Module 3

Simulating Attention in Memory

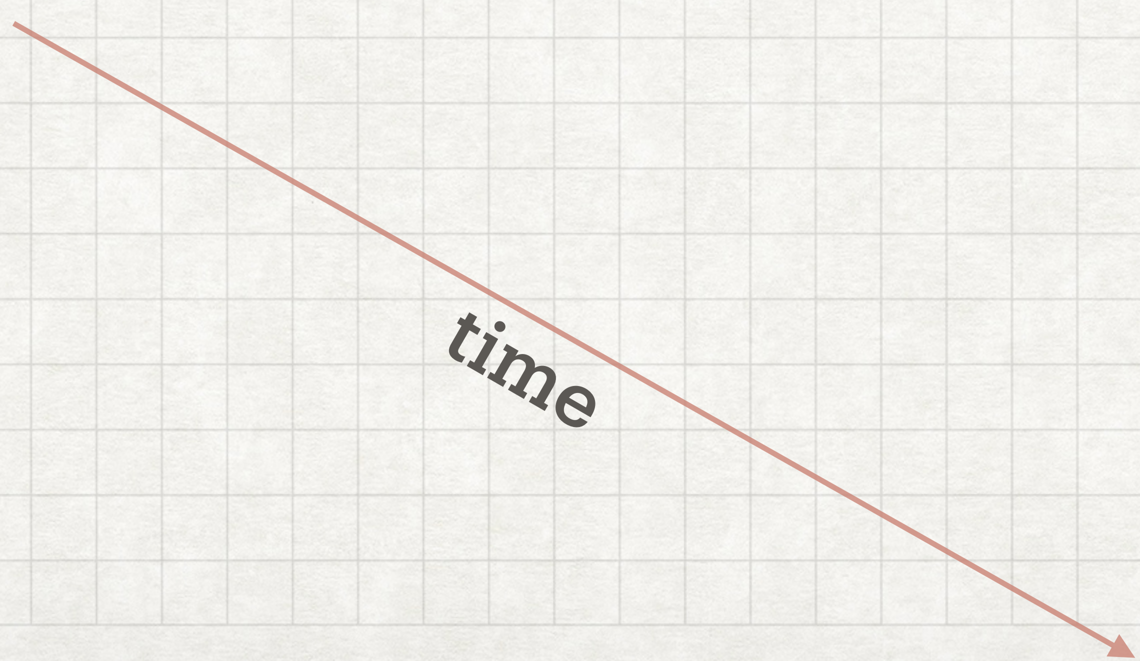
&

Learning Dot-product attention

Mini-Project 3: Attention-based retrieval from memory

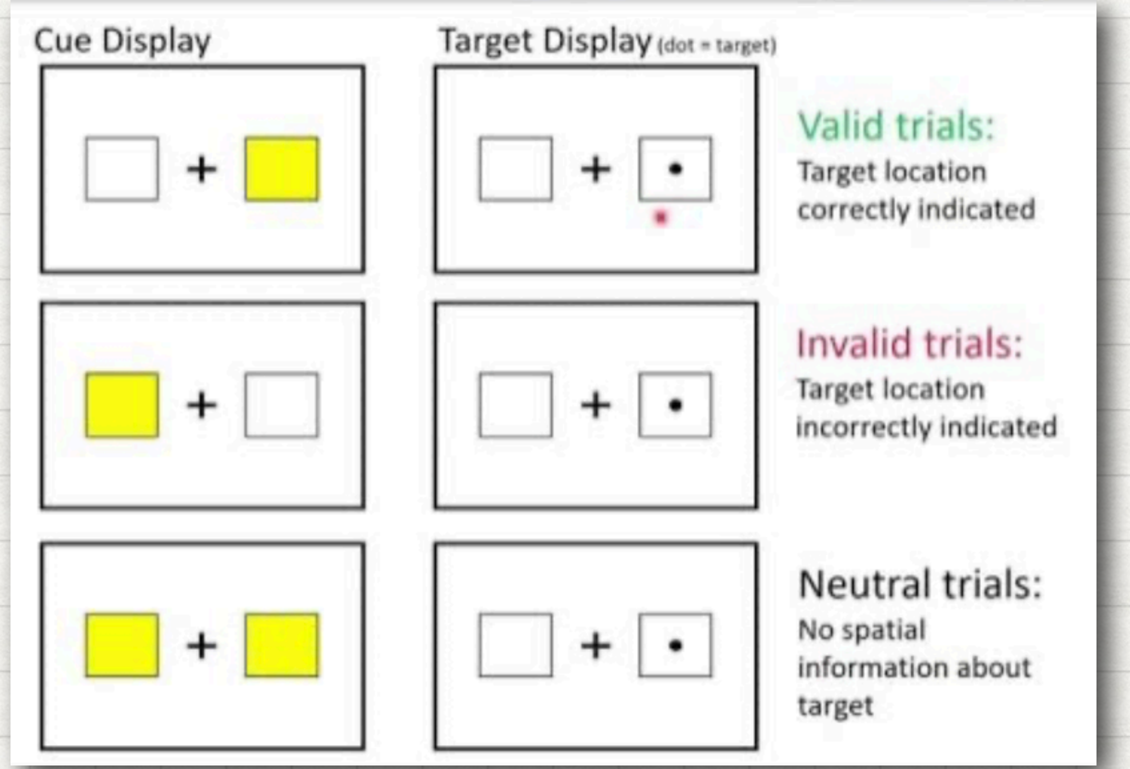
Recall a word from memory based on 'cue'

