(Part of the) Preliminaries for NN Expressiveness

Hao Su

July 1, 2026

Finite Automata & Regular Languages

Deterministic Finite Automata (DFA)

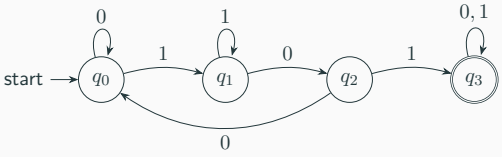
Definition (DFA)

A *deterministic finite automaton* is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma \to Q$ is the transition function,
4. $q_0 \in Q$ is the start state,
5. $F \subseteq Q$ is the set of accept states.



DFA accepting strings containing 101

Regular Languages & Regular Operations

Definition (Regular Languages)

A *string* is a finite sequence of symbols in Σ. A *language* is a set of strings. A language is called a *regular language* if some finite automaton recognizes it.

Regular Languages & Regular Operations

Definition (Regular Languages)

A *string* is a finite sequence of symbols in Σ. A *language* is a set of strings. A language is called a *regular language* if some finite automaton recognizes it.

Definition (Regular Operations)

For languages A and B, define the *regular operations*:

1. Union: $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$,
2. Concatenation: $A \circ B := \{xy \mid x \in A \text{ and } y \in B\}$, and
3. (Kleene) Star: $A^* := \{x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

When $k = 0$, the product $x_1 x_2 \cdots x_k$ is interpreted as ε.

Regular Expressions

Definition (Regular Expressions)

The *regular expressions* over Σ are defined recursively as follows:

1. a, for each $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. (R_1^*), where R_1 is a regular expression.

Regular Expressions

Definition (Regular Expressions)

The *regular expressions* over Σ are defined recursively as follows:

1. a, for each $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. (R_1^*), where R_1 is a regular expression.

Examples

$\Sigma^*101\Sigma^*, (\Sigma\Sigma)^*, a^*b^* (= \overline{\overline{\emptyset} \cdot b \cdot a \cdot \overline{\emptyset}}$, where $\overline{x}$ means the complement of x in Σ^*, hence is star-free) , $(aa)^*$ (not star-free.)

Turing Machines & Complexity

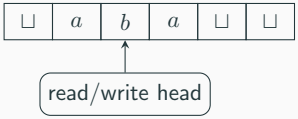
Definition (Turing Machines (TM))

A *Turing machine* is a 7-tuple,

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet, where $\sqcup \notin \Sigma$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.



δ	q_0	q_1
a	(q_0, a, R)	(q_1, b, L)
b	(q_1, a, L)	(q_1, b, L)
$\sqcup$	$(q_{\text{acc}}, \sqcup, R)$	$(q_{\text{rej}}, \sqcup, R)$

More on TMs

Church–Turing Thesis

Turing machines formalize the intuitive notion of an algorithm:

$$\text{effectively computable} \quad = \quad \text{computable by a Turing machine.}$$

This is not a mathematical theorem, but a thesis connecting an informal concept, “algorithm”, with a formal model of computation.

More on TMs

Church–Turing Thesis

Turing machines formalize the intuitive notion of an algorithm:

$$\text{effectively computable} \quad = \quad \text{computable by a Turing machine.}$$

This is not a mathematical theorem, but a thesis connecting an informal concept, "algorithm", with a formal model of computation.

TMs may not halt!

- TM M is a *decider* if it halts on all inputs.
- A language is *decidable* if some TM decides it.

More on TMs

Church–Turing Thesis

Turing machines formalize the intuitive notion of an algorithm:

$$\text{effectively computable} \quad = \quad \text{computable by a Turing machine.}$$

This is not a mathematical theorem, but a thesis connecting an informal concept, "algorithm", with a formal model of computation.

TMs may not halt!

- TM M is a *decider* if it halts on all inputs.
- A language is *decidable* if some TM decides it.
- $H_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ is undecidable.

