Australian National University

CYBERNETIC STUDIO

Weighted Randomness

Learn how to make random choices where some options are more likely than others — an operation at the core of all generative AI.

You will need

- 10-sided dice (d10)
- coloured marbles or beads in a bag

Your goal

To randomly choose from a fixed set of outcomes according to a given probability distribution.

Key idea

Sometimes we need to make random choices where some outcomes are more likely than others. There are ways to do this which ensure certain relationships *on average* between the outcomes (e.g. one outcome happening twice as often as another one).

Algorithm 1: beads in a bag

- **materials**: coloured beads, bag
- **setup**: count out a number of beads corresponding to the desired weights for each outcome
- **sampling procedure**: shake the bag, then draw one bead without looking

Example

You want to choose an ice cream flavour: `vanilla` 50% of the time, `chocolate` 30%, and `strawberry` 20%.

- add 5 white beads to the bag (corresponding to `vanilla`)
- add 3 brown beads to the bag (corresponding to `chocolate`)
- add 2 red beads to the bag (corresponding to `strawberry`)

Draw a bead from the bag — that's your ice-cream choice for today.

Algorithm 2: dice with ranges

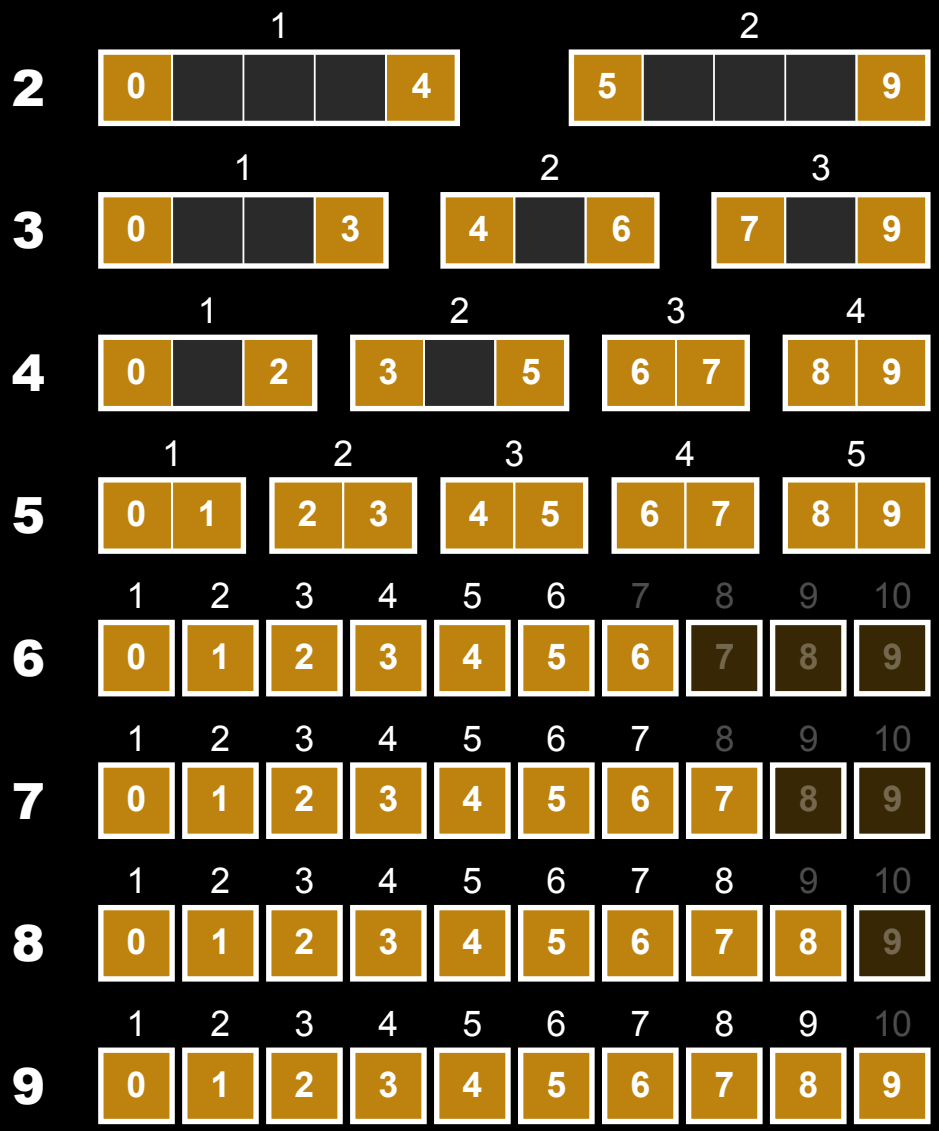
- **materials**: d10 (or d6, d20 as alternatives)
- **setup**: assign number ranges proportional to weights (see table, right)
- **sampling procedure**: roll the die, then look up the corresponding outcome

Example

- for 60% vanilla/40% chocolate, roll a d10: 1-6 means `vanilla`, 7-10 means `chocolate`
- for 50% vanilla/30% chocolate/20% strawberry, roll a d10: 1-5 means `vanilla`, 6-8 means `chocolate`, 9-10 means `strawberry`

You can use different dice (d6, d10, d20, d120, etc.), it will just change the number ranges corresponding to each outcome.

d10 dice roll → outcome mapping table



CYBERNETIC STUDIO

Australian National University

CYBERNETIC STUDIO

Basic Training

Build a bigram language model that tracks which words follow which other words in text.

