Word Representation

A perfect word representations should have:

→ **Shared lemmas:**

Word with same root should have similar representation.

Ex: Mouse / mice , class / classes, eat / ate

→ **Different word sences:**

Should be able to distinguish between different meaning of the same word.

Ex: River Bank, Money Bank, Blood Bank

Computer mouse , pet mouse

→ **Synonyms:**

Synonym words should be close in the vector space.

Ex: Couch / Sofa , Car / automobile

These vectors should have high cosine similarity.

→ **Antonyms:**

Antonyms still be related but in opposite direction

Ex: long / short, dark / light

→ **Word Similarity:**

Words that are not synonyms but are similar in category or roll should be close in vector space.

Ex: Cup / Coffee , doctor / nurse

→ **Word Relatedness:**

Word that often occur together or are functionally connected should also be related, even if they aren't similar.

Example: Cup/coffee , scapel/surgeon

→ **Word valence:**

Represents how positive or negative a word feels. (Emotion Polarity)

Example: High valance: Excited , relaxed (Positive emotion)
Low valance: Angry, depressed (negative emotion)

→ **Word Arousal:**

Represents the intensity / energy or level of activation in emotion.

Example:
High Arousal: Excited , Angry
Low Arousal: Relaxed, Depressed

* A perfect word embedding understands grammar (lemmas), meaning (senses, synonyms), relationship (relatedness, antonyms) and emotional contex (valance, arousal)

* We can guess the semantics of a word from its neighbouring word.

Term Term Matrix

A term-term co-occurrence matrix X is a |V|x|V| matrix where:
→ |V| is the number of words in the vocabulary
→ each cell Xi,j records how often word j occurred in the context of word i
→ each row Xi is the vector representation for word i

Context may be defined in different ways:
- The same document
- The same sentence
- Within ±n words of each other

V is typically the 10,000 - 50,000 most frequent words
- Each word is represented by a large vector

Example:

d_1 = any big cat
d_2 = big cat
d_3 = cat dog cat

Context ± 1 word of each other

	any	big	cat	dog
any	0	1	0	0
big	1	0	2	0
cat	0	2	0	2
dog	0	0	2	0

Context ± 2 word of each other

	any	big	cat	dog
any	0	1	1	0
big	1	0	2	0
cat	1	2	2	2
dog	0	0	2	0

When context window is the entire document, the easy way to build term term matrix is:

① Build a binary BOW for the sentences
② Transpose it
③ Multiply it with the original matrix

Example:

d_1 = any big cat
d_2 = big cat
d_3 = cat dog cat

① Binary BOW:

	any	big	cat	dog
d_1	1	1	1	0
d_2	0	1	1	0
d_3	0	0	1	1

$$\therefore M = \begin{vmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{vmatrix}$$

$$M^T = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

$$M^T \cdot M = \begin{vmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 2 & 0 \\ 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 1 \end{vmatrix}$$

$\therefore$

	any	big	cat	dog
any	1	1	1	0
big	1	2	2	0
cat	1	2	3	1
dog	0	0	1	1

Here,
→ Diagonal values represents how many document had the word.

→ dog appeared in 1 document.

Comparing Word vector:

Most common similarity measure is cosine similarity.

$$\text{cosine}(v,w) = \frac{v^T w}{|v||w|} = \frac{\sum_i v_i w_i}{\sqrt{\sum_i v_i^2} \cdot \sqrt{\sum_i w_i^2}}$$

range of cosine similarity [-1 to 1]

$u = [0\ 1\ 0\ 1]$

$v = [1\ 0\ 1\ 0]$

$w = [3\ 0\ 3\ 0]$

$$\text{cosine}(u,v) = \frac{0*1+1*0+0*1+1*0}{\sqrt{0^2+1^2+0^2+1^2}\ \sqrt{1^2+0^2+1^2+0^2}}$$

Cosine value 1 or closer to 1 means words are similar

Cosine " 0 " " " 0 " " are not related

Cosine " -1 " " " -1 " " are opposite.

Term Term matrix issues:

- Very sparse
- Doesn't carry any contextual information
- Doesn't represent how a word is in a sentence.

How do we use word vectors?

Word level task: finding synonyms via cosine

classify when input is one word

For sentence & document tasks:

- Combine all word vectors by either concatinating or using centroid technique
- Or, Using RNN (More will be discussed letter.
- Then this vectors can be used for classification.

