**SEQUENCE MODEL**

**\*Sequence model example-**

1.speech recognition — input x and output y both in sequence

2.music generation — only y in sequence

3.sentiment classification — sequence input

4.DNA sequence analysis

**\*Notation**

Lets create a sequence model to find the location of persons name in a sentence

Problem here is- named entity recognition

Its mainly used by search engines used to find persons name, location, currency name etc.

The input sentence is represented as X

For ex-

X: “harry potter invented a new spell “

The output Y generated by model is;

1 1 0 0 0 0

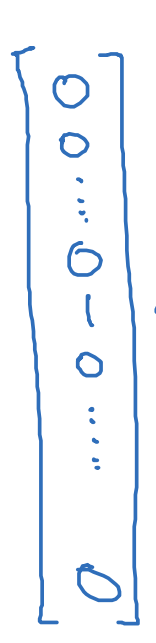
Which tells us whether the input sentence has persons name or not

The t’th element of training example is represented as- X(I)<t>

Input sequence length for training example I - TX(i)

How to represent different words in the sequence?

First we will have to define the dictionary (list of words)

Then use one hot vector to represent them .

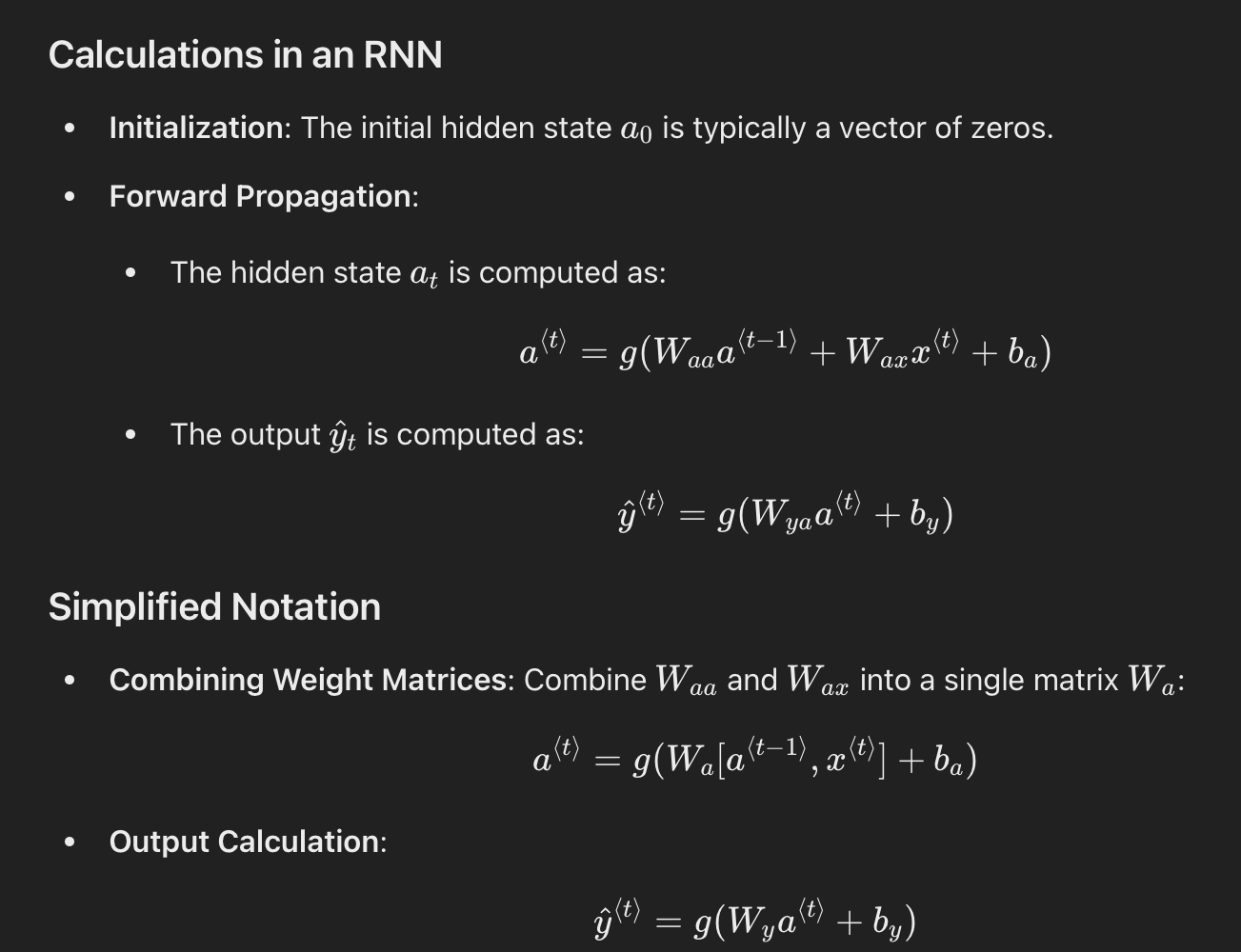
**Recurrent neural network model**

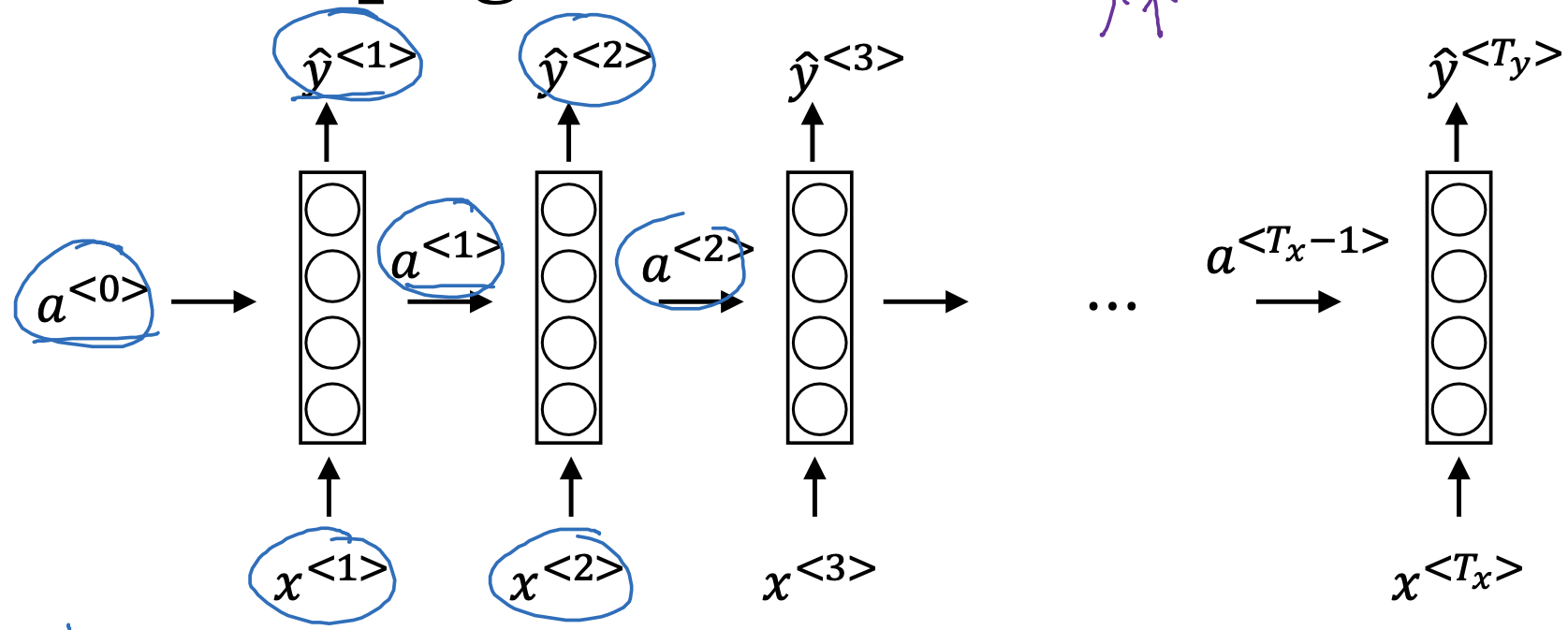
Problem with standard network -

Inputs, outputs can be different lengths in different examples.

Doesn’t share features learned across different positions of text.

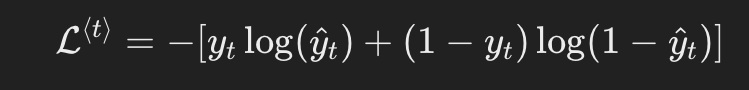
RNN processes the sequence one element at a time by maintaining the hidden state that carries information of previous element .

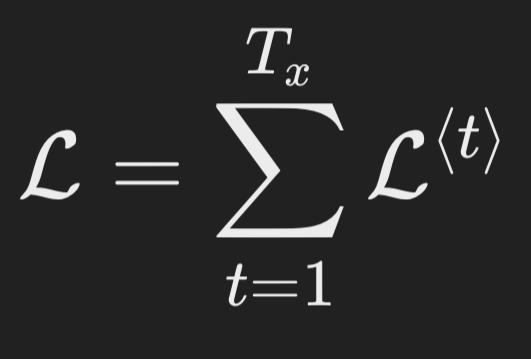
**Calculations-**

Network diagram ex-

**\*Backward propagation**

We define loss function first-

We use cross entropy loss function here-



Overalll loss-

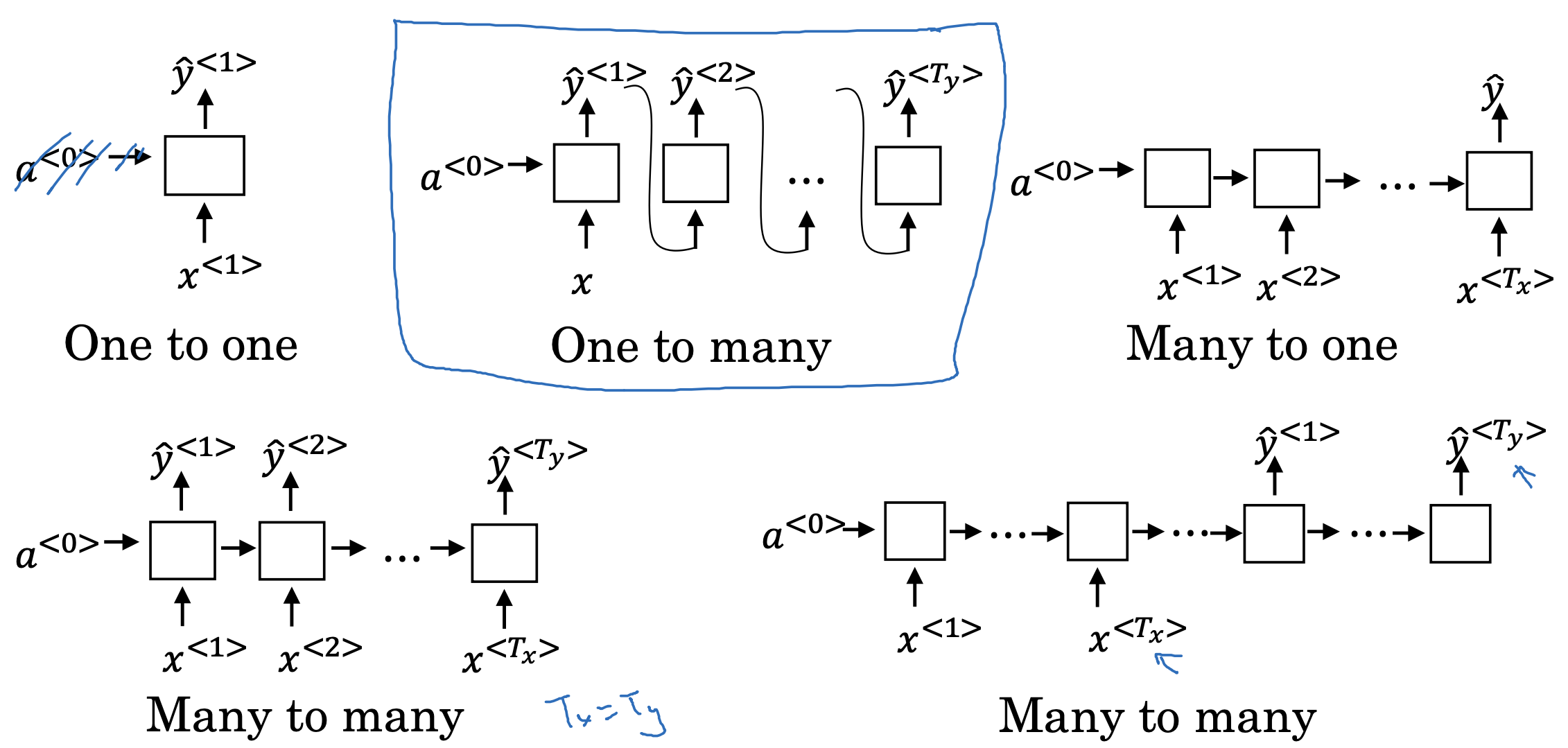
Backprop through time

1.perform backprop right to left (opposite to direction of forward prop)