cue: valid / invalid



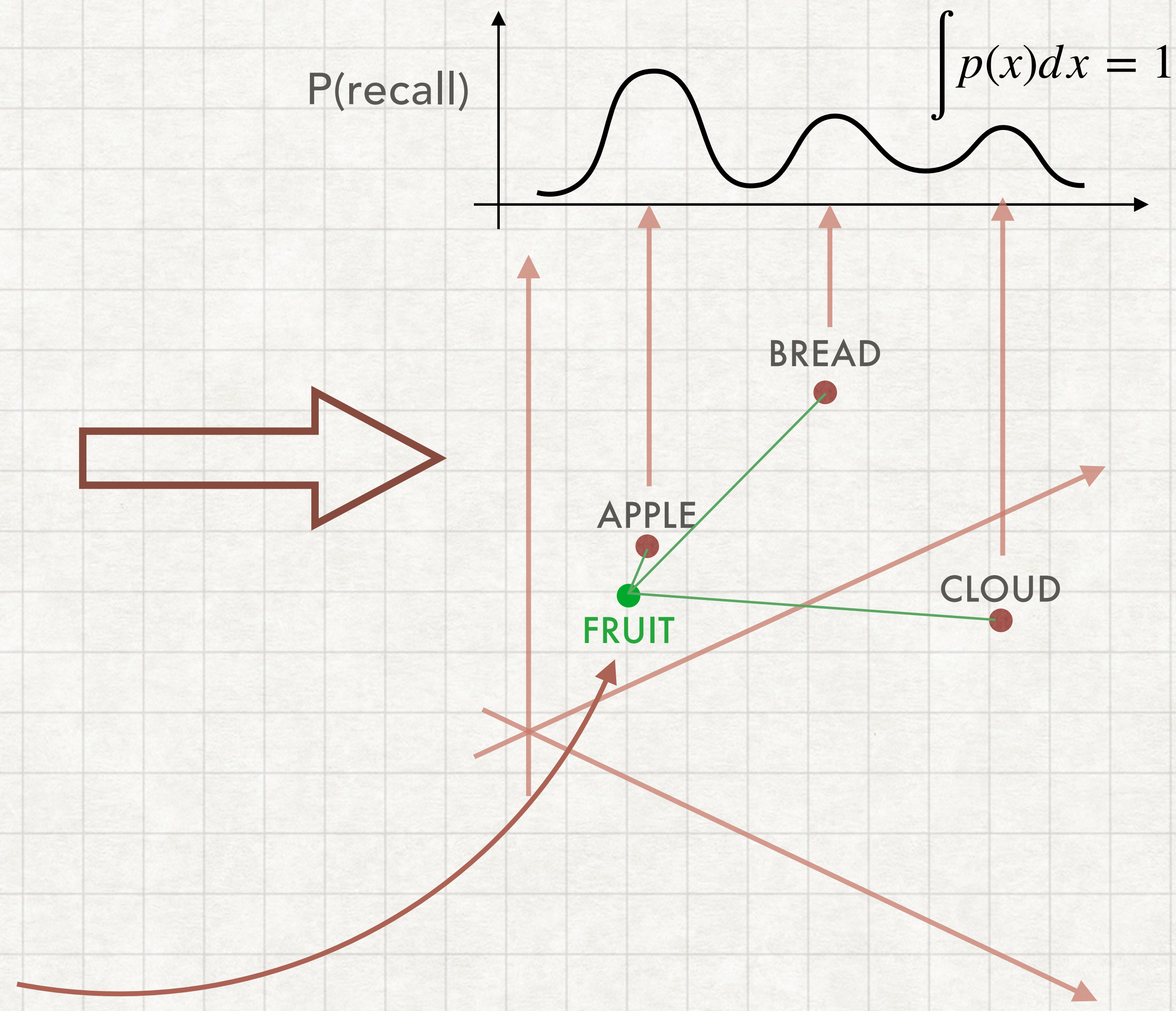
Mini-Project 3: Attention-based retrieval from memory

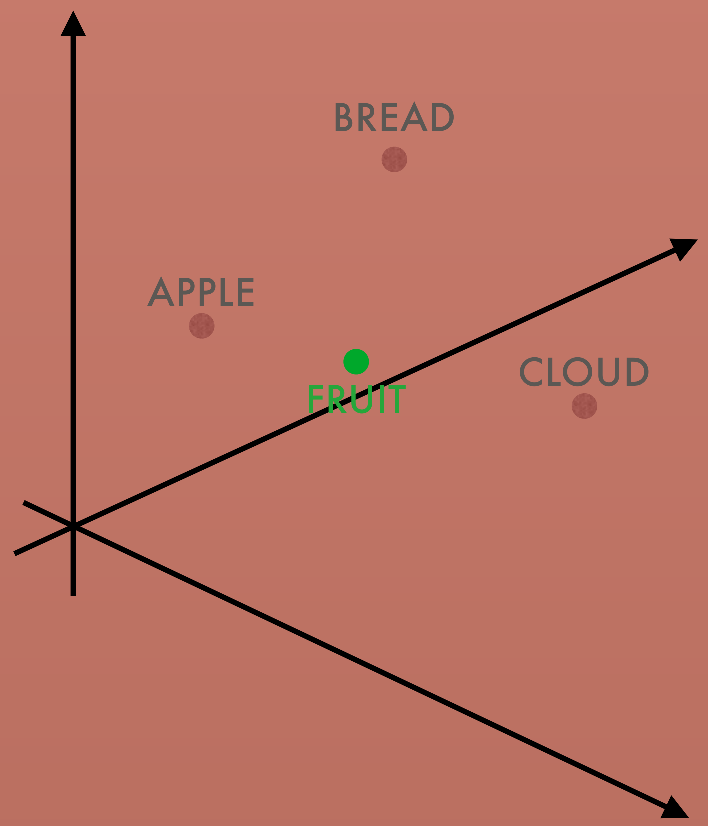
* Recall depends on *match* between cue & memory (Tulving & Thomson, 1973)
* Words are encoded in a semantic context in memory
* Attention can be diverted to target (valid) or away from target (invalid)… Posner cueing paradigm



Mini-Project 3: Attention-based retrieval from memory

How can we simulate this memory process?

