RING:
Regular expressions INference and Generation

TAUDoS, June 16th, 2022

Maxime Raynal (LIG/MRIM & Nokia Bell Labs)
Marc-Olivier Buob (Nokia Bell Labs)
Georges Quénot (LIG/MRIM)

Agenda

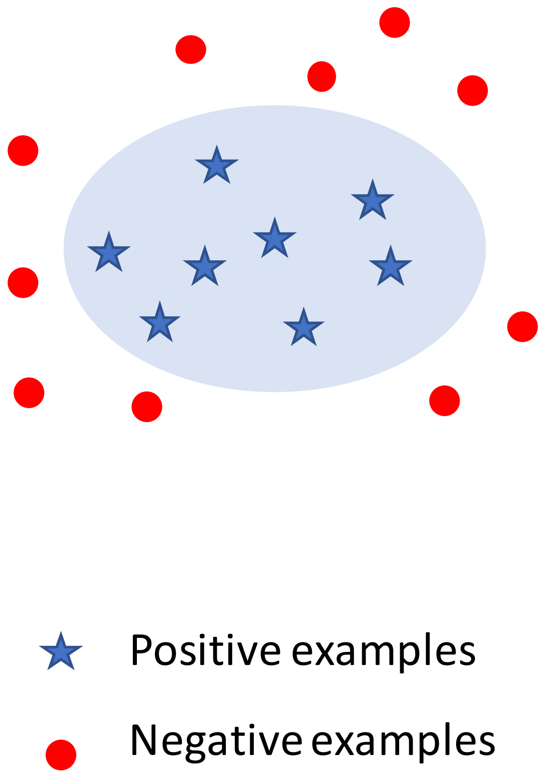
1. Introduction & state of the art
2. RING
   2.1 Overview
   2.2 Loss function
   2.3 Dataset generation
   2.4 Model architecture
3. Future works & conclusion

Part 1: introduction & state of the art

Introduction

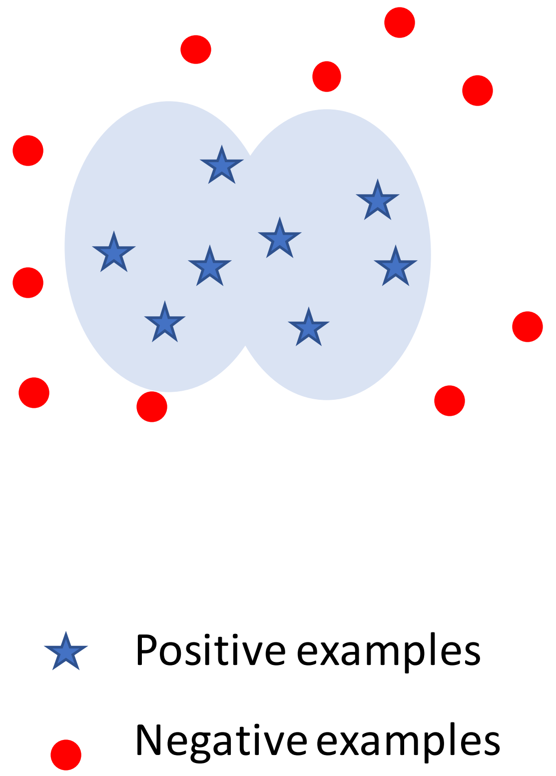
- Grammar inference problem:
  - Input: set of positive and negative examples
  - Output: inferred language

- Problem:
  - In general, an infinity of solutions
  - Two trivial solutions:
    - The PTA (prefix tree acceptor) of positive examples
    - The PTA's complement of negative examples



Introduction

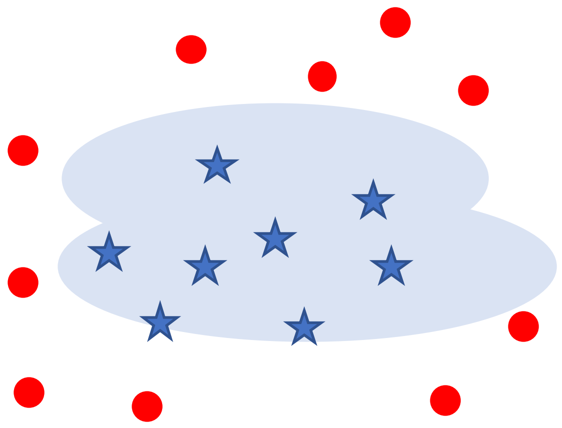
- Grammar inference problem:
  - Input: set of positive and negative examples
  - Output: inferred language
- Problem:
  - In general, an infinity of solutions
  - Two trivial solutions:
    - The PTA (prefix tree acceptor) of positive examples
    - The PTA's complement of negative examples

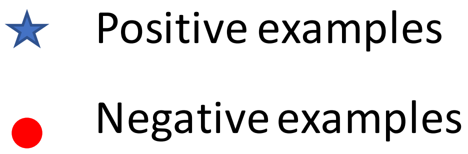


Introduction

- Grammar inference problem:
  - Input: set of positive and negative examples
  - Output: inferred language

- Problem:
  - In general, an infinity of solutions
  - Two trivial solutions:
    - The PTA (prefix tree acceptor) of positive examples
    - The PTA's complement of negative examples





Introduction

- Our goal:
  - Model and train one RNN to infer a regular expression from positive examples
  - Without negative examples

- Our use case: log parsing

```
===========================
Slot/  Provisioned   Admin
Id     Type          State
---------------------------
1      imm-2pac-fp3  up
1/2    p6-10g-sfp    up
2      imm-2pac-fp3  down
2/1    p6-10g-sfp    up
===========================
```

Slot / Id	Provisioned type	Admin state
1	imm-2pac-fp3	up
1/2	p6-10g-sfp	up
2	imm-2pac-fp3	down
2/1	p6-10g-sfp	up

Challenges

- No negative examples
- Trade-off between accuracy and shortness
- Efficiency

Σ = {a,b}

ab
abb
aabb
bbb
bbbb
aaabbb
b

Positive examples

PTA **ab|abb|aabb|bbb|bbbb|aaabbb|b**

(a|aa|aaa)?(b|bb|bbb|bbbb)