More on TMs

Church–Turing Thesis

Turing machines formalize the intuitive notion of an algorithm:

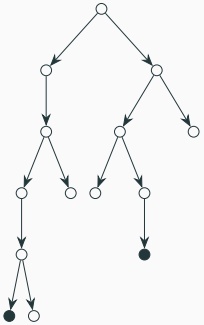
$$\text{effectively computable} \quad = \quad \text{computable by a Turing machine.}$$

This is not a mathematical theorem, but a thesis connecting an informal concept, "algorithm", with a formal model of computation.

TMs may not halt!

- TM M is a *decider* if it halts on all inputs.
- A language is *decidable* if some TM decides it.
- $H_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ is undecidable.
- Computability theory asks, *"Is L decidable?"*, and complexity theory asks, *"What resources does it take to decide L?"*

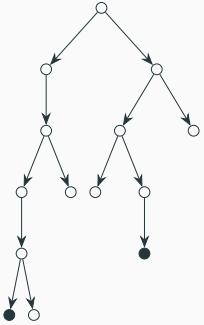
Nondeterminism



Definition (NTM)

A *Nondeterministic* TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Nondeterminism



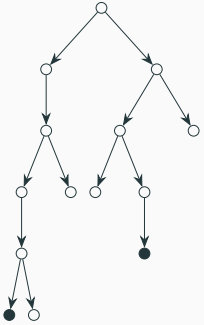
Definition (NTM)

A *Nondeterministic* TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Ways to think about nondeterminism

1. Computational: Fork new parallel thread and accept if any thread leads to an accept state.

Nondeterminism



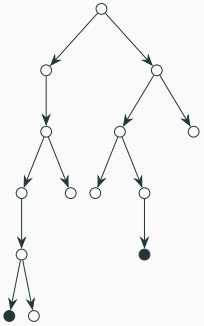
Definition (NTM)

A *Nondeterministic* TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Ways to think about nondeterminism

1. Computational: Fork new parallel thread and accept if any thread leads to an accept state.
2. Mathematical: Tree with branches. Accept if any branch leads to an accept state.

Nondeterminism



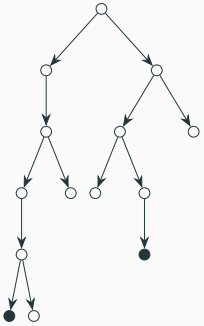
Definition (NTM)

A *Nondeterministic* TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Ways to think about nondeterminism

1. Computational: Fork new parallel thread and accept if any thread leads to an accept state.
2. Mathematical: Tree with branches. Accept if any branch leads to an accept state.
3. Magical: Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to accepting, if possible.

Nondeterminism



Definition (NTM)

A *Nondeterministic* TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$.

Ways to think about nondeterminism

1. Computational: Fork new parallel thread and accept if any thread leads to an accept state.
2. Mathematical: Tree with branches. Accept if any branch leads to an accept state.
3. Magical: Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to accepting, if possible.

Theorem

For every NTM there is an equivalent (D)TM.

Time Complexity

Definition (Time Complexity Class)

$f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for some fixed c independent of n.

Time Complexity

Definition (Time Complexity Class)

$f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for some fixed c independent of n. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. Define the *time complexity class* $\mathrm{TIME}(t(n))$ to be the collection of all languages decidable by a 1-tape TM in $O(t(n))$ steps.

Time Complexity

Definition (Time Complexity Class)

$f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for some fixed c independent of n. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. Define the *time complexity class* $\mathrm{TIME}(t(n))$ to be the collection of all languages decidable by a 1-tape TM in $O(t(n))$ steps.

Definition (Class P)

P is the class of languages that are decidable in polynomial time on a 1-tape TM. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

Time Complexity

Definition (Time Complexity Class)

$f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for some fixed c independent of n. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. Define the *time complexity class* $\mathrm{TIME}(t(n))$ to be the collection of all languages decidable by a 1-tape TM in $O(t(n))$ steps.

Definition (Class P)

P is the class of languages that are decidable in polynomial time on a 1-tape TM. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

NTIME and NP are similar defined only with NTM.

Time Complexity

Definition (Time Complexity Class)

$f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for some fixed c independent of n. Let $t : \mathbb{N} \to \mathbb{N}$ be a function. Define the *time complexity class* $\mathrm{TIME}(t(n))$ to be the collection of all languages decidable by a 1-tape TM in $O(t(n))$ steps.