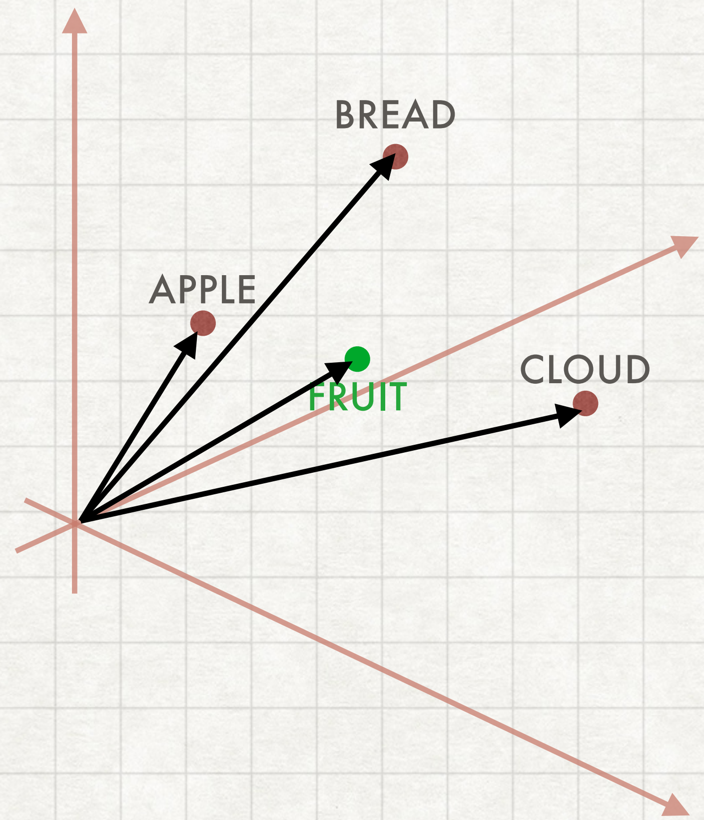


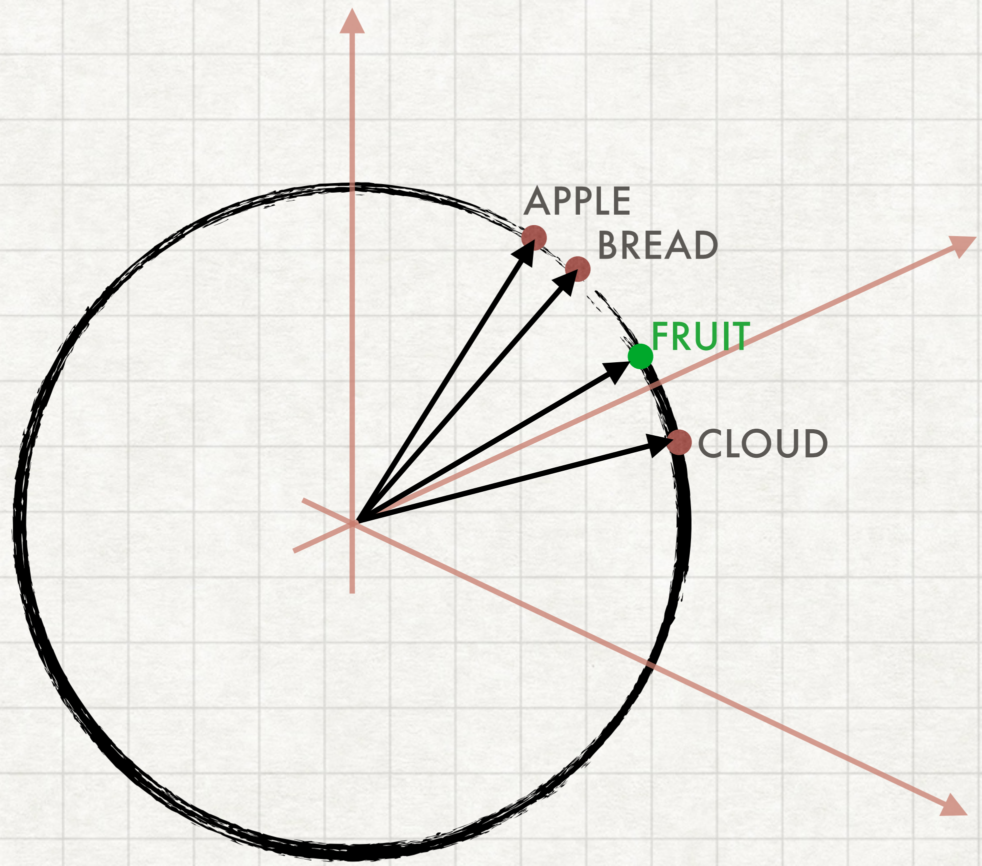


Step 1: Encode words in a high-dimensional space

Note: Similarity using cosine similarity measure

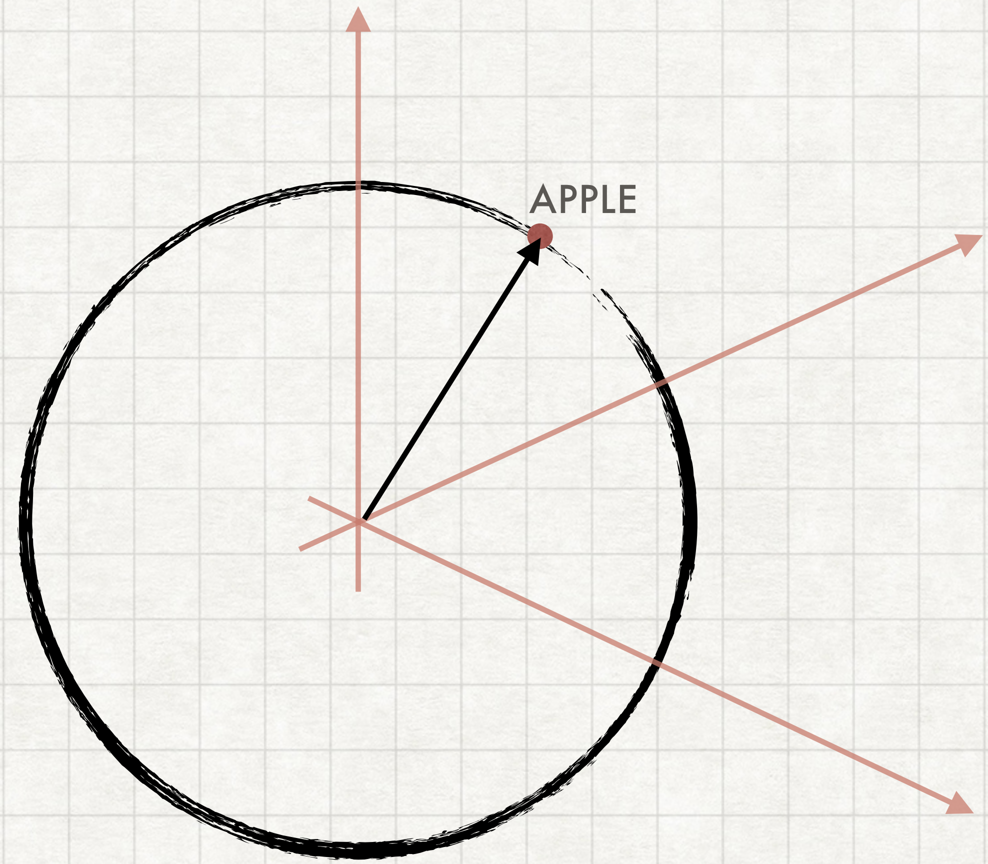
Key part of “dot-product attention”





Demo 1: Encode words in high-dimensional space

APPLE →



We are going to write Python code to convert:

APPLE →

$n\text{-}dim$