You will need

- some text (e.g. a few pages from a kids book, but can be anything)
- pen, pencil and grid paper

Your goal

To produce a grid that captures the patterns in your input text data. This grid is your *bigram language model*. **Stretch goal**: keep training your model on more input text.

Key idea

Language models learn by counting patterns in text. “Training” means building/constructing a model (i.e. filling out the grid) to track which words follow other words.

Algorithm

1. **preprocess your text**:
   - convert everything to lowercase
   - treat words, commas and full stops as separate “words” (and ignore all other punctuation and whitespace)
2. **set up your grid**:
   - take the first word from your text
   - write it in both the first row header and first column header of your grid
3. **fill in the grid** one *word pair* at a time:
   - find the row for the first word (in your training text) and the column for the second word
   - add a tally mark in that cell (if the word isn’t in the grid yet, add a new row *and* column for it)
   - shift along by one word (so the second word becomes your “first” word)
   - repeat until you’ve gone through the entire text

Example

Original text: *“See Spot run. See Spot jump. Run, Spot, run. Jump, Spot, jump.”*

Preprocessed text: `see` `spot` `run` `.` `see` `spot` `jump` `.` `run` `,` `spot` `,` `run` `.` `jump` `,` `spot` `,` `jump` `.`

After the first two words (`see` `spot`) the model looks like:

	see	spot				
see		|				
spot						

After the full text the model looks like:

	see	spot	run	.	jump	,
see		||				
spot			|		|	||
run				||		|
.	|		|		|	
jump				||		|
,		||	|		|	

Australian National University

CYBERNETIC STUDIO

Basic Inference

Use a pre-trained model to generate new text through weighted random sampling.

You will need

- your completed bigram model (i.e. your filled-out grid) from *Basic Training*
- d10 (or similar) for weighted sampling
- pen & paper for writing down the generated “output text”

Your goal

To generate new text from your bigram language model. **Stretch goal**: keep going, generating as much text as possible. Write a whole book!

Key idea

Language models generate text by predicting one word at a time based on learned patterns. Your trained model provides the “next word” options and their relative probabilities; dice rolls provide the randomness to choose one of those options (and this process can be repeated indefinitely).

Algorithm

1. **choose a starting word** — pick any word from the first column of your grid
2. **look at that word's row** to identify all possible next words and their counts
3. **roll dice weighted by the counts** (see the *Weighted Randomness* module)
4. **write down the chosen word** and use that as your next starting word
5. **repeat** from step 2 until you reach the desired length *or* a natural stopping point (e.g. a full stop `.`)

Example

Using the same bigram model from the example in *Basic Training*:

	`see`	`spot`	`run`	`.`	`jump`	`,`
`see`		||				
`spot`			|		|	||
`run`				||		|
`.`	|		|		|	
`jump`				||		|
`,`		||	|		|	

- choose (for example) `see` as your starting word
- `see` (row) → `spot` (column); it's the only option, so write down `spot` as next word
- `spot` → `run` (25%), `jump` (25%) or `,` (50%); roll dice to choose
- let's say dice picks `run`; write it down
- `run` → `.` (67%) or `,` (33%); roll dice to choose
- let's say dice picks `.`; write it down
- `.` → `see` (33%), `run` (33%) or `jump` (33%); roll dice to choose
- let's say dice picks `see`; write it down
- `see` → `spot`; it's the only option, so write down `spot`... and so on

After the above steps, the full output text is *"see spot run. see spot"*

Australian National University

CYBERNETIC STUDIO

Pre-trained Model Inference

Use a (slightly larger) pre-trained model to generate new text through weighted random sampling.