Definition (Class P)

P is the class of languages that are decidable in polynomial time on a 1-tape TM. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

NTIME and NP are similar defined only with NTM.

How many steps does it take to verify the membership?

$\mathrm{P} =$ the class of languages for which membership can be decided quickly.

$\mathrm{NP} =$ the class of languages for which membership can be *verified* quickly.

Reducibility, NP-complete and NP-hard

Definition (Polynomial Time Reducibility)

A is *polynomial time reducible* to B ($A \leq_{\mathrm{P}} B$) if there is a function f computable in polynomial time such that $w \in A \Leftrightarrow f(w) \in B$. f is called a *reduction* from A to B.

Reducibility, NP-complete and NP-hard

Definition (Polynomial Time Reducibility)

A is *polynomial time reducible* to B ($A \leq_{\mathrm{P}} B$) if there is a function f computable in polynomial time such that $w \in A \Leftrightarrow f(w) \in B$. f is called a *reduction* from A to B.

What happens if B is decidable? Which is harder, A or B?

Reducibility, NP-complete and NP-hard

Definition (Polynomial Time Reducibility)

A is *polynomial time reducible* to B ($A \leq_{\mathrm{P}} B$) if there is a function f computable in polynomial time such that $w \in A \Leftrightarrow f(w) \in B$. f is called a *reduction* from A to B.

What happens if B is decidable? Which is harder, A or B?

Definition (NP-hardness)

A language B is NP-*hard* iff $\forall A \in \mathrm{NP}, A \leq_{\mathrm{P}} B$.

Reducibility, NP-complete and NP-hard

Definition (Polynomial Time Reducibility)

A is *polynomial time reducible* to B ($A \leq_{\mathrm{P}} B$) if there is a function f computable in polynomial time such that $w \in A \Leftrightarrow f(w) \in B$. f is called a *reduction* from A to B.

What happens if B is decidable? Which is harder, A or B?

Definition (NP-hardness)

A language B is NP-*hard* iff $\forall A \in \mathrm{NP}, A \leq_{\mathrm{P}} B$.

Definition (NP-completeness)

A language B is NP-*complete* iff (1) $B \in \mathrm{NP}$, and (2) $\forall A \in \mathrm{NP}, A \leq_{\mathrm{P}} B$.

Reducibility, NP-complete and NP-hard

Definition (Polynomial Time Reducibility)

A is *polynomial time reducible* to B ($A \leq_{\mathrm{P}} B$) if there is a function f computable in polynomial time such that $w \in A \Leftrightarrow f(w) \in B$. f is called a *reduction* from A to B.

What happens if B is decidable? Which is harder, A or B?

Definition (NP-hardness)

A language B is NP-*hard* iff $\forall A \in \mathrm{NP}, A \leq_{\mathrm{P}} B$.

Definition (NP-completeness)

A language B is NP-*complete* iff (1) $B \in \mathrm{NP}$, and (2) $\forall A \in \mathrm{NP}, A \leq_{\mathrm{P}} B$.

Hardness and completeness are usually defined like above, but *of course* the reductions differ! (why?)

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$\mathrm{P} \subseteq \mathrm{NP}$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE}$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXP}$$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP}$$

Space Complexity

Definition (Space Complexity of TMs)

A 1-tape TM M *runs in space* $f(n)$ iff M always halts and uses at most $f(n)$ tape cells on all inputs of length n (NTM: all branches halt and each uses at most that many).

Definition (Space Complexity Classes, PSPACE)

$$\mathrm{SPACE}(f(n)) = \{L | L \text{ is decided by an } O(f(n)) \text{ space TM.}\}$$

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k).$$

- NSPACE and $\mathrm{NPSPACE}$ are similarly defined.
- Replace P with EXP for the *exponential* counterpart of *polynomial*.

Theorem (Hierarchy of Complexity Classes)

$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP} \subseteq \mathrm{NEXPSPACE}$.

Masked Unique Hard-Attention Transformers

Masked Unique Hard-Attention (UHA)

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7$

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.

Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.