2.compute gradients for each step and gain information backward through time

3. Using gradient descent update the parameters

Wa ,ba ,Wy ,by

**\*Example of RNN architecture**

**\*Language model and sequence generation**

**Language model-**

It estimates the probability of a sentence

Used in speech recognition

For ex-

1.The apple and the pair salad

2.The apple and the pear salad

The probably of 1 is 3 times that of 2

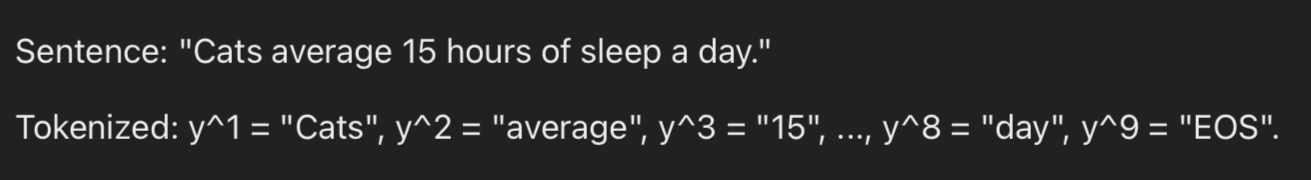
Buiding a language model using RNN

Training set - large corpus of words in English

Now we tokenise the sentence into tokens

And then these tokens into one hot vectors

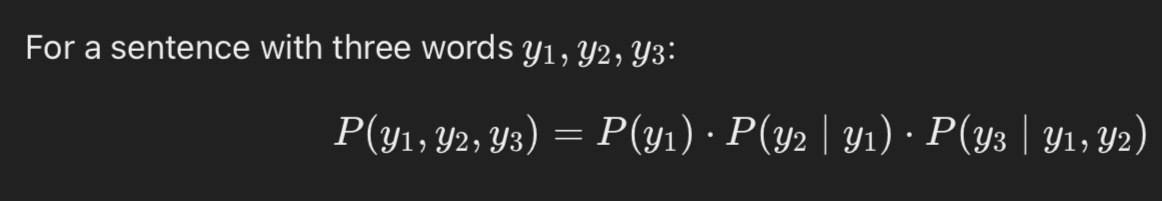
EOS (end of sentence ) can also be included

For Ex-

Now train model

Take input x<t> that gives output y<t-1>

Compute activation ,loss function and prediction

Ex- of model prediction

**Sampling novel sequence**

**Sampling process-**

Initialize x(1) and a(0) as 0 vector

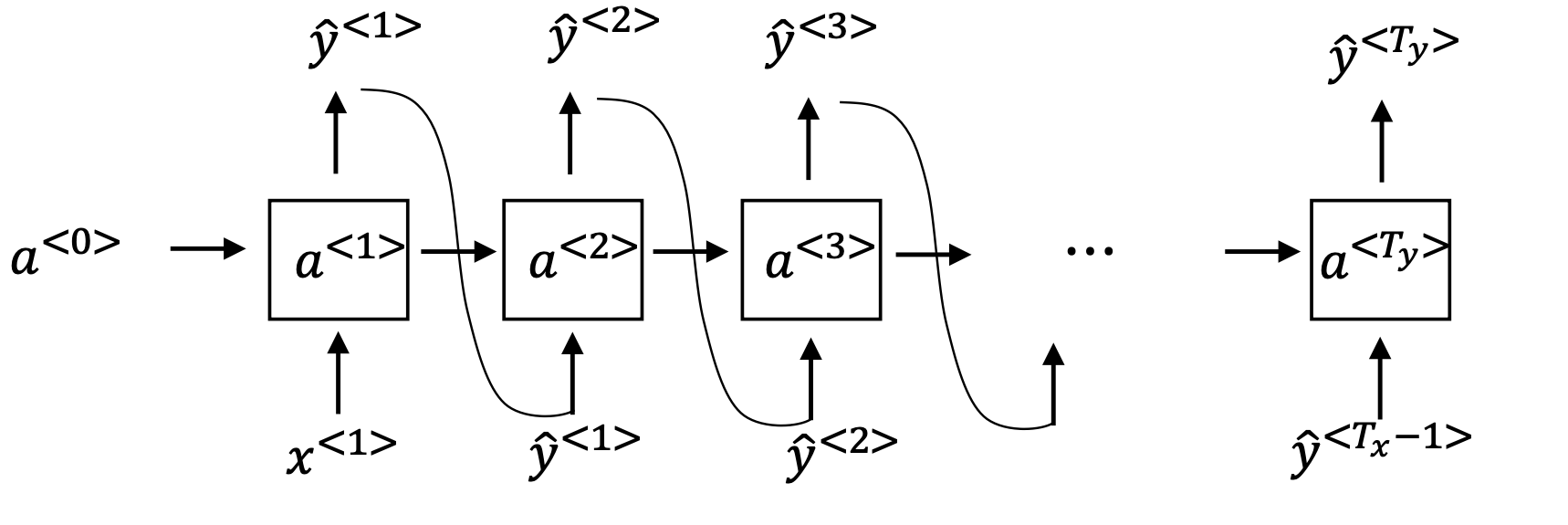
Now compute softmax probability of first word

Sample this first word y(1) from the distribution

For each step use subsequent sampled out word as input

Continue above until EOS is reached

Resample if unknown word is generated



**Word-Level:**

Pros: Captures sentence structure, easier to train.

Cons: Needs handling for unknown words.