You will need

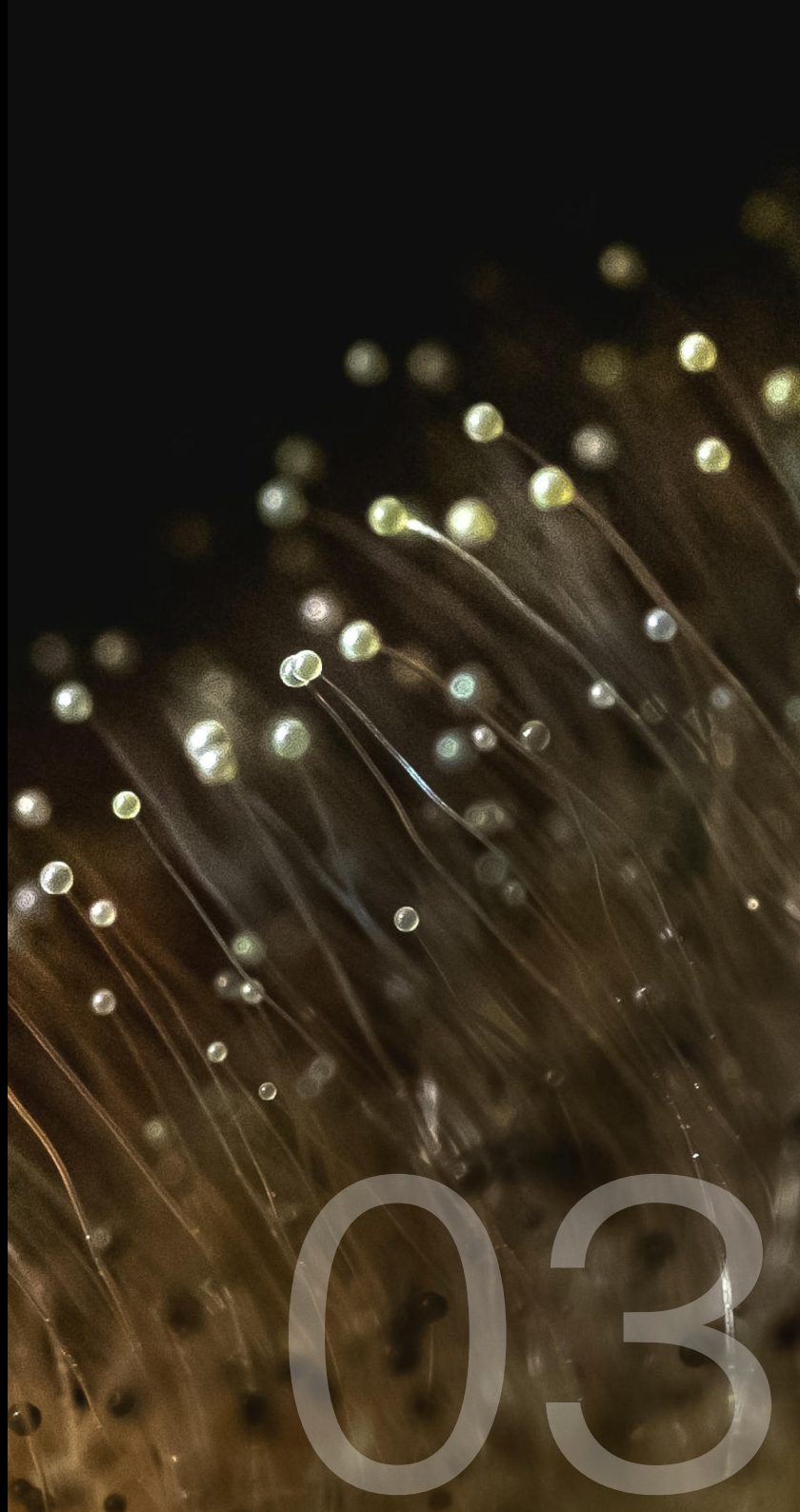
- a pre-trained model booklet
- d10 (or similar) for weighted sampling
- pen & paper for writing down the generated “output text”

Your goal

To generate new text using a pre-trained language model without having to train it yourself. **Stretch goal**: without looking at the title, try and guess which text the booklet model was trained on.

Key idea

You don’t need to train your own model to use one. Pre-trained models capture patterns from large amounts of text and can be used to generate new text just like your “hand-trained” model from *Basic Training*.



Algorithm

Full instructions are at the front of the booklet, but here's a quick summary:

1. **choose a starting word** — pick any bold word from the booklet and write it down
2. **look up the word's entry** (use the booklet like a dictionary) to find all possible next words
3. **roll your d10(s)**:
   - if the word has a ◆*n* indicator then roll a d10 *n* times, otherwise roll it once
   - interpret these dice rolls as digits from a single number (e.g. if you roll 4, then 7, then 2 then your number is 472)
4. **scan through the "next word" options** to find your next word: the first number which is greater than or equal to your roll indicates your next word (write it down)
5. **repeat** from step 2 using this new word, continuing until you reach a natural stopping point (like a period) or your desired length

Example 1: single d10

Your current word is **"cat"** and its entry shows:

cat → **4**|sat **7**|ran **10**|slept

- no indicator means roll your (single) d10
- you roll a 6
- scan through options: **7**|ran is the first number ≥ 6
- your next word is "ran": write it down, look it up and continue

Example 2: multiple d10s

Your current word is **"the"** and its entry shows:

the ◆2 → **33**|cat **66**|dog **99**|end

- the indicator with **2** inside means roll your d10 twice
- you roll 5 and 8, giving you 58
- scan through options: **66**|dog is the first number ≥ 58
- your next word is "dog": write it down, look it up and continue

CYBERNETIC STUDIO

Australian National University

CYBERNETIC STUDIO

Trigram Model

Extend the bigram model to consider *two* words of context instead of one, leading to better text generation.