Demo 1: Encode words in high-dimensional space

Write Python code to:

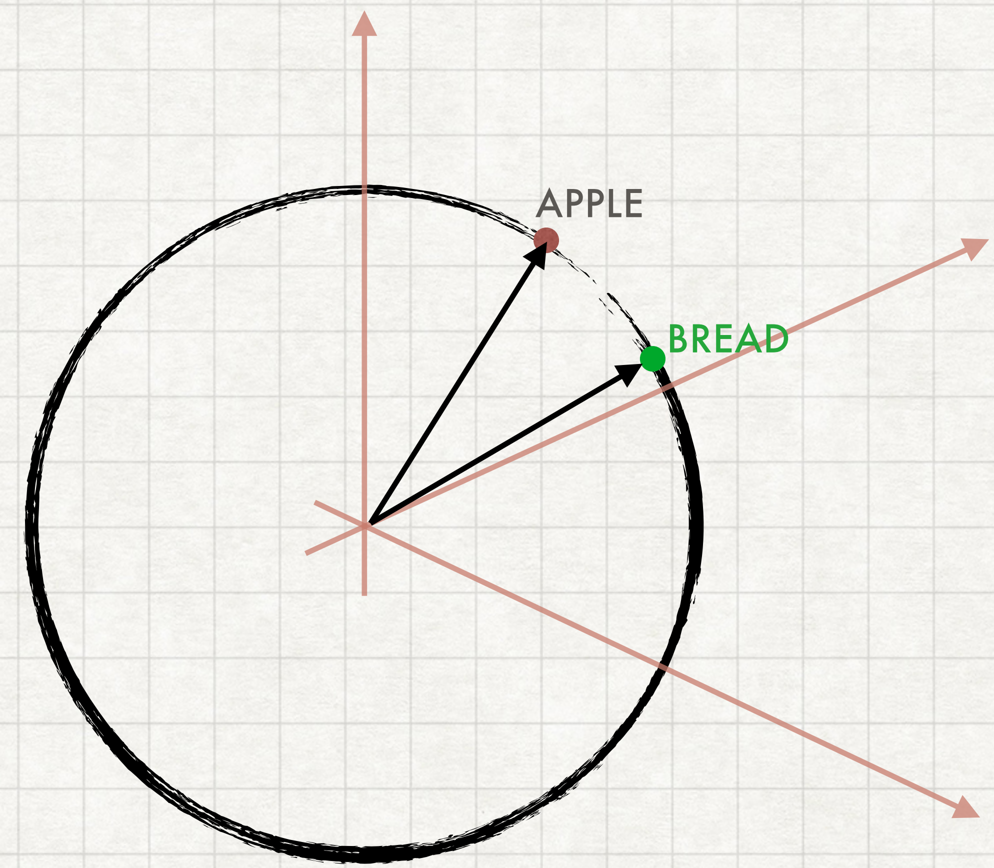
* Create a ‘WordEncoder’ class. This class should:
  * Contain an attribute called ‘dim’ (the dimension of vector space)
  * Contain a method call encode()
  * This method should take a string as input and convert it into a random vector in an n-dimensional space
  * This random vector should lie on the unit circle