Masked Unique Hard-Attention (UHA)

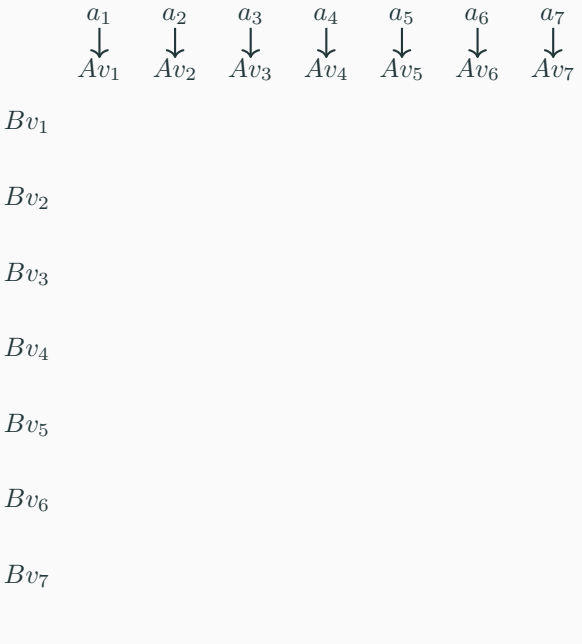
Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.

Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

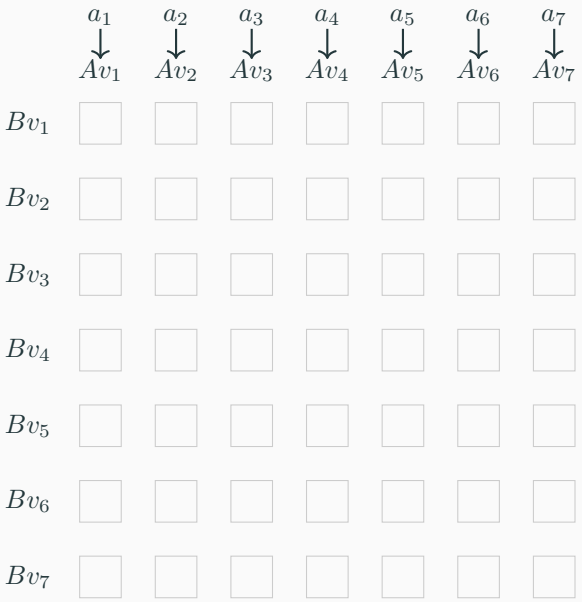
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

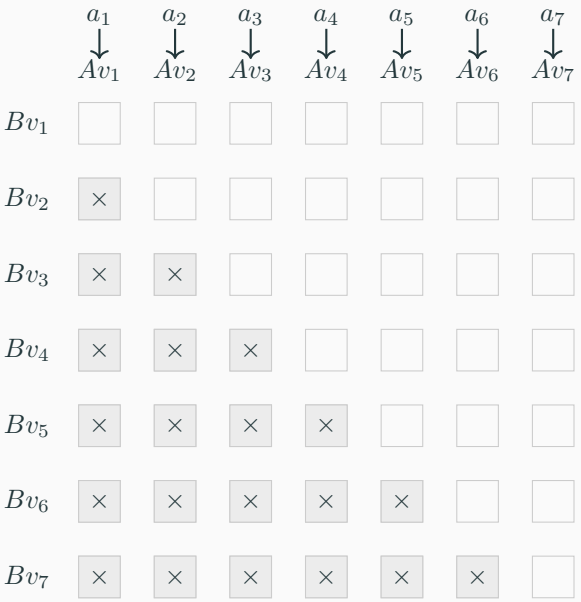
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