(a+b+) | b

a*b+

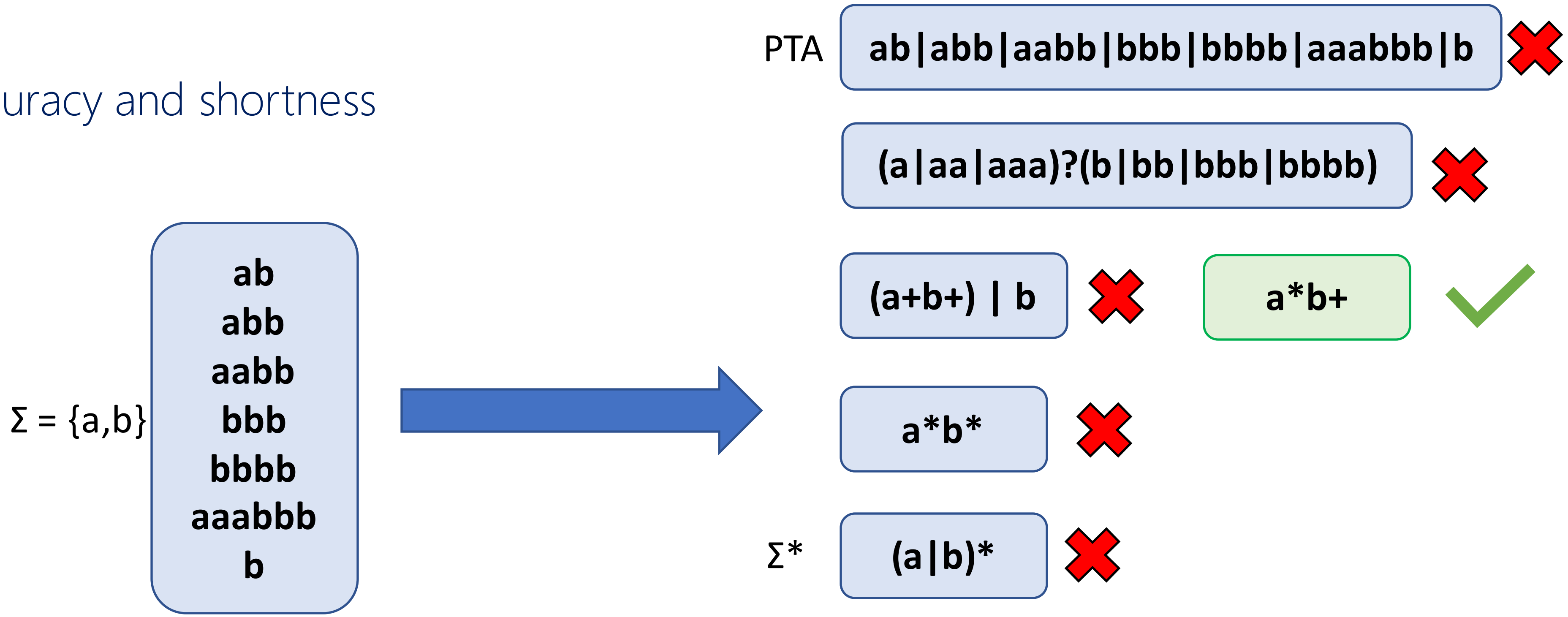
a*b*

Σ* **(a|b)***

Possible solutions (regular expressions)

Challenges

- No negative examples
- Trade-off between accuracy and shortness
- Efficiency



Positive examples

Possible solutions (regular expressions)

State of the art: Gold

- Gold defined a theoretical framework to regular language induction (identification in the limit)
- Gold presented an algorithm to induce an automaton from examples

- Problems:
  - Without negative examples, Gold returns results that are not interesting in practice
  - Works with automata, not with regular expressions

State of the art: inducing an automaton with a RNN

- Inducing a DFA with a RNN has been explored in the 1990s

- Idea:
  - Train a RNN to act as an automaton
  - Use this RNN to extract an automaton

- Problems:
  - Need to train a new RNN for each language we want to induce
  - Unadapted to solve our problem

State of the art: machine learning methods

- Several methods in the literature, mostly using a metaheuristic approach
- Several working directly with regular expressions
- Problems:
  - Do not scale
  - Usually require negative examples

Part 2: RING

2.1 Overview
2.2 Loss function
2.3 Dataset generation
2.4 Model architecture

RING overview

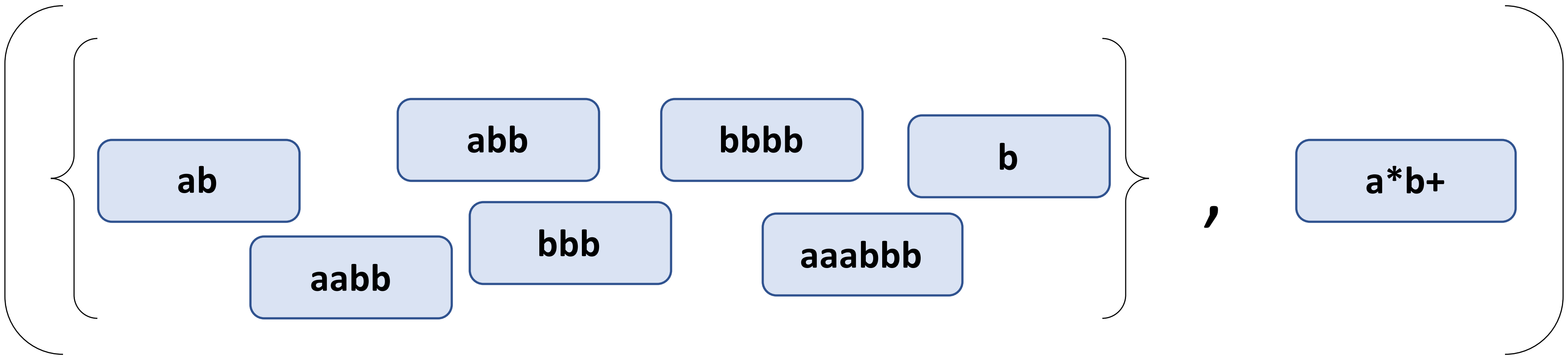
- Novelties
  - "One to infer them all"
  - No negative examples

- Controlled and automated data generation
  - Uncommon in the DL field -> explainability
  - Automated result evaluation

- Our model architecture uses recent DL methods

Training samples generation

- We can easily generate training samples
  - Sample = regular expression and a set of positive examples



Training samples generation

- We can easily generate training samples
  - Sample = regular expression and a set of positive examples

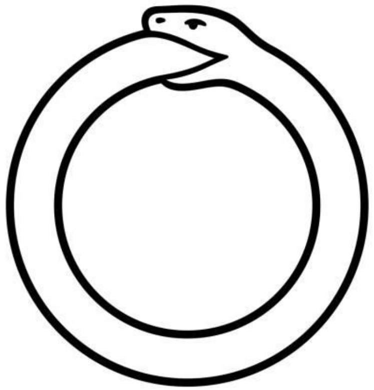
- Advantages:
  - Automated generation
  - Controlled process
  - Allows to test different training strategies
  - Gives better insight on hyperparameters influence

Training samples generation

- We can easily generate training samples
  - Sample = regular expression and a set of positive examples

- Advantages:
  - Automated generation
  - Controlled process
  - Allows to test different training strategies
  - Gives better insight on hyperparameters influence

- Training samples generation strategy:
  - ~~Generate a regular expression from examples~~
  - Generate examples from a regular expression



Generating random regular expressions

- Challenges
  - We would like to pick a random regular expression "uniformly" but ...
    - Infinite space to pick in
    - Regular expression are ambiguous by design

Generating random regular expressions

- Challenges
  - We would like to pick a random regular expression "uniformly" but ...
    - Infinite space to pick in
    - Regular expression are ambiguous by design
- Possible approaches:
  - Random DFA generation: DFA-to-RE conversions (Arden, state suppression) lead to long regular expressions

Generating random regular expressions

- Challenges
  - We would like to pick a random regular expression "uniformly" but ...
    - Infinite space to pick in
    - Regular expression are ambiguous by design
- Possible approaches:
  - Random DFA generation: DFA-to-RE conversions (Arden, state suppression) lead to long regular expressions
  - Random AST generation with given size: difficult to generate without bias

Generating random regular expressions

- Challenges
  - We would like to pick a random regular expression "uniformly" but ...
    - Infinite space to pick in
    - Regular expression are ambiguous by design
- Possible approaches:
  - Random DFA generation: DFA-to-RE conversions (Arden, state suppression) lead to long regular expressions
  - Random AST generation with given size: difficult to generate without bias
  - Solution:
    - Generate a random string and reject it if it is not a valid regular expression
    - Enable regular expressions simplifications (e.g., a** becomes a*)