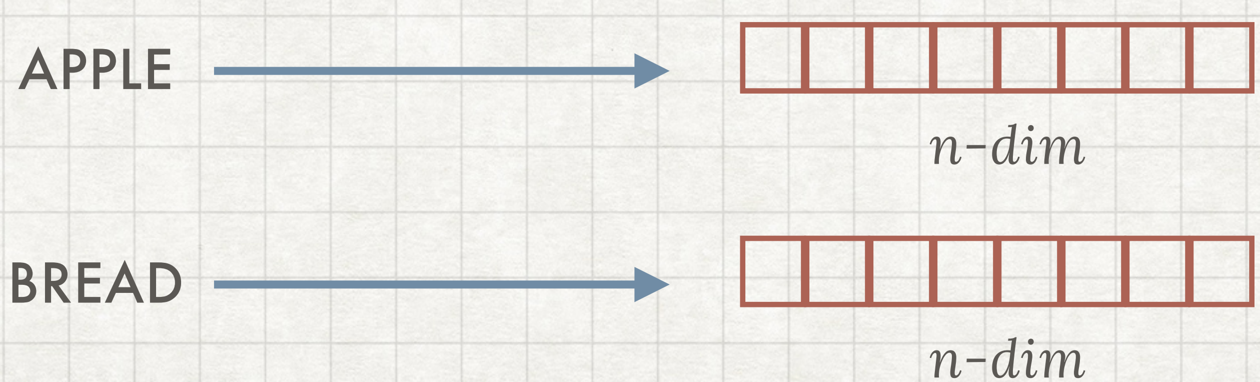
Demo 2: Find similarity between representations

Write Python code to:

* Define a function class called 'Similarity', with functions cosine() & euclidean() that calculate the cosine & euclidean distances

* Use one of these functions based on the initialisation of Similarity instance.