You will need

- same as *Basic Training* module
- additional paper for the three-column model
- pen, paper & dice as per *Basic Inference*

Your goal

To train a trigram language model (a table this time, not a grid like your bigram model from *Basic Training*) and use it to generate text. **Stretch goal**: train on more data, or generate more text.

Key idea

More context leads to better predictions. A trigram model considers two previous words instead of one, demonstrating the trade-off between context length and data requirements that shapes all language models.

04

Algorithm (training)

1. **create a four-column table** (see example on right)
2. **extract all word triples**: for each (overlapping) *word 1*/*word 2*/*word 3* "triple" in your text increment the **count** column for that triple, or create a new row if it's a triple you've never seen before and set the count to 1 (note: row order doesn't matter)

Example (training)

After the first four words (`see` `spot` `run` `.`) the model is:

word 1	word 2	word 3	count
`see`	`spot`	`run`	|
`spot`	`run`	`.`	|

After the full text (`see` `spot` `run` `.` `see` `spot` `jump` `.`) the model is:

word 1	word 2	word 3	count
`see`	`spot`	`run`	|
`spot`	`run`	`.`	|
`run`	`.`	`see`	|
`.`	`see`	`spot`	|
`see`	`spot`	`jump`	|
`spot`	`jump`	`.`	|

Note: the order of the rows doesn't matter, so you can re-order to group them by **word 1** if that helps.

Algorithm (inference)

1. pick any row from your table; write down **word 1** and **word 2** as your starting words
2. find *all rows* where **word 1** and **word 2** are exact matches for your two starting words, and make note of their **count** columns
3. as per *Basic Inference* roll a d10 weighted by the counts and select the **word 3** associated with the chosen row
4. move along by *one* word (so **word 2** becomes your new **word 1** and **word 3** becomes your new **word 2**) and repeat from step 2

Example (inference)

1. from the table above, choose `see` (**word 1**) and `spot` (**word 2**) as your starting words
2. find all rows with **word 1** = `see` and **word 2** = `spot`; in this case rows 1 and 5 (both have *count* == 1)
3. roll a d10 and write down the **word 3** from the row chosen by the dice roll
4. move along by *one* word (so **word 1** is `spot` and **word 2** is either `run` or `jump` depending on your dice roll) and repeat from step 2

Australian National University

Context Columns

Enhance the bigram model with context columns that capture grammatical and semantic patterns.

You will need

- your completed bigram model from *Basic Training*
- pen, paper & dice as per *Basic Inference*

Your goal

To add new “context” columns to an existing bigram model and generate text from your newly context-aware model. **Stretch goal**: add and evaluate your own new context columns.

Key idea

The concept of attention — selectively focusing on relevant context — is a key innovation in Large Language Models. Adding context columns to your model gives it more information about which previous words matter most for prediction, leading to better generated text (with the trade-off being a slightly larger grid and more complex algorithm).

CYBERNETIC STUDIO

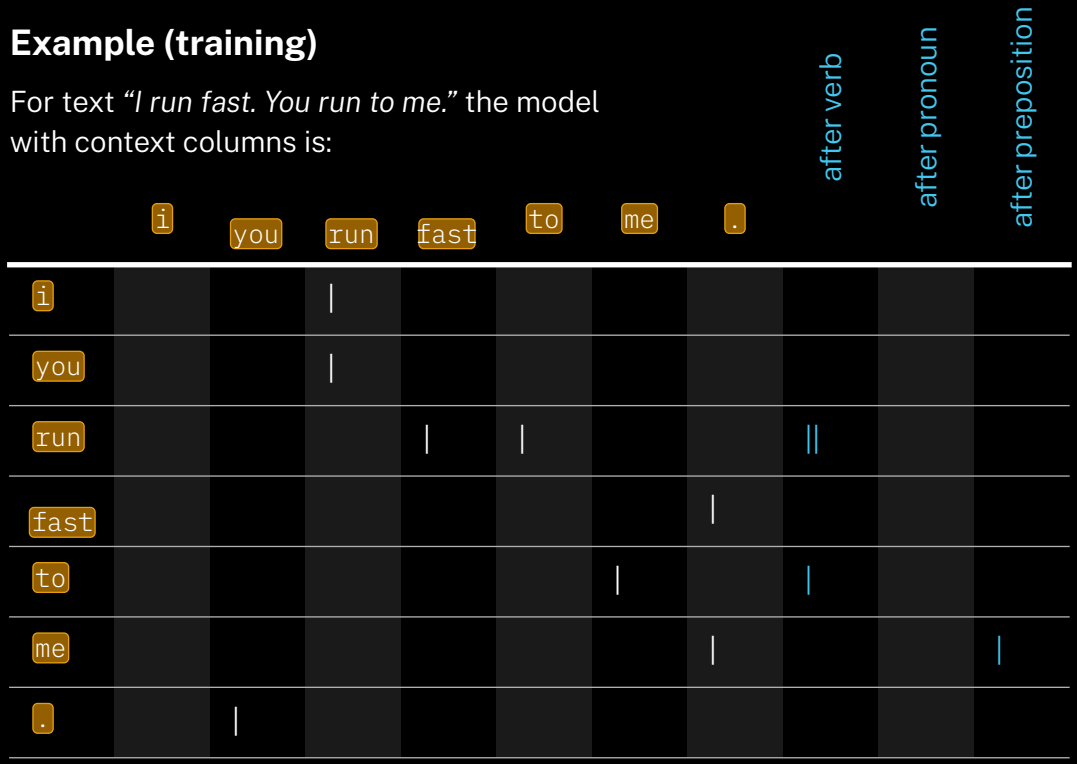
Algorithm (training)