Generating positive examples from a regular expression

- Challenges
  - Difficult to pick positive examples smaller than given size uniformly from a regular expression
  - Due to intrinsic ambiguities in regular expressions

- Possible approaches:
  - Random walk on a DFA or on an AST: biased
  - Solution: Combinatorial generation of positive examples (~enumeration)

Quality function

- Challenges
  - Multiple feasible solutions
  - No unique, canonical way to define what is the best regular expression
  - Several criteria

- Criteria:
  - A solution must recognize all examples (feasibility)
  - A solution must be short (shortness)
  - A solution must be specific (accuracy)

Quality function

- Shortness:
  - Number of nodes in the corresponding AST

- Accuracy:
  - *Density* of the language represented by the RE
  - Intuition: partition a language L by word length, L = $\{L_0, L_1, L_2, ...\}$
  - Each of these subsets is finite
  - In real use-cases, only the first ones are interesting for us

- Density formula
  - Conserves inclusion relationship
  - Characterizes how big a language is

$$\rho(r) = \sum_{n \in N*} \frac{1}{2^{n-1}} \cdot \frac{|\mathcal{L}(r)_n|}{|\Sigma|^n}$$

Quality function

- Multiple criteria
- Optimize Shortness |r| and density rho(r)

$$\text{Loss}(r) = |r|^{\alpha} \cdot \rho(r)^{\beta}$$

- Where α and β are positive hyperparameters

RING model architecture

- Seq2seq: RNNs rule the field

- *Input*: sequence of positive examples
  - Using a metacharacter to split examples
  - One-hot encoding

<SOS> a b b $ a b $ a a b <EOS>

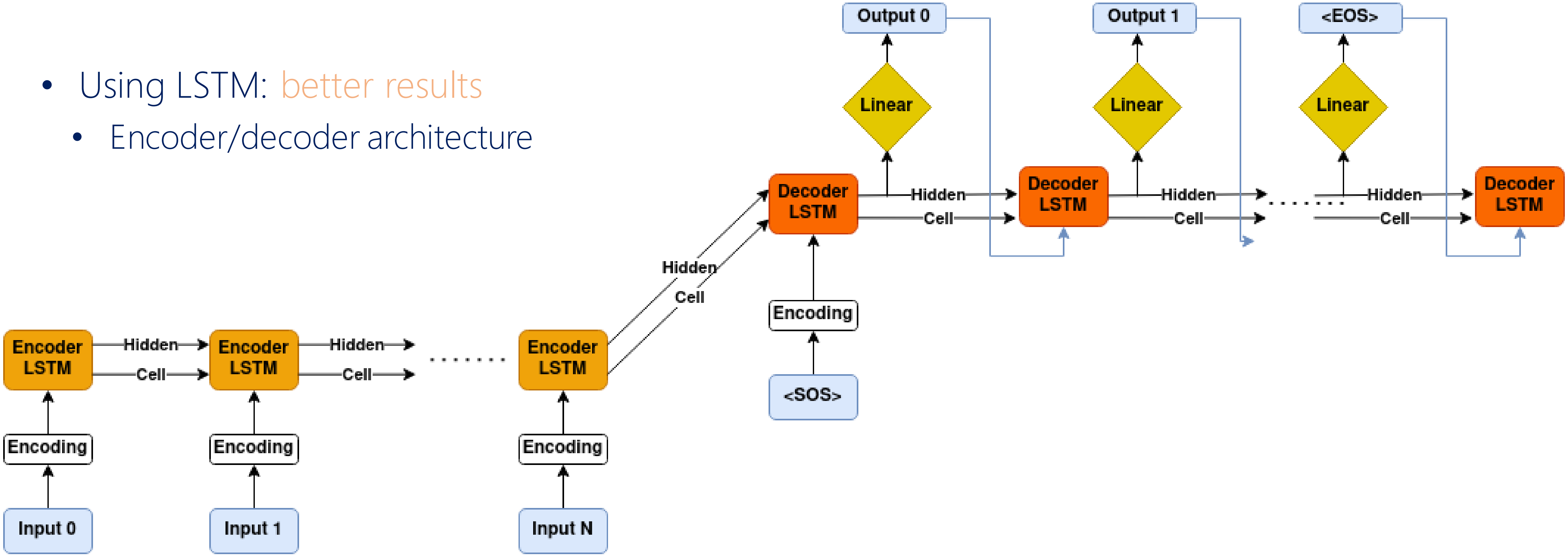
- *Output*: inferred regular expression in prefix notation
  - Usual set of operators: · + | * ?
  - One hot encoding

<SOS> · + a + b <EOS>

RING model architecture

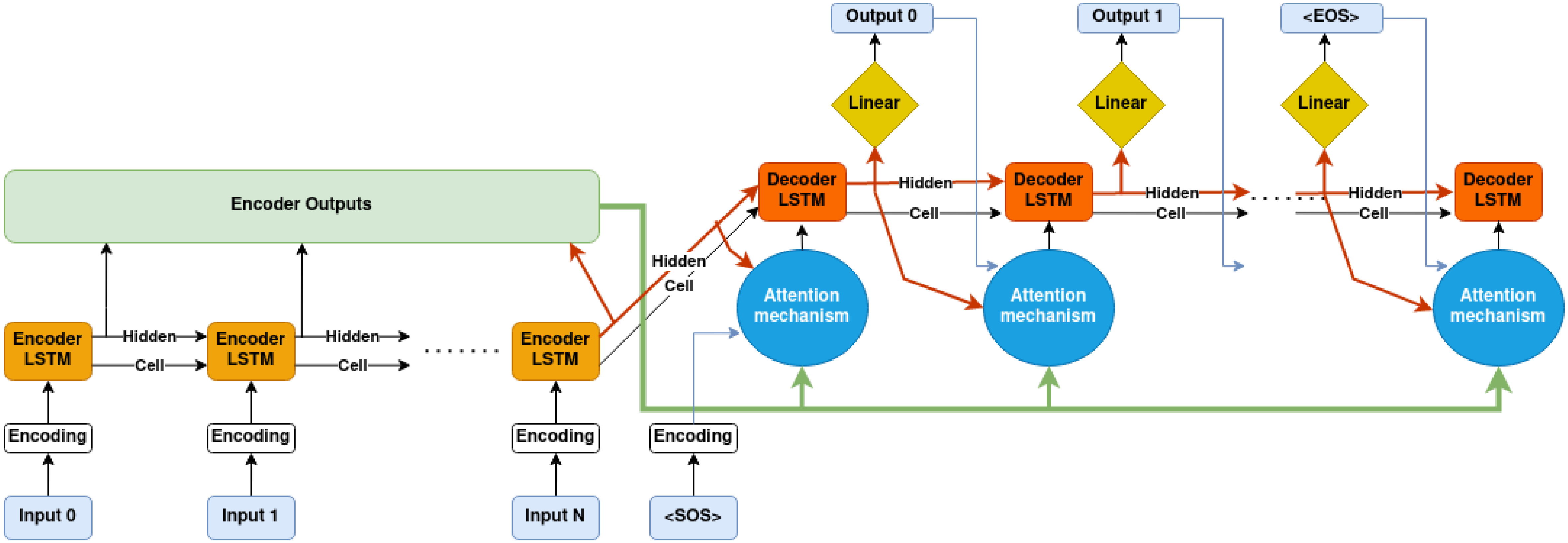
- Using simple RNNs: poor results
  - Lack of long-term memory