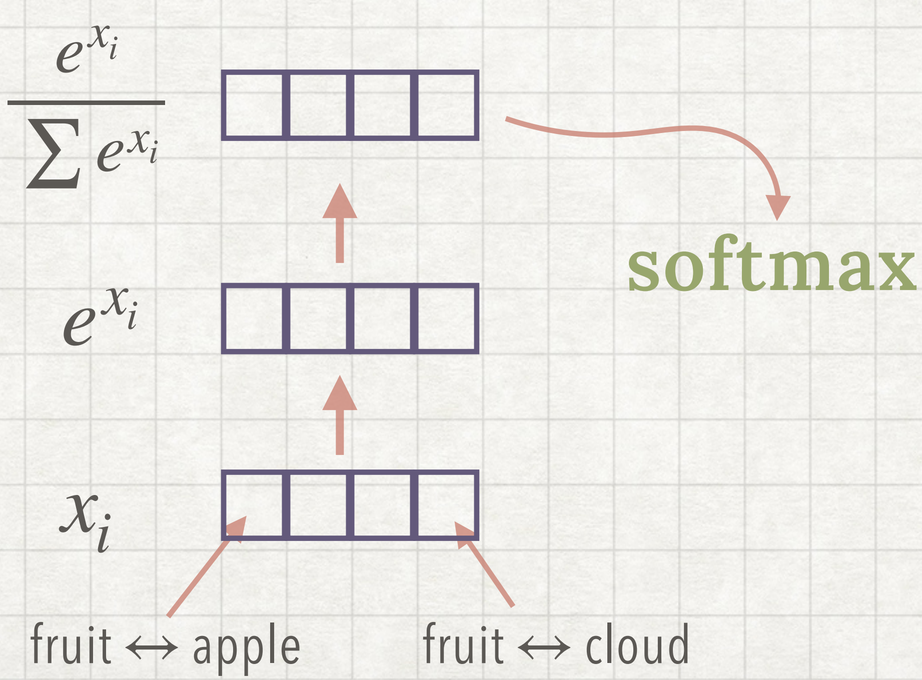


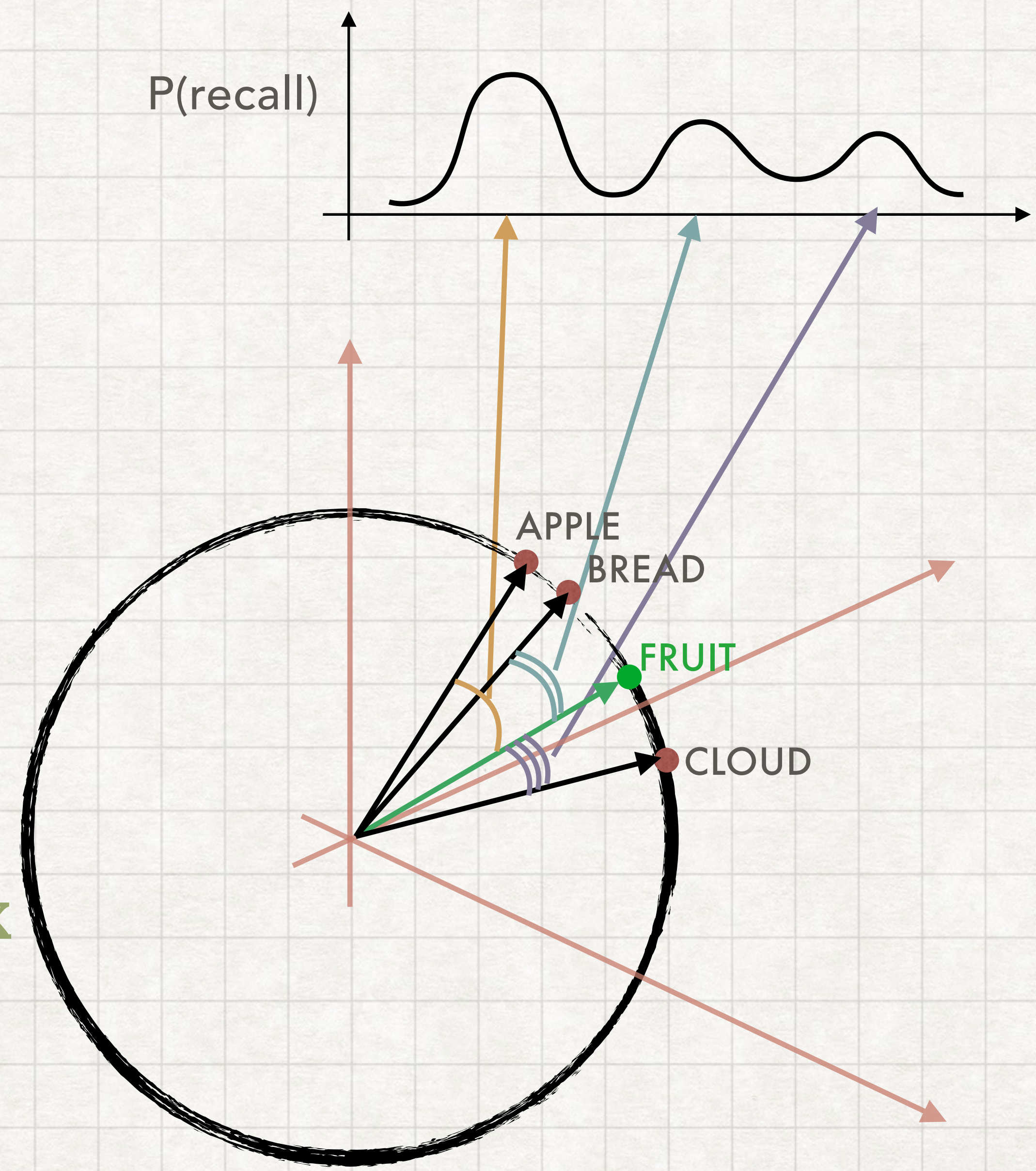
$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$$

Demo 3: Scores to probabilities

Write Python code to:

* Create a numpy array called scores, with distances to each stored memory
* Convert these scores to a probability distribution over memory traces