1. **add context columns** to your existing bigram model: *after verb*, *after pronoun* and *after preposition*
2. proceed as per *Basic Training*, but each time after updating the cell count for a word pair:
   - if the first word is a verb, increment the value in the second word's *after verb* column
   - if the first word is a pronoun (I/you/they etc.), increment the value in the second word's *after pronoun* column
   - if the first word is a preposition (in/on/at/with/to etc.), increment the value in the second word's *after preposition* column

This is a little tricky to get the hang of, but the key point is that you're updating two different rows each time — once for the "word follows word" cell, and once for the "context column" cell.

Example (training)

For text *"I run fast. You run to me."* the model with context columns is:



Algorithm (inference)

1. **choose a starting word**
2. check its row to identify the "normal" transition counts, but *also* check if the starting word is a verb/pronoun/preposition and if so add the values from the relevant "context" column before using a d10 to choose the next word
3. **repeat** from step 2 until you reach the desired length *or* a natural stopping point (e.g. a full stop `.`)

If you like, you can add your own context columns (based on patterns which *you* think are important).

Example (inference)

Starting word: `run` (a verb):

1. check `run` row: potential next words are `fast` (1) or `to` (1)
2. check all context columns: for `to` the **after verb** column has a count of 1 (appears after verbs)
3. combine both counts: roll a dice to choose either `fast` (1) or `to` ($1 + 1 = 2$)
4. repeat from step 1 until you reach the desired length *or* a natural stopping point (e.g. a full stop `.`)

CYBERNETIC STUDIO

Australian National University

CYBERNETIC STUDIO

Word Embeddings

Transform words into numerical vectors that capture meaning, revealing the semantic relationships between words in your model.

You will need

- your completed bigram model grid (including context columns if you have them)
- another empty grid (same size as your bigram model)
- pen, paper & dice as per *Basic Inference*

Your goal

To create a similarity matrix (another square grid) which captures how similar (or different) all the words in your bigram model are. **Stretch goal**: create a visual representation of this similarity matrix.

Key idea

Each word's row in your model is its embedding under that model — a numerical fingerprint that captures meaning through context. Distances between words reveal grammatical and semantic relationships. Similar words have similar embeddings.

Algorithm

For this algorithm you'll need two grids: your original *bigram model* grid and a new *embedding distance* grid (with the same words as row/column headers, but otherwise blank to start with).

1. for the first row and second row in the bigram model, add up the total of the absolute differences between corresponding cells in the two rows and write it in the empty cell for that word pair in the embedding distance grid
2. fill out the embedding distance grid by repeating step 1 for all different pairs of rows in the bigram model grid

Example

Original text: "*See Spot. Spot runs.*"

Completed bigram model grid:

	see	spot	.	runs
see		1		
spot			1	1
.		1		
runs			1	

The embedding distance between the first two rows (see and spot) is the sum of the absolute differences between corresponding elements (0 for blank cells):

$$\begin{aligned} d(\text{see}, \text{spot}) &= |0-0| + |1-0| + |0-1| + |0-1| \\ &= 0 + 1 + 1 + 1 \\ &= 3 \end{aligned}$$

Put this distance in the embedding distance grid (note diagonals are already pre-filled with 0 as well):

	see	spot	.	runs
see	0	**3**		
spot		0		
.			0	
runs				0

Complete embedding distance grid (no need to fill out the bottom triangle — the embedding distance is symmetric):

	see	spot	.	runs
see	0	3	0	2
spot		0	3	2
.			0	2
runs				0

The distances show that see and . have identical embeddings (distance = 0), while see and spot are quite different (distance = 3).