- Using LSTM: better results
  - Encoder/decoder architecture



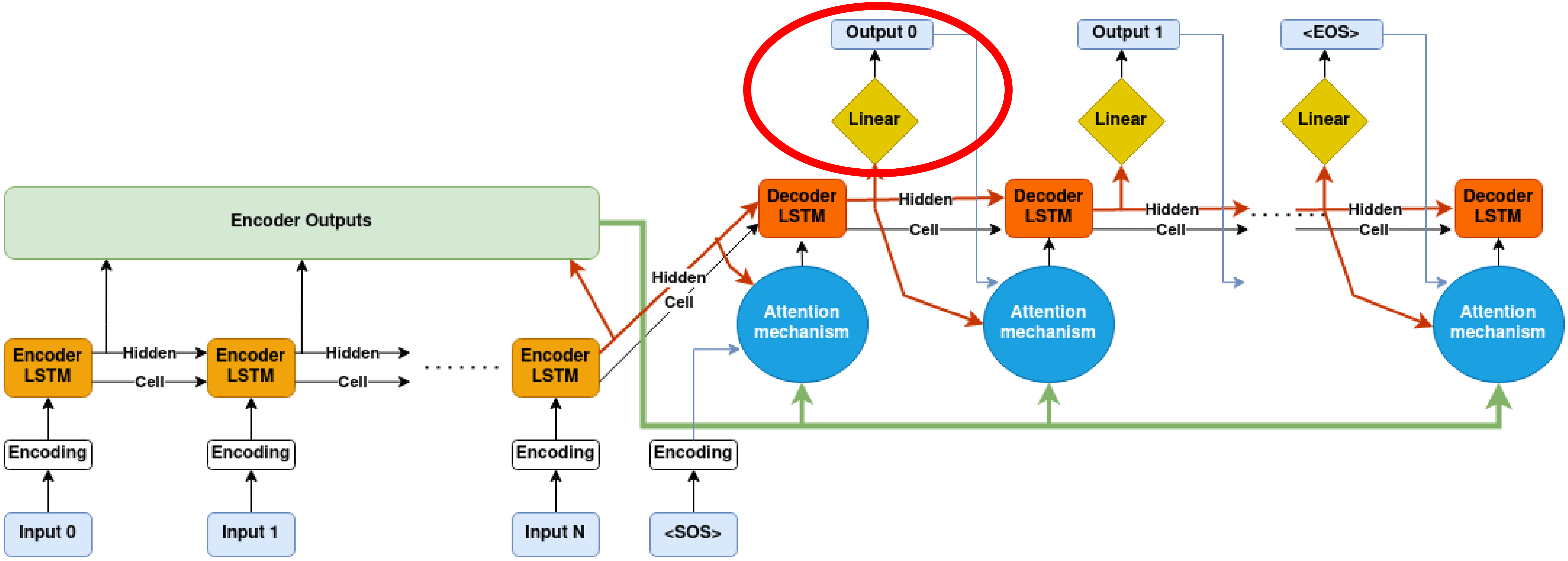
Improvement 1: attention mechanism

- Allows the decoder to use all encoders outputs to improve the output quality



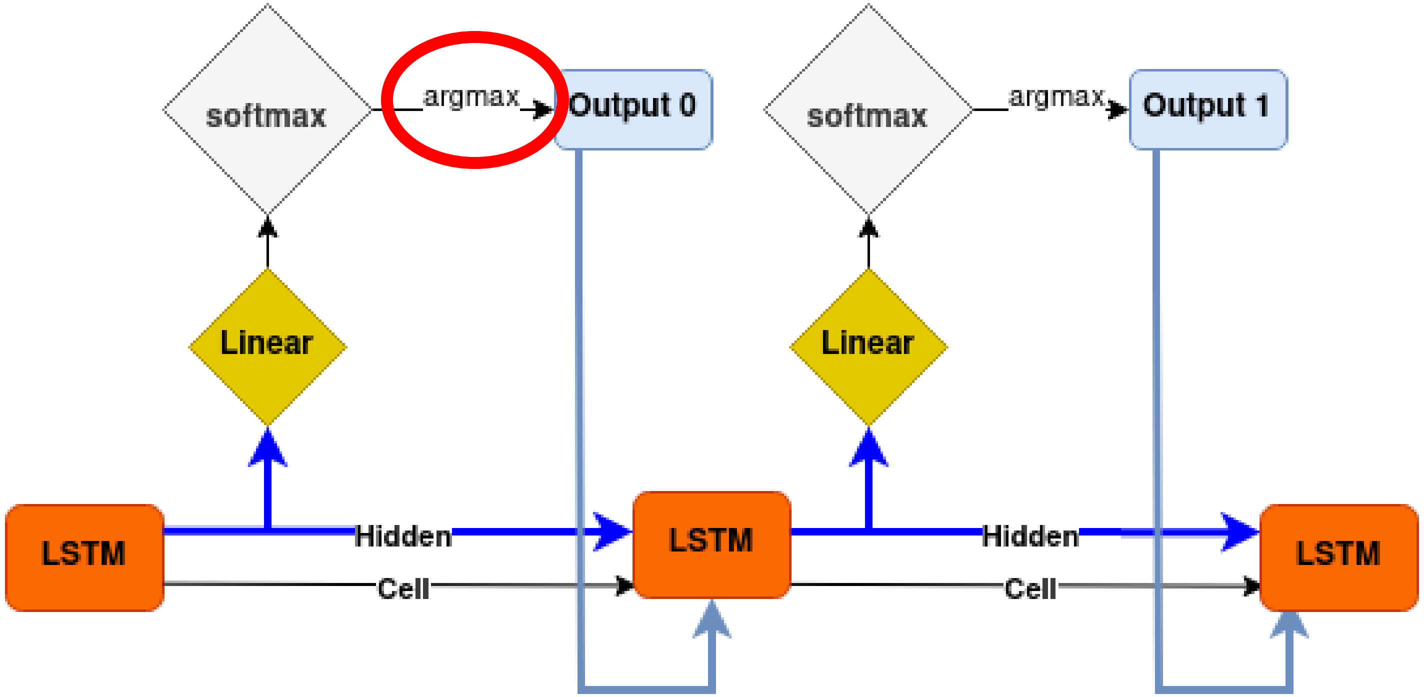
Improvement 2: beam search decoding (BSD)