**Character-Level:**

Pros: Avoids unknown words, can create novel words.

Cons: Longer sequences, more computationally intensive.

Ex-

Word-Level: "Cats average 15 hours of sleep a day."

Character-Level: "C-a-t-s- -a-v-e-r-a-g-e- -1-5- -h-o-u-r-s- -o-f- -s-l-e-e-p- -a- -d-a-y.”

After training we can sample sequences to generate new text.

**\*vanishing gradient**

RNN struggle to capture information much earlier in the text which can be useful in the later part of sentence

Now gradients diminish as they are propagated backward in the deep layers

Making it much harder for earlier information to affect the later one

Now gradients an explode due to exponential growth during back prop

To prevent above issue we use gradient clipping to set a threshold value to prevent

gradients from becoming larger.

Advanced models like GRUs and LSTMs can also be used.

\***GRUs**

GRUs: Enhanced RNNs designed to better capture long-term dependencies and

mitigate vanishing gradient problems.

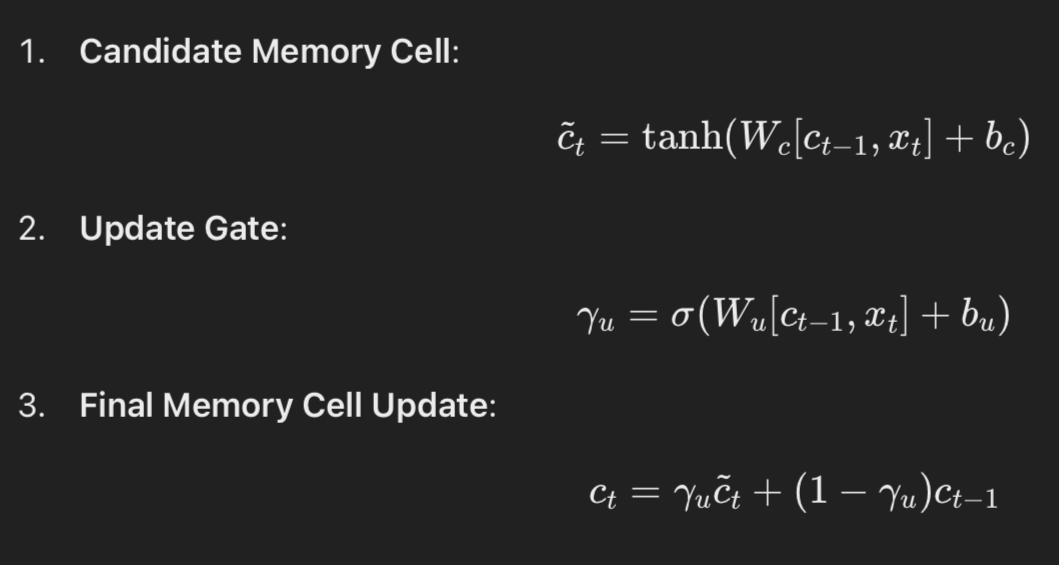
**Key Components of GRUs**

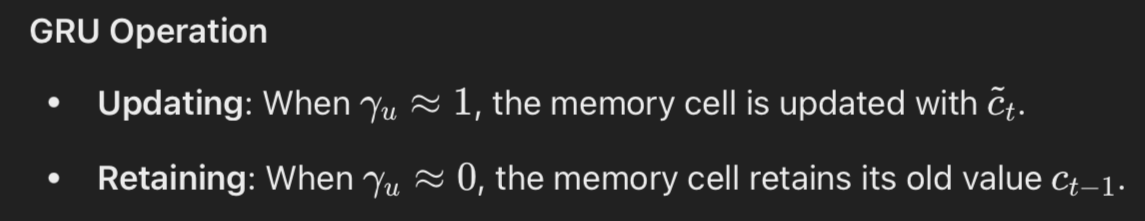
Memory Cell (c(t))

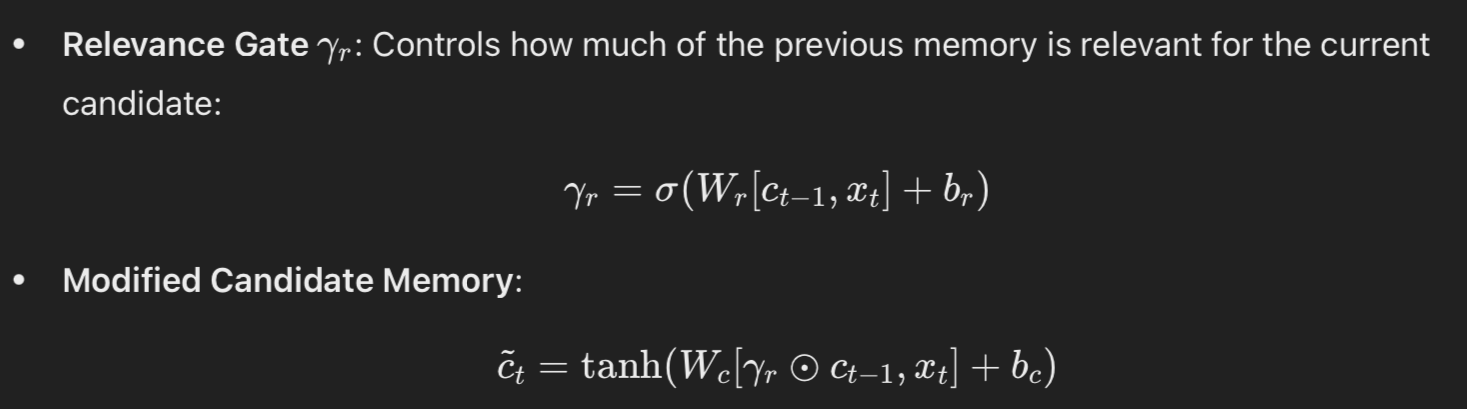
​ : Stores information that needs to be remembered across time steps.

Update Gate (γ(u))

​: Determines whether to update the memory cell with new information or retain the old value.