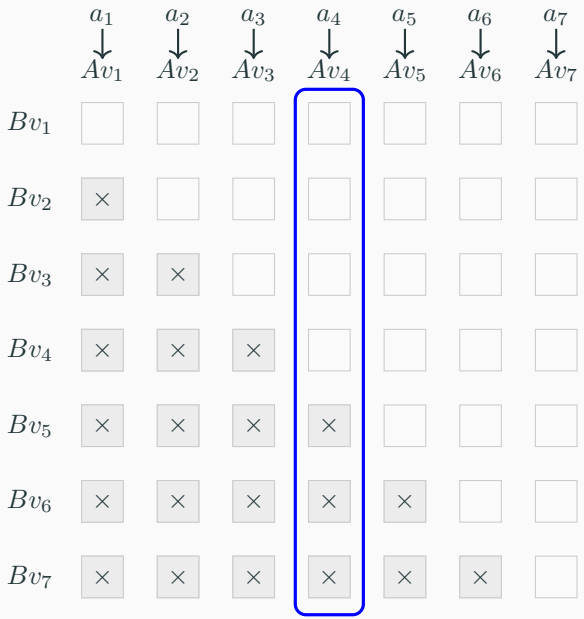
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

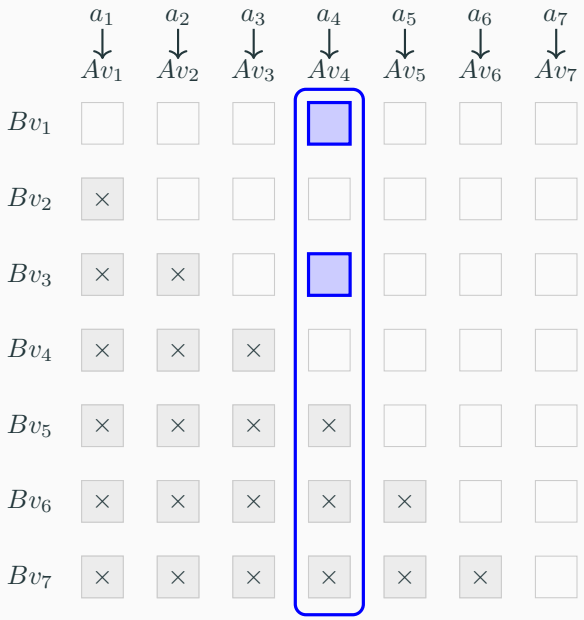
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

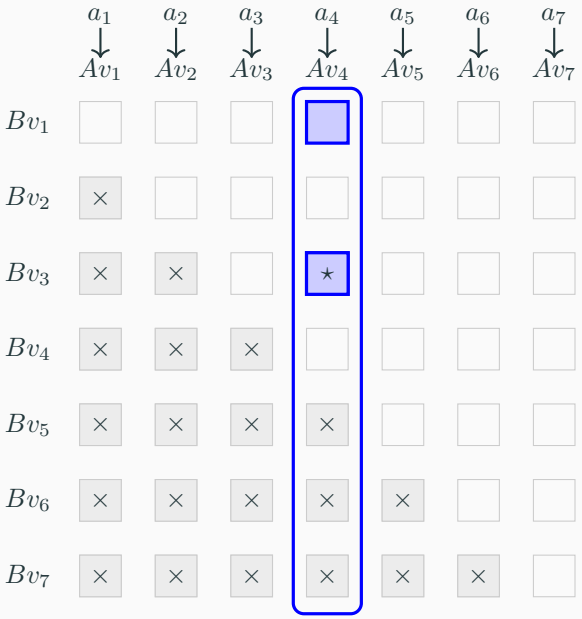
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

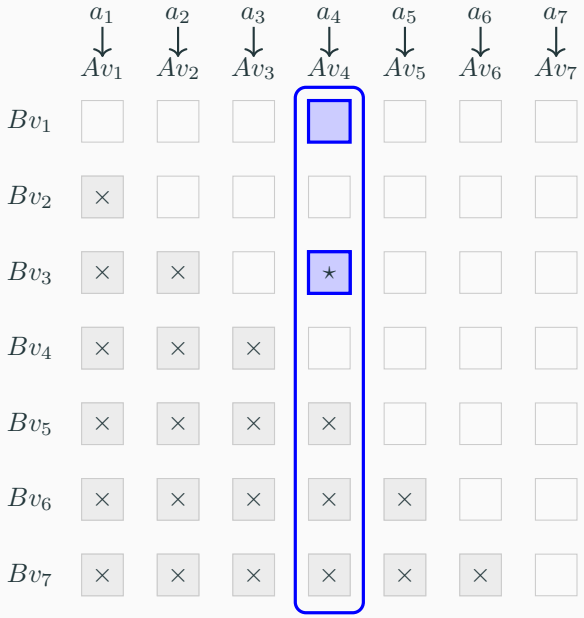
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.
- Tie Breaking τ. (why?)



Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.
- Tie Breaking τ. (why?)
- Value Update $C : \mathbb{Q}^{2r} \to \mathbb{Q}^s$.
  For example, $v'_4 = C(v_3, v_4)$.



ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

Definition (UHAT)

A *masked unique hard-attention transformer* (UHAT) is a length-preserving function $\mathcal{T} : \Sigma^+ \to (\mathbb{Q}^s)^+$ defined by application of a token embedding followed by application of a fixed sequence of UHA layers and ReLU layers of matching width.

ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

Definition (UHAT)

A *masked unique hard-attention transformer* (UHAT) is a length-preserving function $\mathcal{T} : \Sigma^+ \to (\mathbb{Q}^s)^+$ defined by application of a token embedding followed by application of a fixed sequence of UHA layers and ReLU layers of matching width.

Definition (UHAT Acceptance)

Assume that $\mathcal{T}$ is given together with an *acceptance vector* $t \in \mathbb{Q}^s$. The recognized language $L(\mathcal{T})$ is then the set of words on which $\mathcal{T}$ outputs a sequence $v_1, \ldots, v_n \in \mathbb{Q}^s$ with $\langle t, v_k \rangle > 0$, where $k \in [1, n]$ is either 1 or n depending on whether the first or last position is regarded the *output position* of $\mathcal{T}$.

Boolean RASP

Feature vectors & B-RASP programs

Vectors P_i in Restricted Access Sequence Processing (RASP)

- A feature vector assigns one value to each position of the input.
- A Boolean vector can record whether some property is true at each position.
- $Q_a(i) = 1 \Leftrightarrow w(i) = a$.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
P_3	0	1	0	0	0

P_3: position i is the first b.

Feature vectors & B-RASP programs

Vectors P_i in Restricted Access Sequence Processing (RASP)

- A feature vector assigns one value to each position of the input.
- A Boolean vector can record whether some property is true at each position.
- $Q_a(i) = 1 \Leftrightarrow w(i) = a$.

Definition (Sketching B-RASP)

- A *B-RASP* program *builds* new features with basic token features $(Q_a, Q_b, \dots)$.
- One of the vectors is designated as the *output vector* deciding acceptance.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
P_3	0	1	0	0	0

P_3: position i is the first b.

Building Operations

Position-wise operation

$P_{t+1}(i) := R(i)$, where $R(i)$ is some Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$.

Building Operations

Position-wise operation

$P_{t+1}(i) := R(i)$, where $R(i)$ is some Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$.

Attention operation

An attention operation defines a new feature vector P_{t+1} by letting each position i look at one selected position j,

$$P_{t+1}(i) := \triangleright_j [M(i,j), S(i,j)]\, V(i,j) : D(i),$$

where S, V, D are Boolean combinations of $\{P_1(i), \ldots, P_t(i)\}$.

- $M(i,j)$ is the mask, saying which positions j are visible from i.
- $S(i,j)$ is the score predicate, saying which visible positions are relevant.
- $\triangleright_j$ means: choose the rightmost visible j such that $S(i,j) = 1$.
- If such a position j exists, set $P_{t+1}(i) := V(i,j)$. If no such position exists, use the default value: $P_{t+1}(i) := D(i)$.

Example of B-RASP

Recognizing $0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$

Here $a_i \in \{a, b, c\}$ and $(a_i, a_{i+1}) \in H = \{(a, b), (b, c), (b, a), (c, b)\}$.

Example of B-RASP

Recognizing $0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$

Here $a_i \in \{a, b, c\}$ and $(a_i, a_{i+1}) \in H = \{(a, b), (b, c), (b, a), (c, b)\}$.
We check

1. the bit counter is incremented and
2. $(a_i, a_{i+1}) \in H$.