- Allows to use the entire network as a function to guide a search algorithm



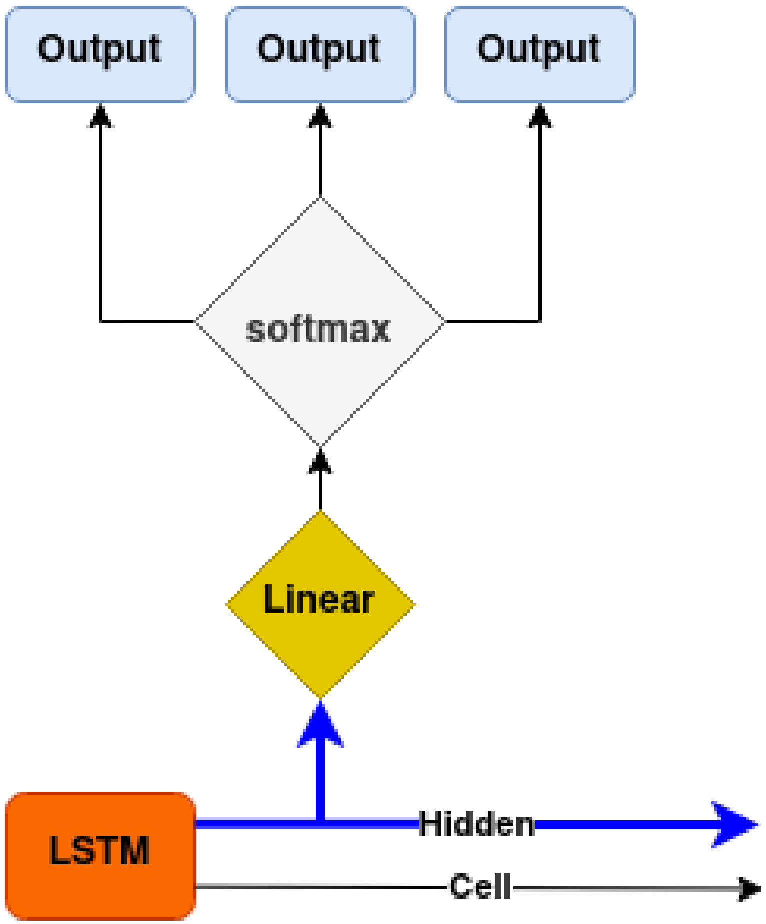
Improvement 2: beam search decoding (BSD)

- Without BSD, we fetch the best character at each step (greedy search)



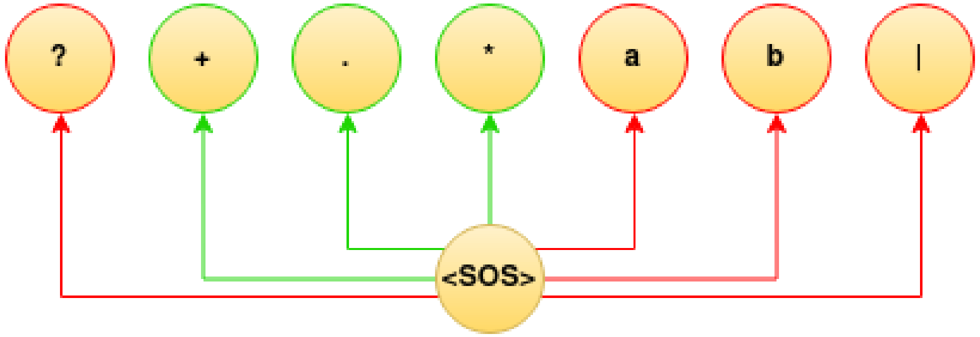
Improvement 2: beam search decoding (BSD)

- With BSD, we fetch the B best character at each step (where B is the beam width)



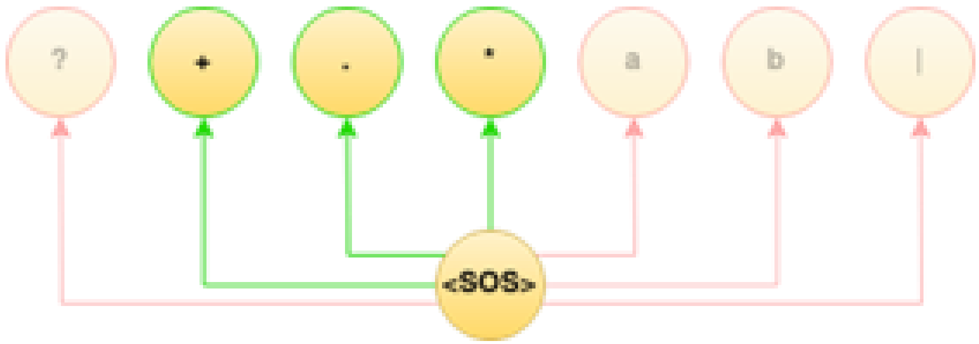
Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character



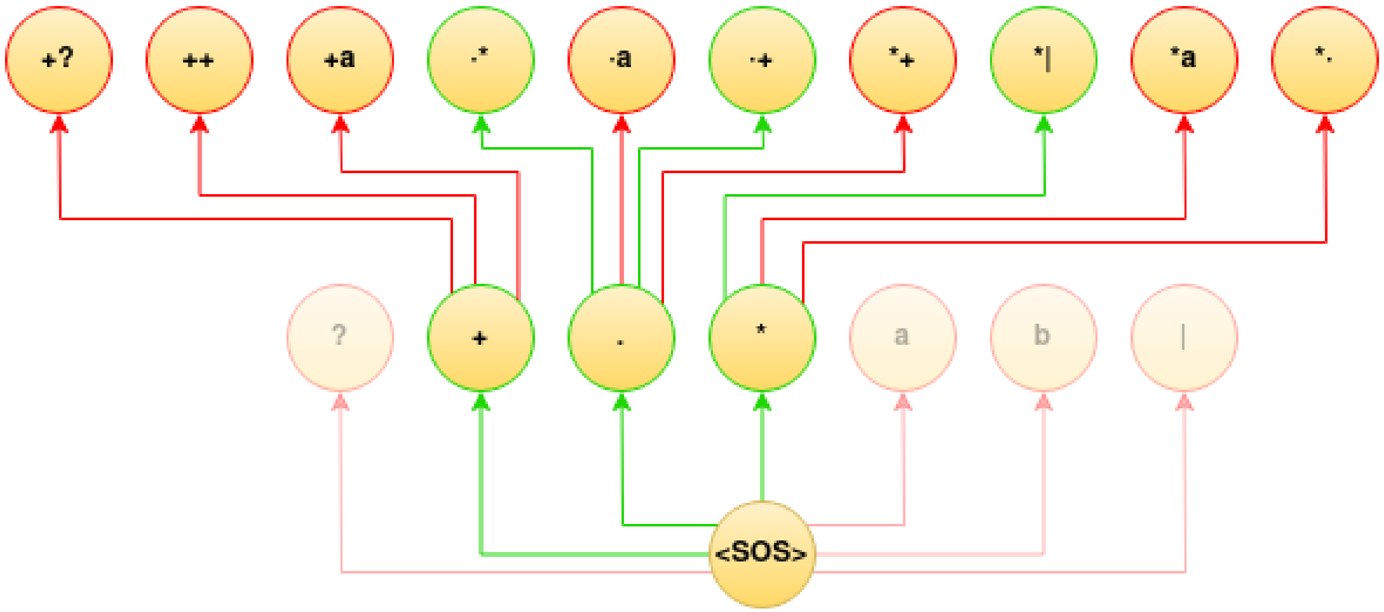
Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character