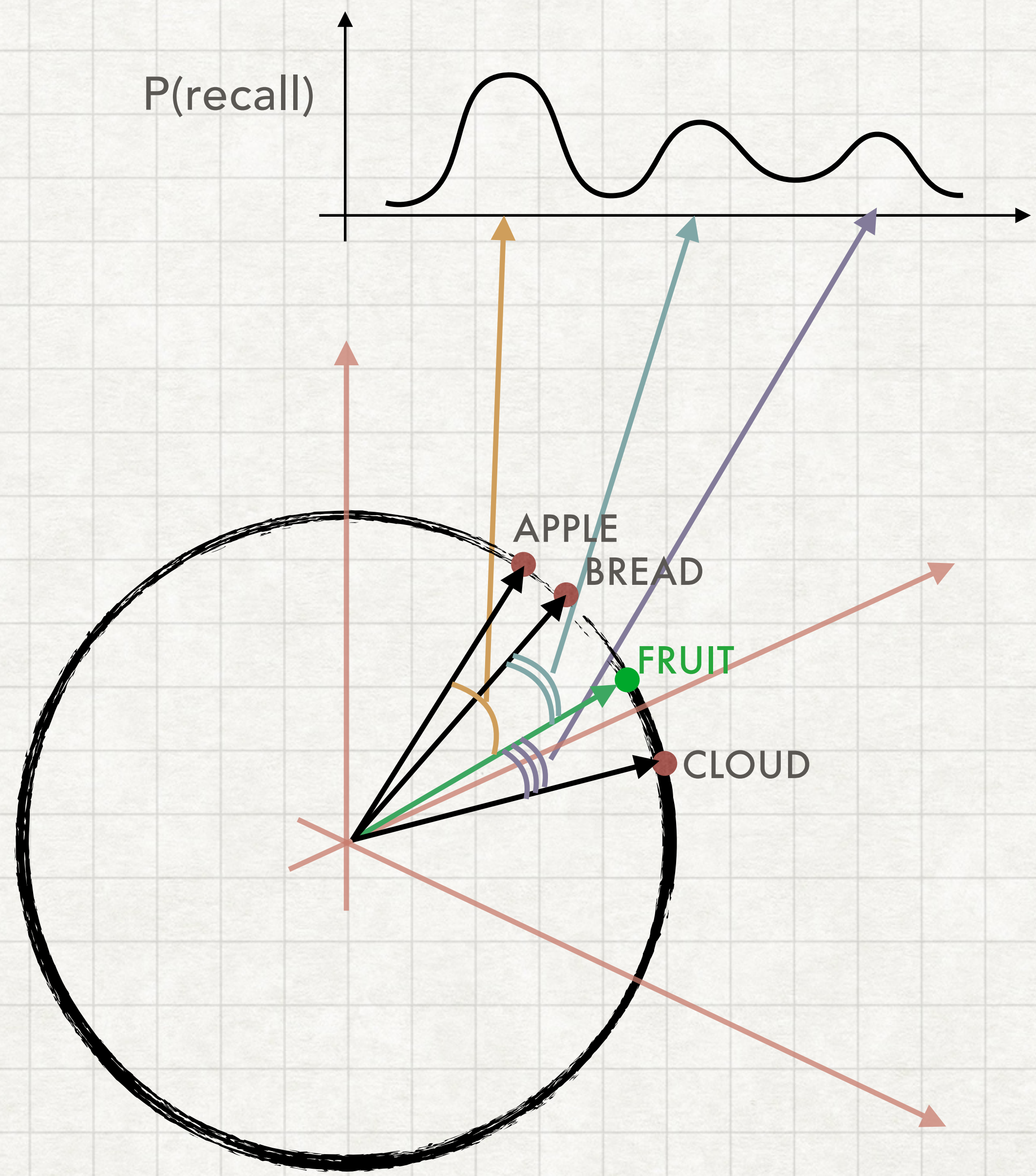


Demo 3: Pseudocode

```
FUNCTION softmax(scores):
    # 1) exponentiate to amplify differences
    e_i ← exp(z_i) for each z_i

    # 2) normalize to make probabilities
    total ← sum(e_i)
    w_i ← e_i / total for each i

    RETURN [w_1, ..., w_n]
```



Demo 4: Scale the attention based on temperature

Softmax function allows temperature scaling - that is how sharply focused the attention is. When temperature is high, attention is “diffused”, while when temperature is low, attention is highly focused (winner-take-all).

$$\frac{e^{x_i}}{\sum e^{x_i}} \longrightarrow \frac{e^{\frac{x_i}{\tau}}}{\sum e^{\frac{x_i}{\tau}}}$$

* Write Python code to change the softmax function so that the probabilities are computed based on temperature.

* Test the function for different values of temperature parameter, but same scores

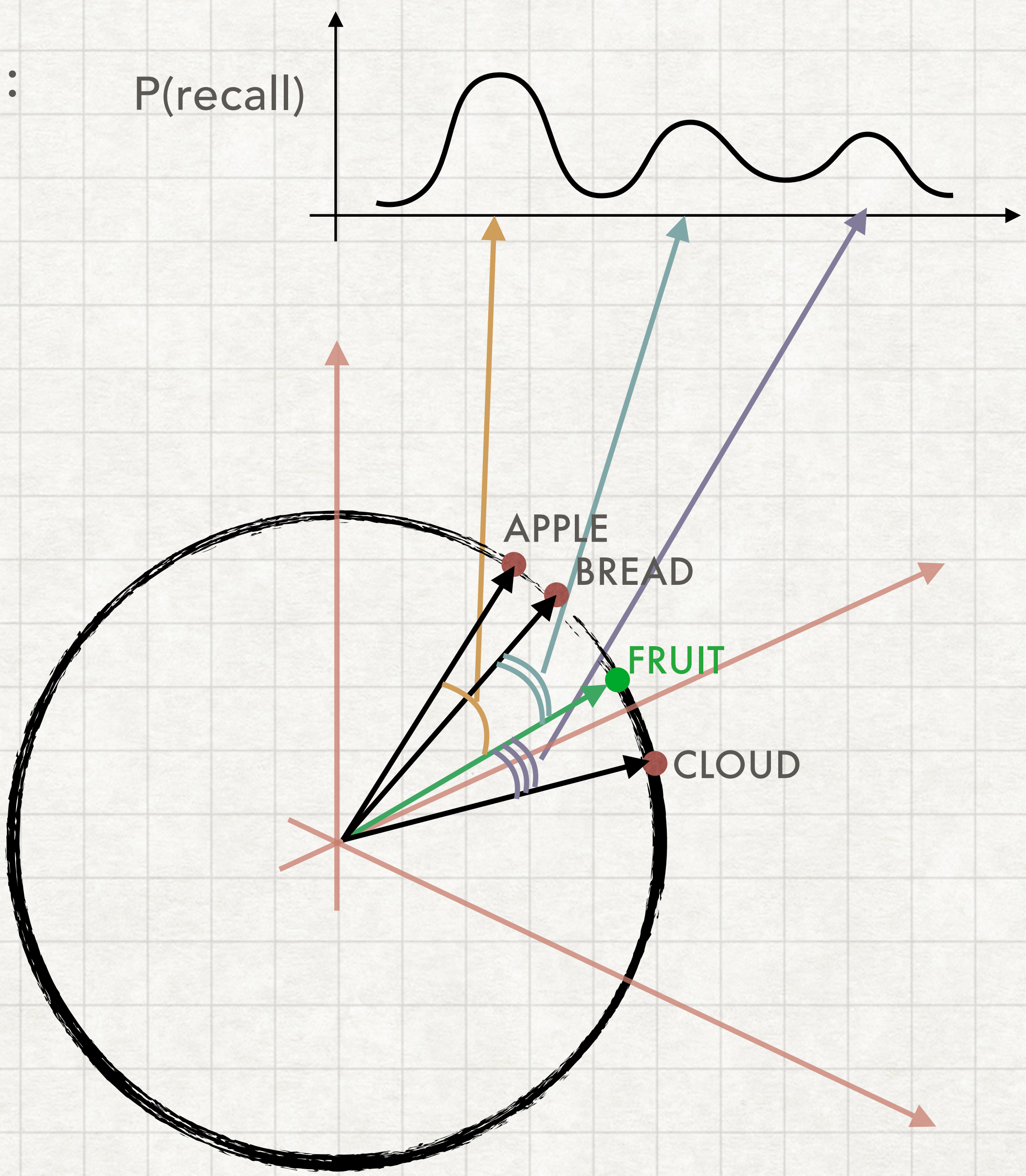
Demo 4: Pseudocode

```
FUNCTION softmax(scores, temperature τ):
    # 1) temperature scaling
    for each s in scores:
        z_i ← s / τ

    # 2) exponentiate to amplify differences
    e_i ← exp(z_i) for each z_i

    # 3) normalize to make probabilities
    total ← sum(e_i)
    w_i ← e_i / total for each i

    RETURN [w_1, ..., w_n]
```



Demo 5: Compute Context vector via Dot-product Attention

Cue → Query

Memory index → Key

Memory rep → Value

Let us put it all together and parallelise the attention mechanism.