Example of B-RASP

Recognizing $0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$

Here $a_i \in \{a, b, c\}$ and $(a_i, a_{i+1}) \in H = \{(a, b), (b, c), (b, a), (c, b)\}$.
We check

1. the bit counter is incremented and
2. $(a_i, a_{i+1}) \in H$.

Successor Checking

$C_{+1}(i) := \triangleright_j[j < i, Q_\#(j)]\ V(i, j) : 1$ where

$$V(i,j) = \bigvee_{k=1}^{4} \left[\left(\bigwedge_{r<k} \neg C_r(i) \wedge C_r(j) \right) \wedge C_k(i) \wedge \neg C_k(j) \wedge \left(\bigwedge_{r>k} C_r(i) \leftrightarrow C_r(j) \right) \right].$$

Example of B-RASP

Recognizing $0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$

Here $a_i \in \{a, b, c\}$ and $(a_i, a_{i+1}) \in H = \{(a, b), (b, c), (b, a), (c, b)\}$.
We check

1. the bit counter is incremented and
2. $(a_i, a_{i+1}) \in H$.

Successor Checking

$C_{+1}(i) := \triangleright_j[j < i, Q_\#(j)]\ V(i, j) : 1$ where

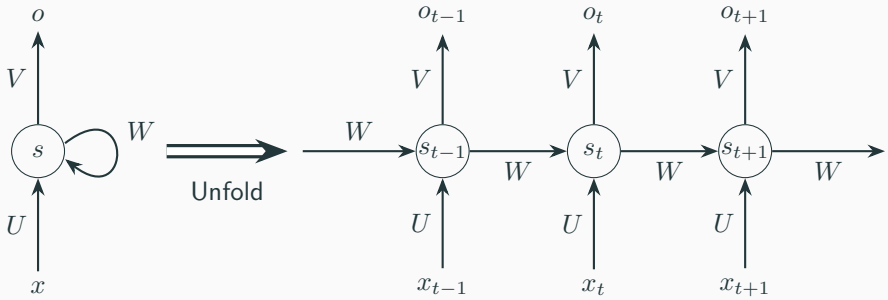
$$V(i,j) = \bigvee_{k=1}^{4} \left[\left(\bigwedge_{r<k} \neg C_r(i) \wedge C_r(j) \right) \wedge C_k(i) \wedge \neg C_k(j) \wedge \left(\bigwedge_{r>k} C_r(i) \leftrightarrow C_r(j) \right) \right].$$

Format Checking

$$M_{\leftarrow}(i) := \triangleright_j\left[j < i,\ Q_a(j) \vee Q_b(j) \vee Q_c(j)\right] \bigvee_{(h,h') \in H} (Q_h(j) \wedge Q_{h'}(i)) : 1.$$

Recurrent Neural Networks (RNN)

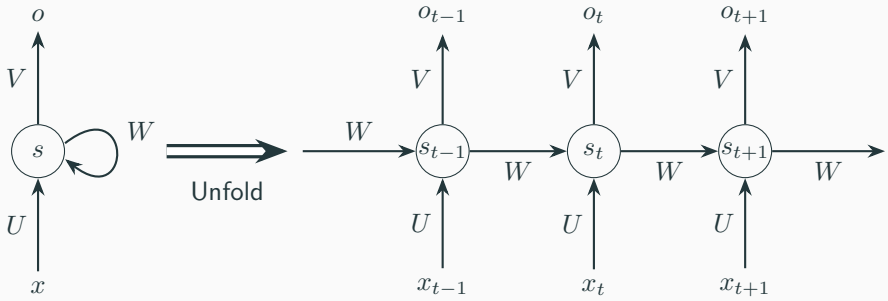
Recurrent Neural Networks (RNN)



Notes

- The state $s \in \mathbb{R}^d$ updates along with the input.