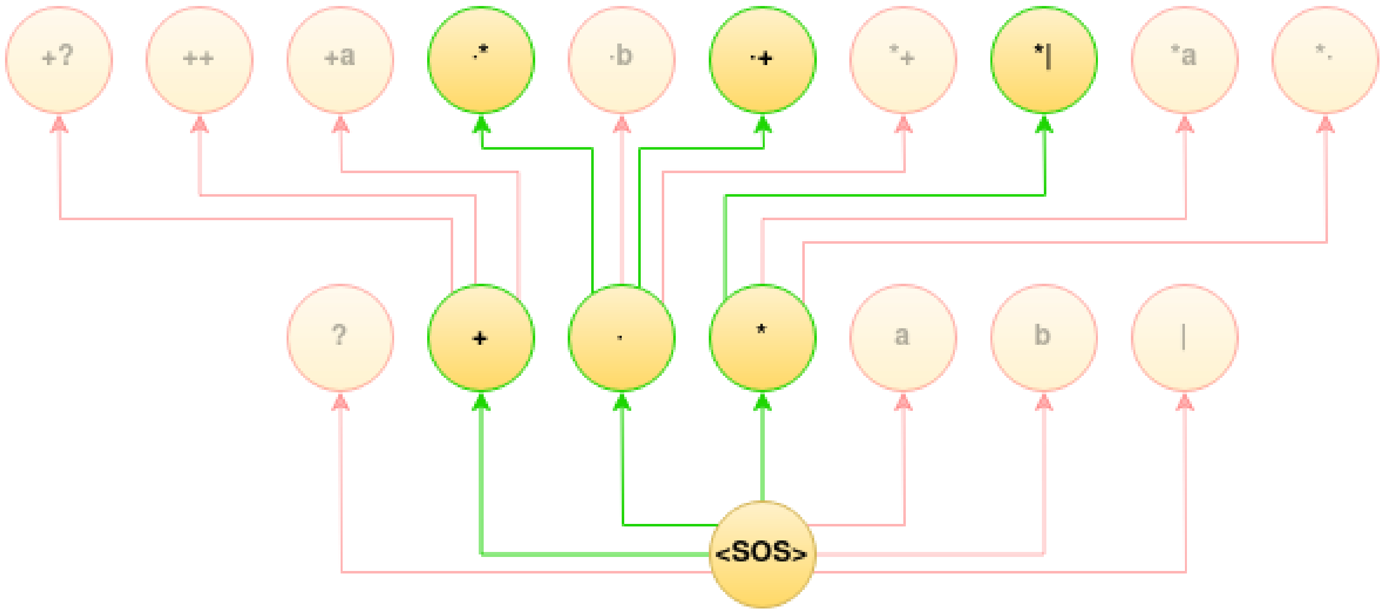
Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character



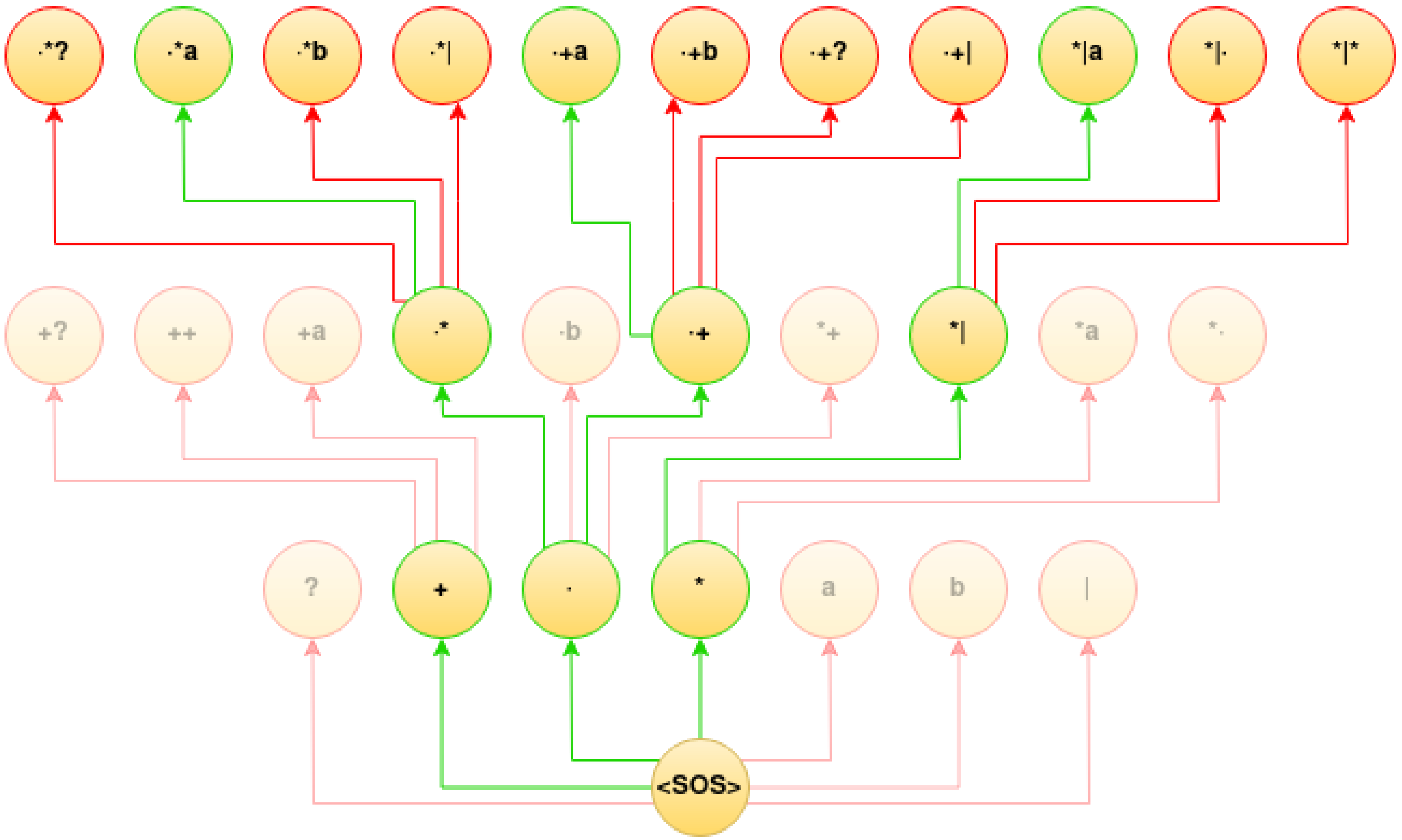
Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character