Australian National University

CYBERNETIC STUDIO

Sampling

When generating text the language model gives several different options for which word could come next in the generated text — which one to choose?

You will need

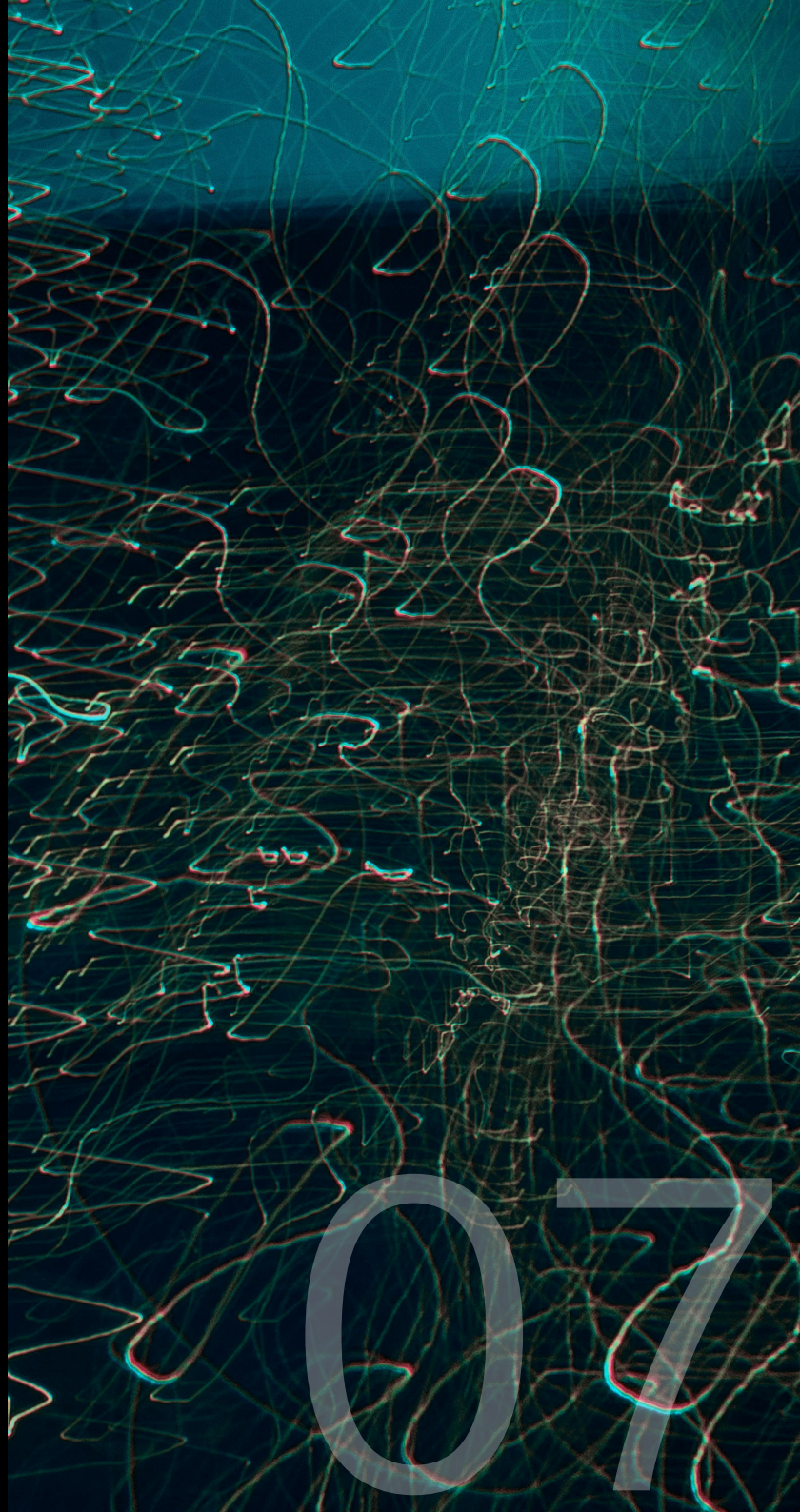
- a completed model from an earlier module
- pen, paper & dice as per *Basic Inference*

Your goal

To generate text (with the same model) using at least two different temperature values and at least two different truncation strategies. **Stretch goal**: design and evaluate your own truncation strategy.

Key idea

There are lots of different sampling algorithms — ways to select the next word during inference (text generation). Each strategy has different strengths and weaknesses, and can significantly influence the generated text even if the rest of the model is identical.



Temperature control

The temperature parameter (a number) controls the randomness by adjusting the relative likelihood of probable vs improbable words. The higher the temperature, the more uniform the distribution becomes, increasing randomness and allowing more sampling from unlikely words.