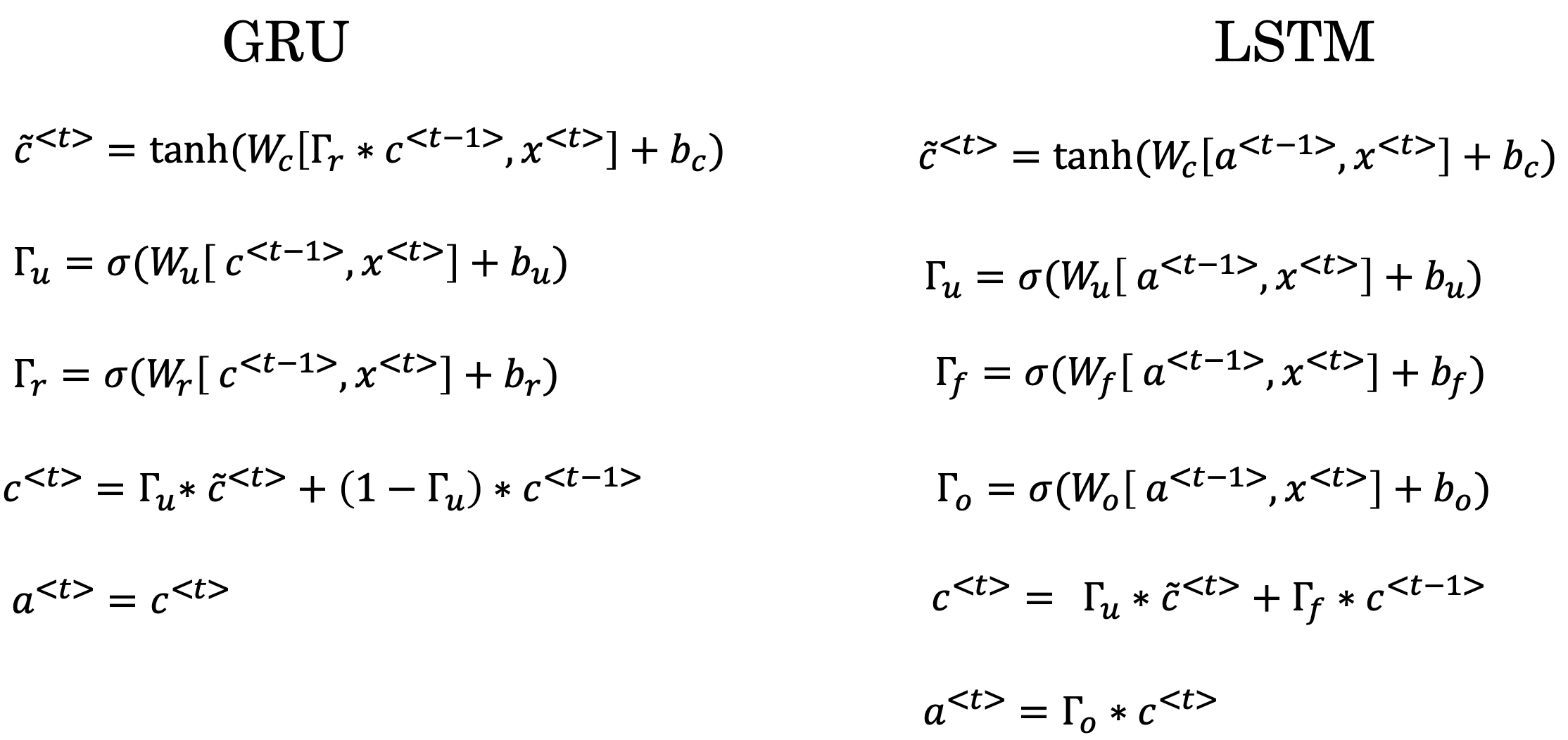
**Equations-**

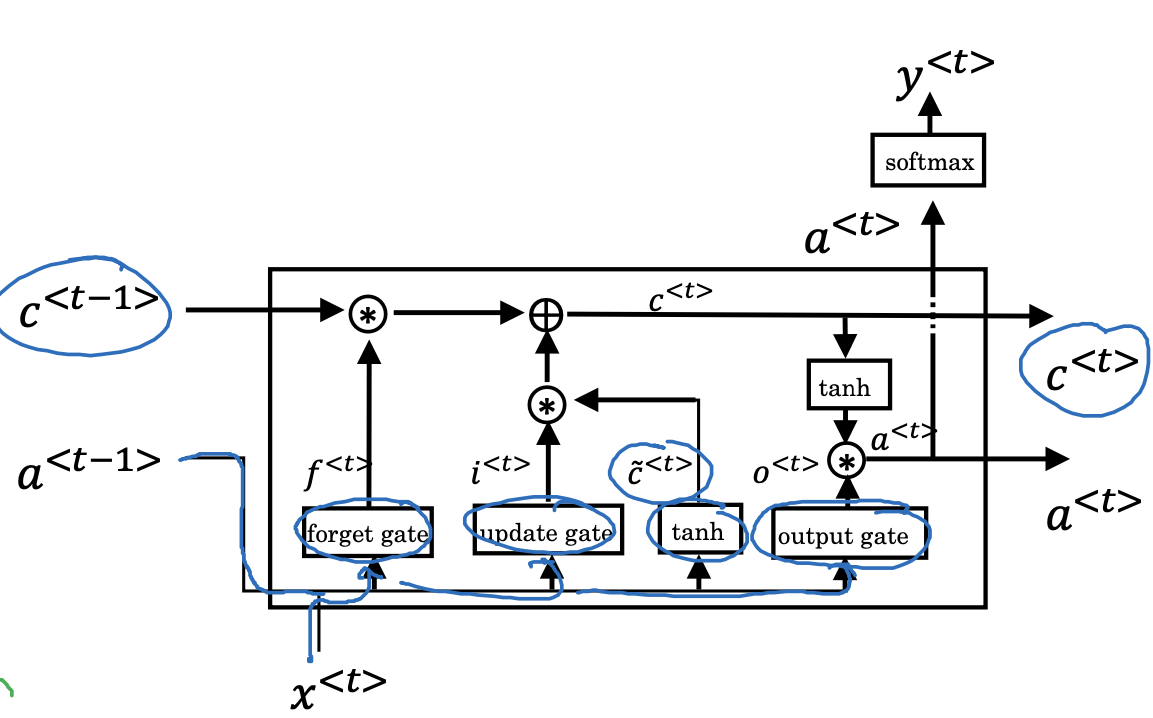


**Full GRU units**

**\*LSTM**

Its more powerful than GRU

Equation comparison between them -

**Diagram-**

a(t-1) and x(t) —- inputs

Gates-forget,update,output

Their is combination of gates and memory cells

Variation of gates depends on previous memory cell value (c(t-1))