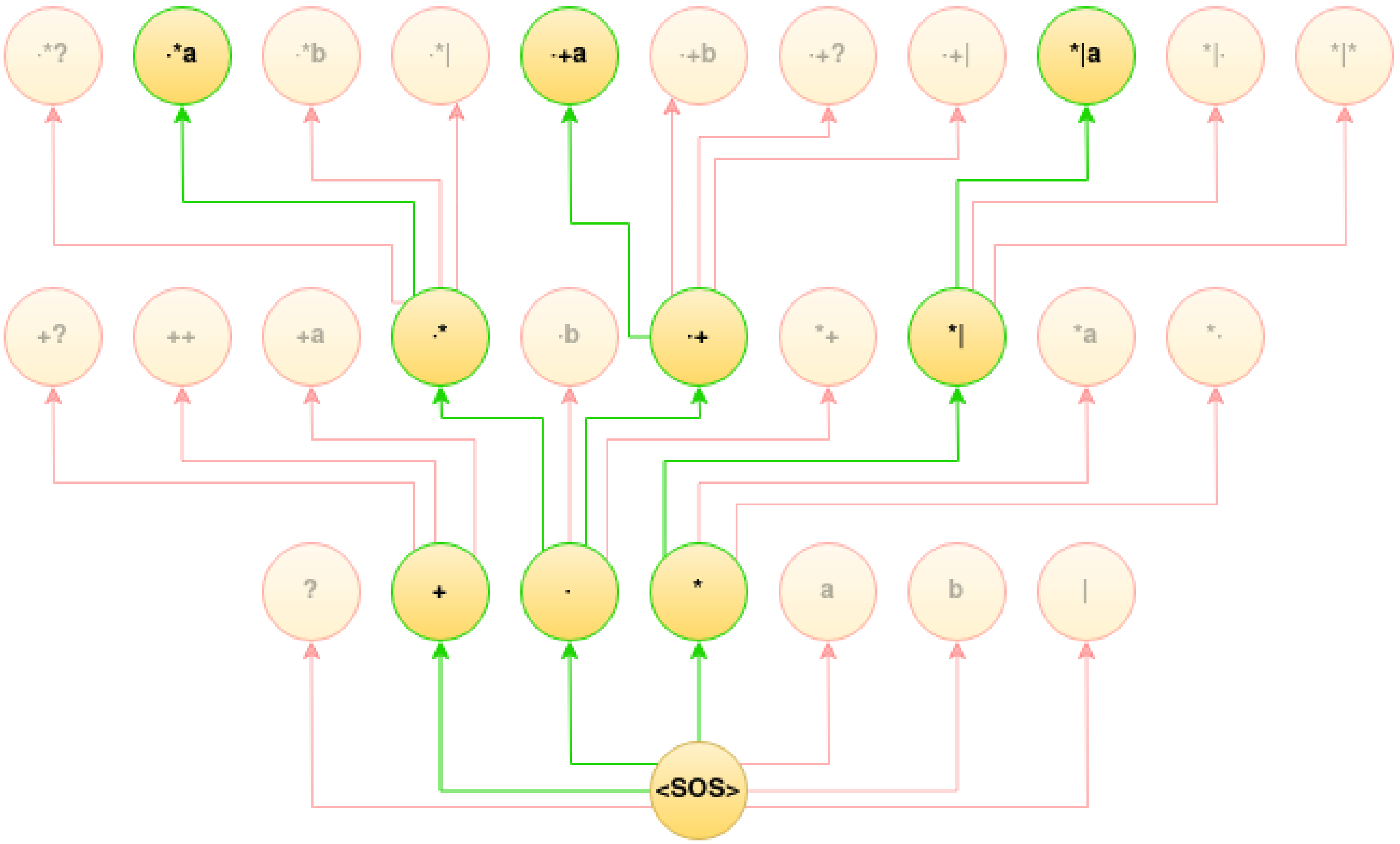
Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character



Improvement 2: beam search decoding (BSD)

- With BSD, we perform a beam search over the output of the network
- Candidate sequences are weighted by the product of probabilities of each character

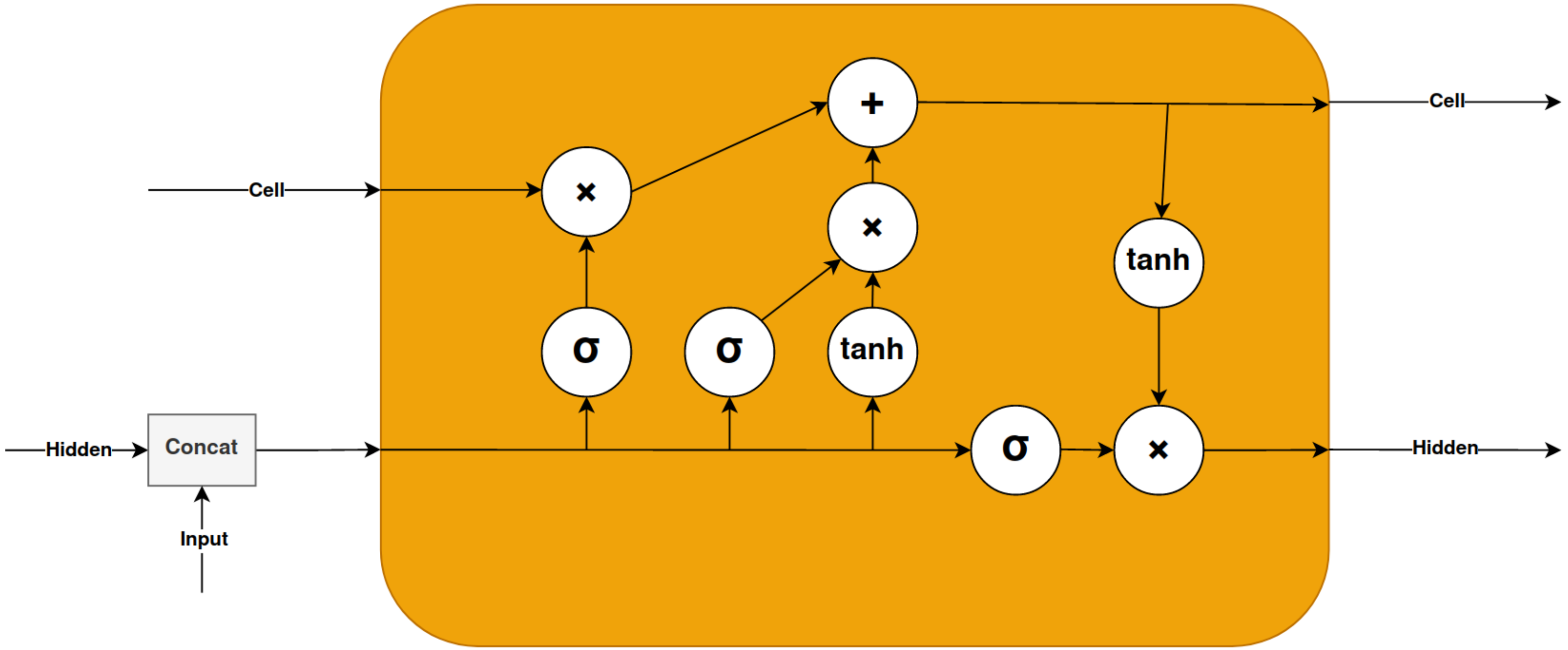


Improvement 2: beam search decoding (BSD)