Algorithm

1. when sampling the next word, divide all counts by temperature value (round down, min 1)

Example

If the counts in a given row are:

	`spot`	`run`	`jump`	`.`
`see`	4	2	1	1

1. if **temperature = 1**: use counts as-is (4, 2, 1, 1)
   - `spot` is 2x as likely as `run`, 4x as likely as `jump` or `.`
2. if **temperature = 2**: divide counts by 2 → (2, 1, 1, 1)
   - `spot` still most likely, but only 2x as likely as others
3. if **temperature = 4**: divide counts by 4 → (1, 1, 1, 1)
   - all words equally likely

Truncation strategies

Truncation narrows the viable “next word options” by ruling out some options. Any truncation strategy can be combined with temperature control.

Greedy sampling

1. find current word’s row
2. select the word with the highest count
3. if there’s a tie, roll dice to choose equally among the most likely options

Haiku sampling

1. track syllables in current line (5-7-5 pattern)
2. roll dice to select next word as normal
3. if selected word exceeds line’s syllable limit, re-roll
4. start new line when syllable count reached

Non-sequitur sampling

1. find current word’s row
2. pick the column with the lowest (non-zero) count
3. if there’s a tie, roll dice to choose equally among the least likely options

No-repeat sampling

1. track all words used in current sentence
2. roll dice to select next word as normal
3. if word already used, re-roll
4. if no valid options remain, insert `.` and continue

Alliteration sampling

1. note first letter/sound of previous word
2. if any next-word options start with same letter/sound, sample only from those alliterative options
3. otherwise use standard sampling

Australian National University

CYBERNETIC STUDIO

LoRA

Efficiently adapt a trained language model to a new domain or style without retraining the entire model from scratch.

You will need

- a completed bigram model from an earlier module
- pen, pencil and grid paper
- some new text in a different style or domain

Your goal

To create a lightweight “adaptation layer” that modifies your existing model’s behaviour for a new domain. **Stretch goal**: combine the base model and LoRA layer with different mixing ratios.

Key idea

Low-Rank Adaptation (LoRA) allows you to specialise a language model by adding small adjustments rather than retraining everything. The LoRA layer is typically much smaller than the base model because you only track the *changes* from the base model, not the full weights.

08

Algorithm

1. **choose an existing bigram model as the "base model"**
2. **train a LoRA layer**:
   - start with a new grid (same columns as the base model)
   - process your new domain-specific text using the same algorithm as *Basic Training*, but only include rows for words that appear in your new text
3. **apply the adaptation**:
   - as per *Basic Inference*, but add the counts from both grids (if current word is in the LoRA grid)
   - optionally scale the LoRA values up or down to control adaptation strength

Example

Base model trained on general text:

	saw	they	we	the	a	red
saw		2		4	2	1
they	1			2	1	
we				3		
the			1			
a						2
red		1				

LoRA layer trained on "I saw a red cat. I saw the red dog." (smaller — only 1 row):

	saw	they	we	the	a	red
saw				1	1	2

Combined model (add counts):

	saw	they	we	the	a	red
saw		2		5	3	3
they	1			2	1	
we				3		
the			1			
a						2
red		1				

- `saw` row:
  - `[–,2,–,4,2,1]` (base)
  - `[–,–,–,1,1,2]` (LoRA)
  - `[–,2,–,5,3,3]` (base + LoRA)
- `red` now equally likely as `the` after `saw`
- other rows: base + zero = unchanged
- LoRA is smaller: only 1 row vs 6 in base model

CYBERNETIC STUDIO

Australian National University

CYBERNETIC STUDIO

Synthetic Data

Use your language model to generate new training data, then train a new model on that synthetic data to see how patterns degrade or change.