**LSTM vs GRU**

LSTM-

Has more gates for better control and flexibility

More proven choice

GRU-

Only 2 gates so simple

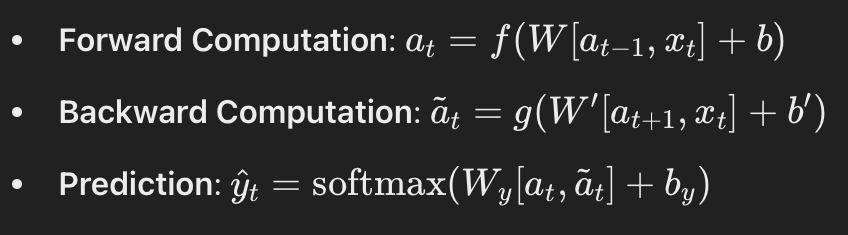
Much useful for high speed computation

**\*Bidirectional RNN**

It solves the problem of unidirectional RNN as it captures both past and future information

Forward RNN- processes the sequence from start to end

Backward RNN- processes the sequence from end to start

**Computational flow-**

It is widely used in NLP for maned entity recognition.

Limitation - it requires full sequence input to make prediction so not useful for speech recognition (real time)

**\*NLP AND WORD EMBEDDINGS**

**\*Word representation**

As far now we use vocabulary to represent words and the encode them into one hot vectors

The weakness of it is it treats each word as thing unto itself and doesn’t allow algorithm to easily generalise cross words.

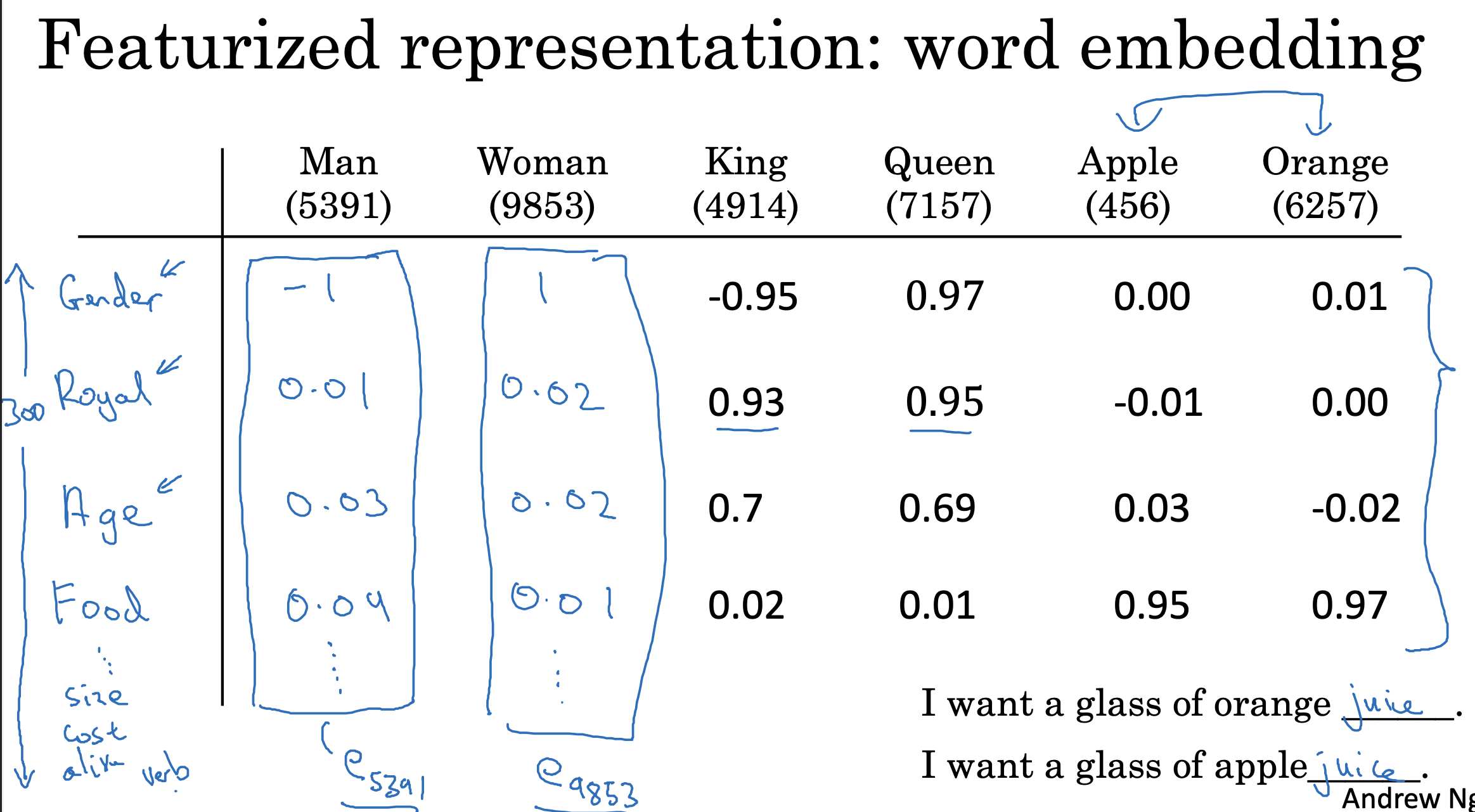
For ex-

I want a glass of orange juice

Even I the algorithm has learned the above sentence it cannot complete the sentence if it for for apple as it does not see any relationship between apple and orange.

So , we use featureized representation

We learn the set of features for each words



So above we come up with a lot of features

Lets say 300 for each column

In the above figure e represents 300 dimensional vector

Now algorithm will see that apple and orange are very similar based on their feature and will complete the sentence for apple too.