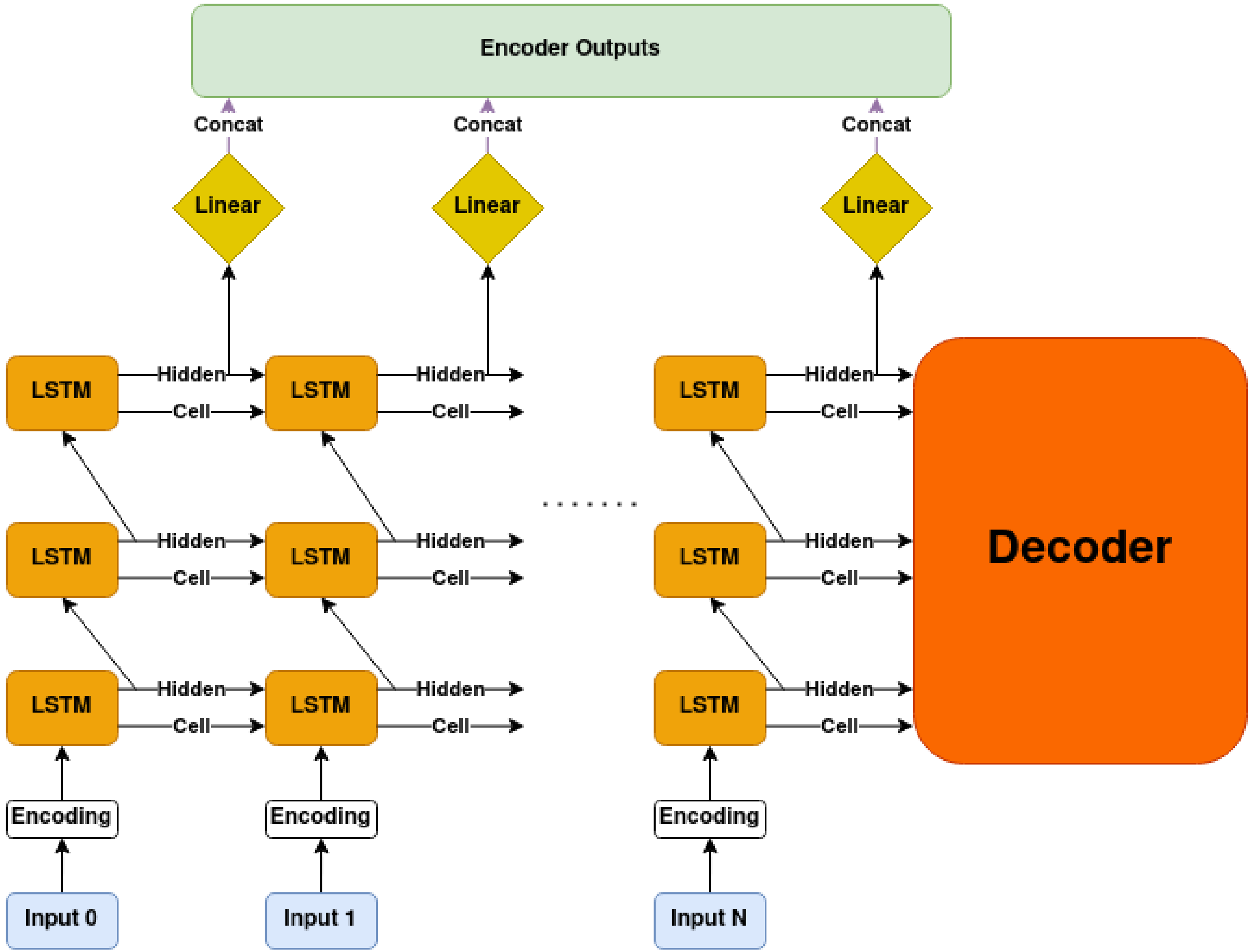
- Advantages:
  - Significant improvement of results quality
  - Possibility to include our home-made heuristic to guide the search

- Drawbacks:
  - Requires B time more computations

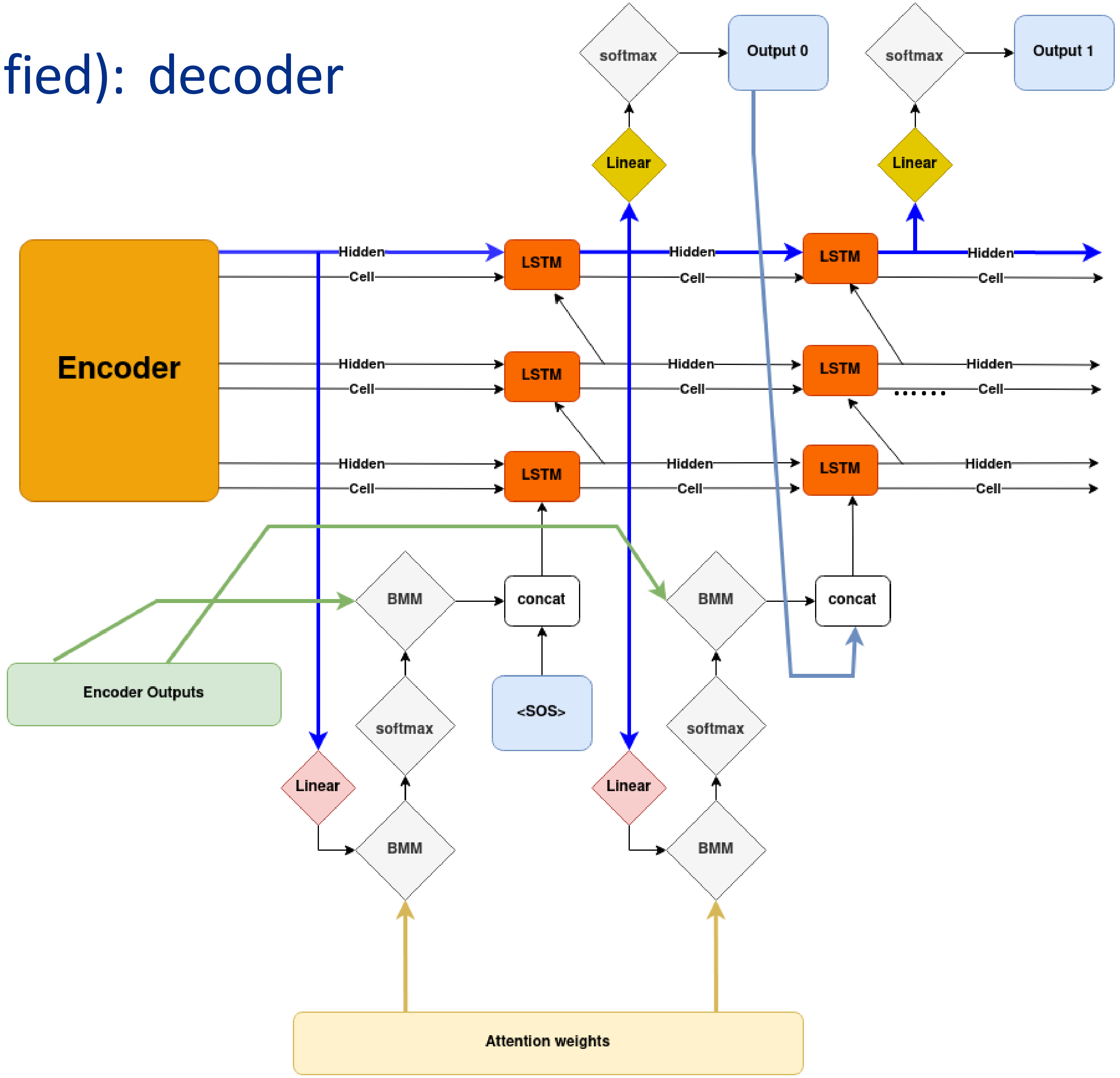
Final architecture (simplified): LSTM



Final architecture (simplified): encoder



Final architecture (simplified): decoder



Part 3: conclusion & future works

Conclusion

- A new approach to the grammatical induction problem

- Pros:
  - One to infer them all
  - No negative examples required
  - Once trained, very quick to infer a regular expression
  - Insights for RNN explainability and training methods

- Limitations:
  - Training takes time and energy
  - Only infers "short" regular expressions (~15 characters)

Future works

- Improving results:
  - Shuffle examples at the batch level to improve results
  - Use multiple decoders (prefix, infix, suffix) to improve results
  - Trim the BSD tree to improve results
  - Handle examples at the pattern level

- Explainability:
  - Study the influence of learning strategy to offer insight on RNN inner mechanisms.

Questions ?