You will need

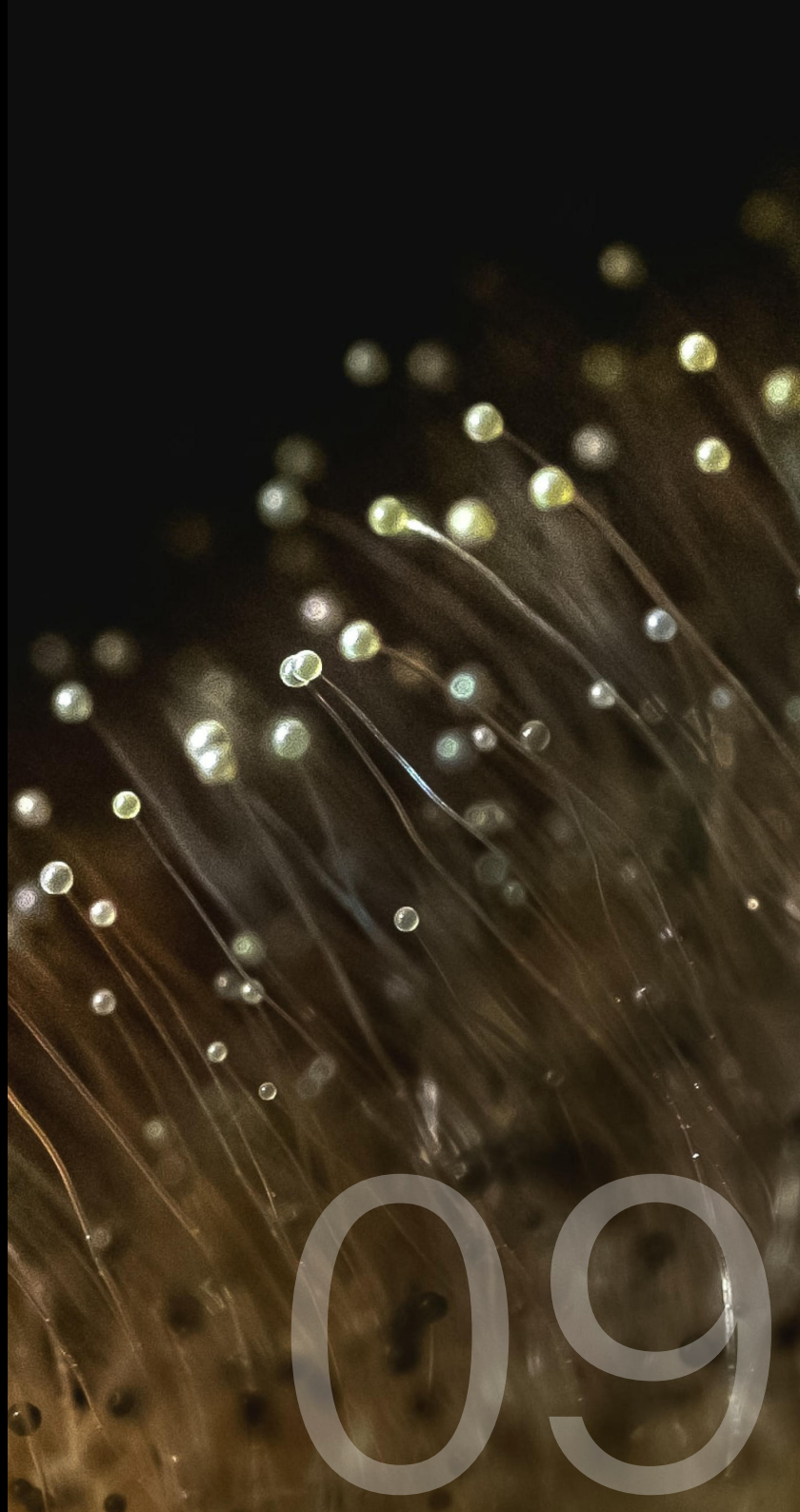
- a completed model from an earlier module
- pen, paper & dice for text generation
- grid paper for a new model

Your goal

To generate synthetic text using your model, then train a new “generation 2” model on that synthetic output. Compare the two models to observe what patterns are preserved or lost. **Stretch goal**: train a generation 3 model on generation 2 output. Or go “full Joker”.

Key idea

Models trained on synthetic data (output from other models) can drift from the original patterns. This demonstrates model collapse and the importance of real training data.



Algorithm

1. **generate synthetic text**:
   - use your existing model to generate text (as in Basic Inference)
   - generate enough text for meaningful training (at least 50-100 words)
   - this is your *synthetic training corpus*
2. **train generation 2 model**:
   - create a new grid following the Basic Training algorithm
   - use your synthetic text as the input corpus
   - this new model learns from AI-generated text, not human-written text
3. **compare the models**:
   - look for words that appear in the original but not in generation 2
   - compare the relative frequencies (counts) in cells that appear in both
   - generate text from both models and compare the outputs

Example

Original training text: *"See Spot run. See Spot jump."*

Generation 1 model's synthetic output: *"See run. Run spot. Spot run run."*

Notice how the synthetic text:
- uses all the same words as the original
- has different patterns (more `run` `run`, no `spot` `jump`)
- might lose some variety from the original

Generation 2 model trained on the synthetic output will amplify these changes:
- `run` `run` becomes more common
- `spot` `jump` disappears entirely
- new unlikely patterns may emerge

Joker mode

Instead of generating synthetic text from an existing model, create a completely random model:

1. draw a grid with any words you choose in the rows and columns
2. add tally marks in any cells you want, with any frequencies
3. this creates a model with no connection to real text patterns
4. generate text from this random grid using dice as normal
5. train a generation 2 model on the output from your random grid

Example

A completely random Joker grid might look like this:

	`pizza`	`robot`	`moon`	`dance`
`pizza`	3		1	2
`robot`	1	4		1
`moon`		2	1	3
`dance`	2	1	2	