Another representation- t-SNE is used to convert 300D to 2D

**\*Named entity recognition example**

To detect people name

Ex-

Sally is an orange farmer

One way is to look for sally

Other way is to look of the orange farmer as it is meant to be a person

Now if there Is this- Rohit is a durian cultivator (uncommon )

If we have small level training set for named entity recognition task ,so we haven’t seen the word durian / cultivator in the training set

If we have learned a word embedding that tells you durian is a fruit like orange and cultivator corresponds to farmer . So by seeing orange farmer we know durain cultivator to be a person.

So algorithm do learning the word embeddings to examine very large text courses.

Now we an take that orange and durian embedding to apply it to other data.

To apply it to much smaller data set - 100k words

So this allows to carry out transfer learning

**\*Transfer learning & word embedding**

1. Learn word embeddings from large text corpus.(1-100Bwords) (Or download pre-trained embedding online.)

2. Transfer embedding to new task with smaller training set. (say, 100k words)

3. Optional: Continue to finetune the word embeddings with new data.

**\*Word embeddings properties**

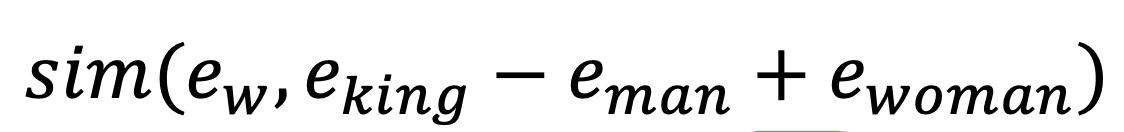
Relationship between words can be represented by vector arithmetic

Vector arithmetic:

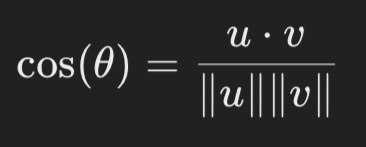
Words can be represented as vectors (e(king),e(queen) etc)

Difference between the vectors capture relationship



We have to find that w that maximises similarity

**\*Cosine similarity**

It measures the similarity between 2 vector u&v.

The value rangers from -1(opposite) to +1(same)

**\*Word2Vec**

**\*skip grams**

The Skip-Gram model predicts context words given a target word.

This differs from traditional models that use the last few words as context to predict the next word.

**Here,**

Randomly pick the context word

Ex- orange

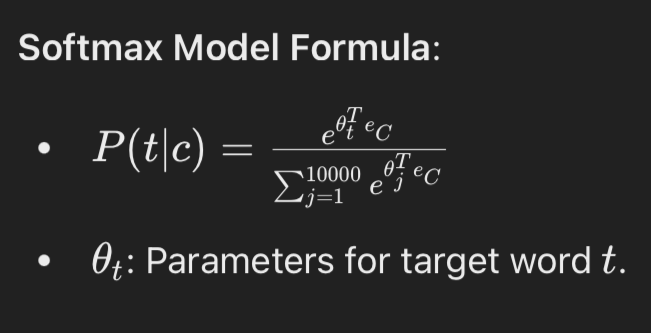
Randomly pick another word withoin context for window

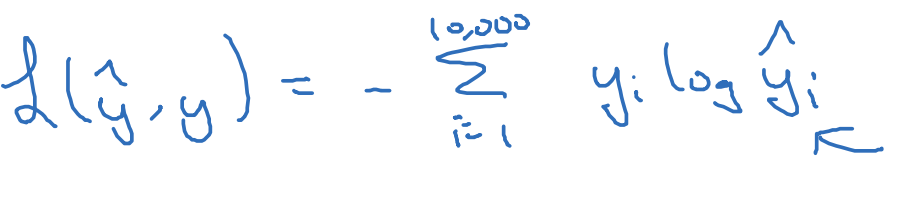
Ex context:orange target:juice

Given a context word, the model predicts a randomly chosen word within the window.

Get the one hot vector for context word and then multiply it by embedding matrix to get the embedding vector e(c)

Now comes the **softmax Layer**



**Loss function :**

The problem with softmax classification -

Requires summing over all vocabulary words, which is slow for large vocabularies.

Hierarchical Softmax:

Use a tree of binary classifiers to reduce computation.

The tree scales logarithmically with vocabulary size (log(vocab size)).

**Uniform Sampling Issue:**

Overrepresentation of common words like "the", "of", "a".

**Balanced Sampling:**

Use heuristics to ensure both common and less common words are adequately represented.

**Comparison b/w CBOW & Skip gram**

Continuous Bag-of-Words (CBOW) Model:

Predicts a word based on its surrounding context words.

Skip-Gram Model:

Takes a single word and predicts its surrounding words, skipping some.

The embedding matrix E and softmax parameters are optimised using gradient descent

**\*Negative sampling**

For further efficiency gain we use this

Predicts whether pairs of words are likely to appear together (context-target pairs).

Dataset Generation:

Positive Ex: Pairs like (orange, juice) where words are within a context window.

Negative Ex: Pairs like (orange, book) where the target word is randomly sampled from the vocabulary.

Train a binary classifier to distinguish between positive and negative pairs.

Model:

Uses logistic regression with a sigmoid function to model the probability of a pair being positive.

Parameters include embeddings for each word and weights for each target word.

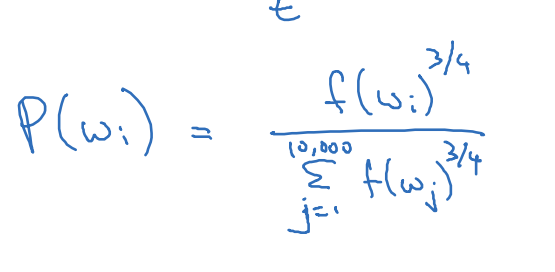
Instead of a single large softmax classification, trains multiple binary classifiers (k+1, where k is the number of negative samples).

Updates parameters using gradient descent based on the binary classification results.

Reduction in computational complexity compared to softmax by transforming the problem into several independent binary classification tasks.

Sampling Strategy:

Employs a heuristic where negative samples are chosen based on a modified frequency distribution (f(w\_i)^3/4), balancing between frequency and uniform sampling.