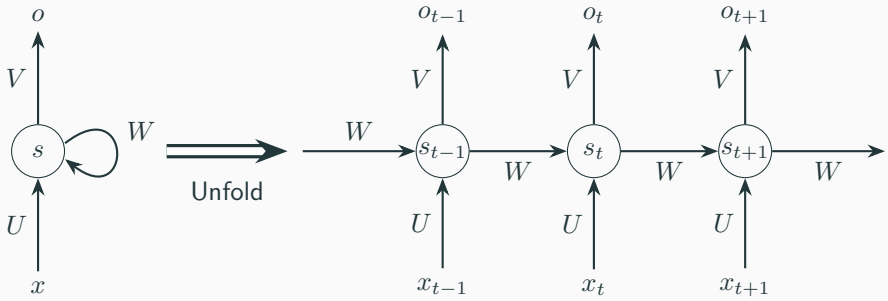
Recurrent Neural Networks (RNN)



Notes

- The state $s \in \mathbb{R}^d$ updates along with the input.
- Decision on input: $f(s_n) \in \{\mathrm{Acc}, \mathrm{Rej}\}$.

Recurrent Neural Networks (RNN)



Notes

- The state $s \in \mathbb{R}^d$ updates along with the input.
- Decision on input: $f(s_n) \in \{\mathrm{Acc}, \mathrm{Rej}\}$.
- Fixed precision (k bits) makes an RNN representable with an FA with 2^{kd} states.

Expressiveness and Succinctness

Two Measures of Expressiveness

Here we discuss the expressiveness of *classes* of models (Transformers, DFAs, etc.)

Two Measures of Expressiveness

Here we discuss the expressiveness of *classes* of models (Transformers, DFAs, etc.)

What languages can the models recognize?

For example, DFAs cannot recognize $a^n b^n$ (why?) but TMs can. Hence in a sense we say TMs are more expressive than DFAs.

Two Measures of Expressiveness

Here we discuss the expressiveness of *classes* of models (Transformers, DFAs, etc.)

What languages can the models recognize?

For example, DFAs cannot recognize $a^n b^n$ (why?) but TMs can. Hence in a sense we say TMs are more expressive than DFAs.

Theorem

Certain transformers recognize star-free languages, whereas certain RNNs can recognize all regular languages.

If RNNs are just better why bother with tranformers?

Two Measures of Expressiveness

Here we discuss the expressiveness of *classes* of models (Transformers, DFAs, etc.)

What languages can the models recognize?

For example, DFAs cannot recognize $a^n b^n$ (why?) but TMs can. Hence in a sense we say TMs are more expressive than DFAs.

Theorem

Certain transformers recognize star-free languages, whereas certain RNNs can recognize all regular languages.

If RNNs are just better why bother with tranformers?

Given L, how big a model we need to recognize it?

This is mainly by comparison, e.g., the space we need to store a transformer of L is much smaller than an RNN.

Two Measures of Expressiveness

Here we discuss the expressiveness of *classes* of models (Transformers, DFAs, etc.)

What languages can the models recognize?

For example, DFAs cannot recognize $a^n b^n$ (why?) but TMs can. Hence in a sense we say TMs are more expressive than DFAs.

Theorem

Certain transformers recognize star-free languages, whereas certain RNNs can recognize all regular languages.

If RNNs are just better why bother with tranformers?

Given L, how big a model we need to recognize it?

This is mainly by comparison, e.g., the space we need to store a transformer of L is much smaller than an RNN.

The latter is called *succinctness*.

Size Measures and Succinctness

Definition (Size)

The *size* of $\mathcal{R}$, denoting an arbitrary model, is given by the length of its usual binary encoding. For neural networks fix their precision k.

Size Measures and Succinctness

Definition (Size)

The *size* of $\mathcal{R}$, denoting an arbitrary model, is given by the length of its usual binary encoding. For neural networks fix their precision k.

Definition (Succinctness)

Let $\mathcal{C}_1, \mathcal{C}_2$ be classes of finite representations of languages. Say that $\mathcal{C}_1$ can be exponentially more succinct than $\mathcal{C}_2$ if for every function $f \in 2^{o(n)}$ there is an $\mathcal{R}_1 \in \mathcal{C}_1$ such that any $\mathcal{R}_2 \in \mathcal{C}_2$ representing the same language as $\mathcal{R}_1$ has size

$$|\mathcal{R}_2| > f(|\mathcal{R}_1|).$$

Similarly, define doubly exponentially more succinct using functions $f \in 2^{2^{o(n)}}$ instead.

Note that this is the lower bound!

Questions?