigma \times Q \times \{0,1\}^k \to (\{+m : m \in \mathbb{Z}\} \cup \{\times 0\})^k$$

5. A state transition function

$$\delta : \Sigma \times Q \times \{0,1\}^k \to Q$$

6. An acceptance mask

$$F \subseteq Q \times \{0,1\}$$

The counter machine computation is formalised in Definition 5. The finite mask of the current state is created using a zero-check function $z(\mathbf{v})$ for a vector v, where:

$$z(\mathbf{v})_i = \begin{cases} 0, & \text{if } v_i = 0 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

Definition 5. (Counter Machine Computation) Let $\langle q, \mathbf{c} \rangle \in Q \times \mathbb{Z}^k$ be a configuration of machine M. Upon reading input $x_t \in \Sigma$, we define the transition

$$\langle q, \mathbf{c} \rangle \to_{x_t} \langle \delta(x_t, q, z(\mathbf{c})), u(x_t, q, z(\mathbf{c}))(\mathbf{c}) \rangle$$