**\*GloVe**

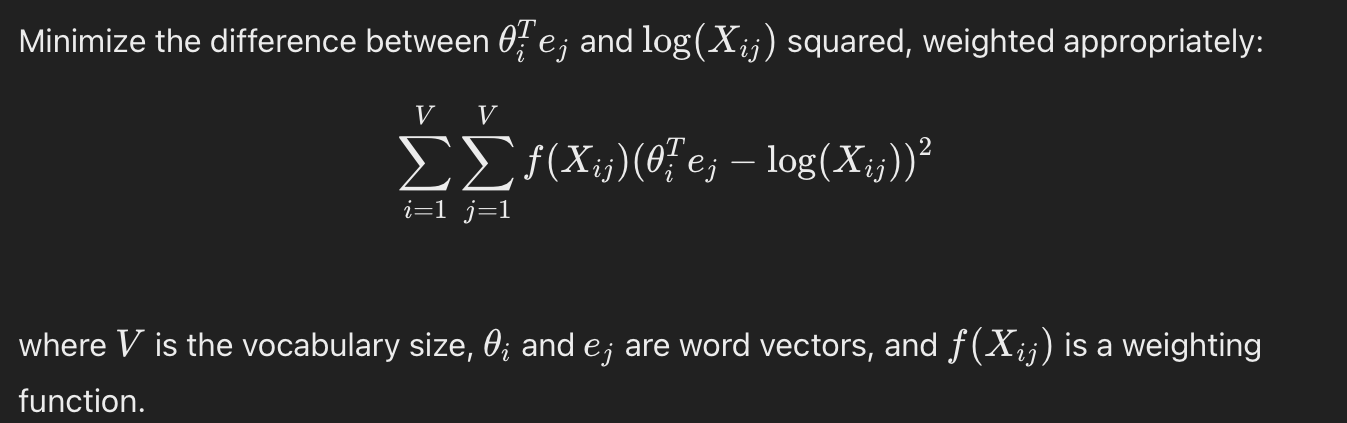
This algorithm simplifies the learning of word embeddings by focusing on co-occurrence statistics.

Here we define a co-occurance matrix

Deine X(i,j)

Here number of times the word I appears I. The context of word jin a context window.

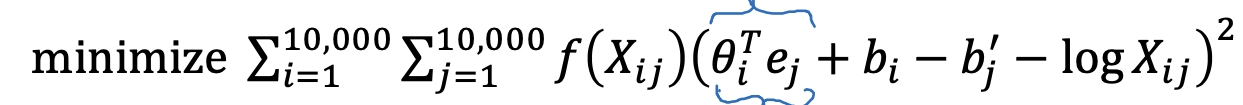
Objective fucntion-



**Weighting function (f(X(I,j))**

Adjusts the frequent and infrequent pairs.

Uses gradient descent to optimize θ and e to minimize the objective function.

Unlike previous models, GloVe treats θ and e symmetrically, allowing averaging of θ and e for final word embeddings.

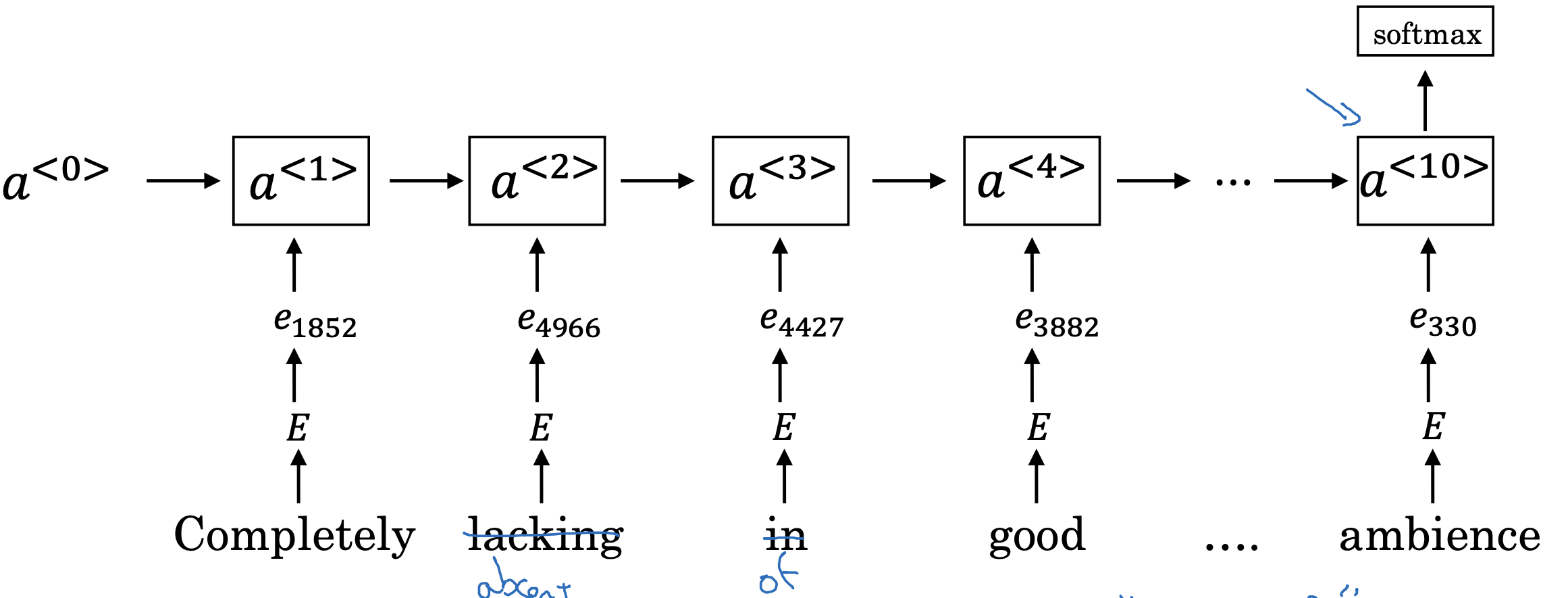
**\*Sentiment Classification**

Sentiment classification in NLP predicts sentiment (e.g., positive, negative) from text

Challenges: Limited labeled data, word order nuances.Use word embeddings (from large corpora) with methods like averaging or LSTM networks

Problem here is Limited labeled data, word order nuances.

Benefits: Generalizes to new words and contexts from embedding training data.

RNN for sentiment classification is given below-

**\*DeBiasing in Word Embeddings**

Steps of debiasing are-

1. Identify bias direction.

2. Neutralize: For every word that is not definitional, project to get rid of bias.

3. Equalize pairs.