Write a class called DotProductAttention with a function attend() that

1. computes the scores for all memories in parallel
2. Computes the attention weights for each memory
3. Computes a "context" based on aggregated memory recalled
4. Returns this context and attention weights

Test this function for memories: ['apple', 'bread', 'cloud', 'drum', 'eagle'] & query 'apple'

Demo 5: Compute Context vector via Dot-product Attention

Cue → Query

Memory index → Key

Memory rep → Value

```
CLASS DotProductAttention(temperature τ):
    METHOD attend(query q, keys K, values V):
        d ← dimension of q

        # 1) compute similarity scores (dot products)
        for each key k_i in K:
            score_i ← q · k_i

        # 2) convert scores to attention weights via softmax
        w ← softmax([score_1, ..., score_n], τ)

        # 3) compute context vector as weighted sum of values
        c ← Σ_i (w_i * v_i)

        RETURN w, c
```

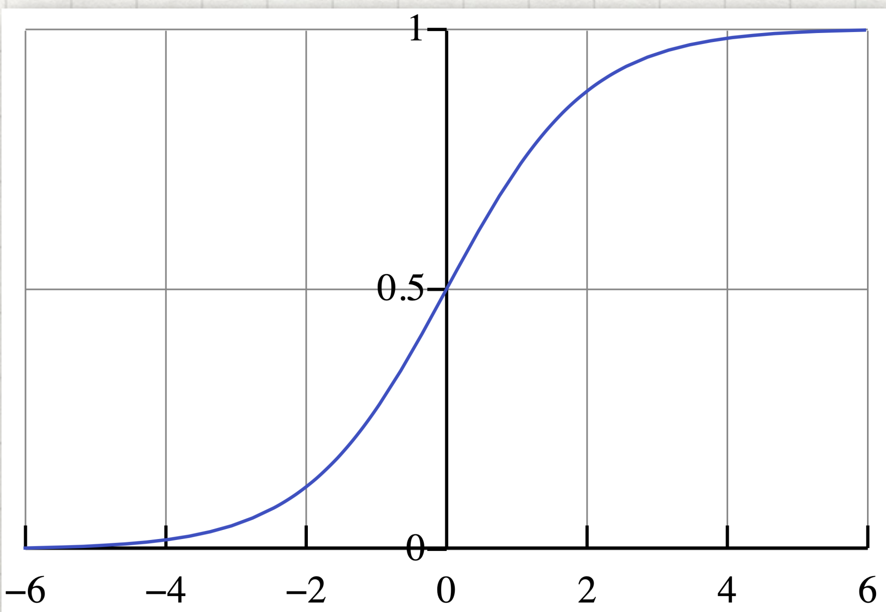
Demo 6: Convert context into recall probability

Once you have retrieved a context vector, you can work out whether this context will lead to a successful recall. To do this:

1. You could calculate the similarity between the context and the cue (we know how to do this, right!).

2. Convert this similarity into a probability. We've previously converted a set of scores into probability distribution, but now we have just one similarity — so softmax is not the right function. Instead, we can use the **logistic function**. Cosine function varies between [-1, 1] and logistic function varies between [0,1], which is perfect for computing probability.

3. Logistic function has two parameters
   1. **Gain**: Slope of function
   2. **Bias**: location of mid-point
   3. You can play around with different values



Demo 6: Convert context into recall probability

Pseudocode

```
FUNCTION logistic(x; g, b):
   RETURN 1 / (1 + exp(-(g*x + b)))

PROCEDURE context_to_recall(context c, target t, gain g, bias b):
   # 1) compute similarity between retrieved context and true target
   sim ← cosine_similarity(c, t)

   # 2) turn similarity into probability (smooth S-shaped mapping)
   p_recall ← logistic(sim; g, b)

   # 3) (optional) simulate an observed outcome
   recalled ← random_uniform(0,1) < p_recall

   RETURN p_recall, recalled
```

<u>Mini-projects</u>

Project 1

Goal: Add *semantic similarity* between words so cues can partially match non-identical targets.

So, if cue = ‘Fruit’ and memories = [‘Bread’, ‘Cloud’, ‘Apple’, ‘Horse’] then recall probability should be higher than when cue = ‘Tractor’

Hint: You will need to change your ‘encode’ function, so that the encodings are not random, but depend on semantic context. You can use pre-trained embeddings (look into ‘GloVe’ and ‘spaCy’)

Deliverable: Demonstration and a bar plot of *semantic distance* vs *recall probability*

Mini-projects

Project 2

Goal: Add a time dimension, so that recall declines between study & test.

Hint: Introduce a forgetting parameter that changes memories at time or recall by adding Gaussian noise to memories based on the parameter value

Deliverable: Demonstration and a bar plot of *time of recall* vs *recall probability* over a set of items.