Sparse vs Dense vector:

So far, all of the technique we learnt were sparse. Sparse vector & matrixes have their own disadvantages. So, we will now focus on dense vectors:

Dense vector has following advantages:

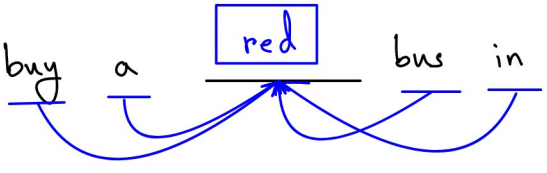
- fewer weights to learn
- fewer features can reduce overfitting
- forces sharing as there are not enough dimension for each word to be completely independent.

Word Embedding:

→ Instead of looking at co-occurance, lets try to predict

Two ways:

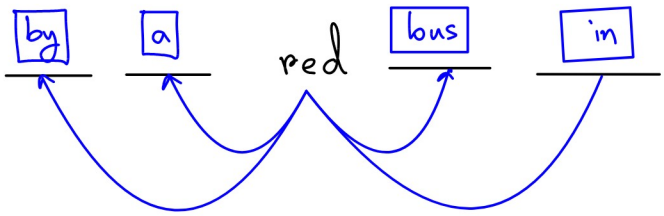
CBOW: Given neighbour words, predict the target word



context/window_size = 2

means how many neighbour we will consider

Skip gram: Predict neighbouring words given the target word.



Cbow is easy, lets focus on Skipgram.

Creating training data is very easy. Given a corpus, we select each word and its neighbour words (each a pair) and append them in training data.

Example: Context window = 2

ϡ live in Dhaka.....

input word	target word
ϡ	live
ϡ	in
live	ϡ
live	in
live	Dhaka
in	ϡ
in	live
in	Dhaka
in	.
Dhaka	live
Dhaka	in
⋮	⋮

* For the very starting word/s and ending word/s, we can add dummy words.

Example:

input word	target word
ϡ	<dummy>
ϡ	<dummy>

Initial Idea of training an embedding model.

① for each word in the dataset, create a one hot vector (shape $1 \times |V|$)

② Pass it to the neural network to calculate prediction

③ Project to output vocabulary using softmax activation function.

The NN architecture :

Vocabulary : { I, live, in, Dhaka }

I live in Dhaka

① One hot vector for each word is =

I = [1 0 0 0]

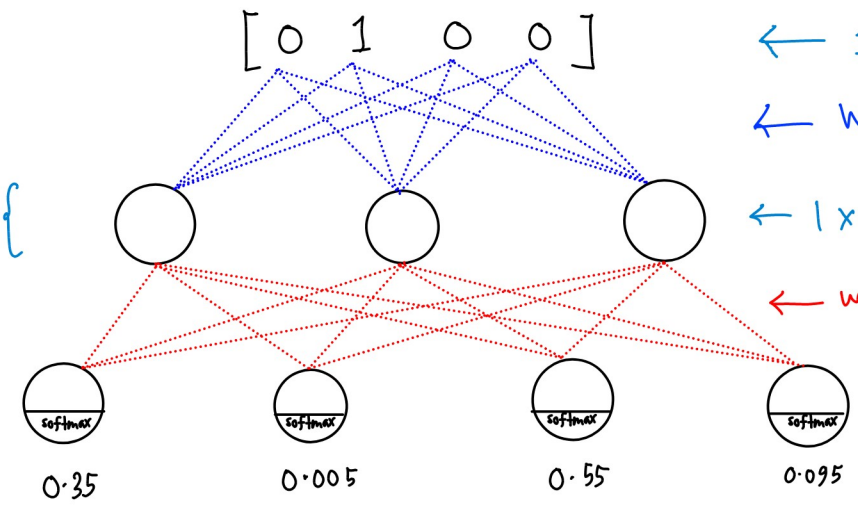
live = [0 1 0 0]

in = [0 0 1 0]

Dhaka = [0 0 0 1]

Let, Our training data is (input: live, target: in)

② live = [0 1 0 0]



← $1 \times |V|$

← Weight matrix = Embedding matrix (EM)
shape = $|V| \times k$

← $1 \times k$

← weight matrix = Context Matrix (CM)
shape = $k \times |V|$

← $1 \times |V|$

hidden layer with k units { here k=3

Output layer $|V|$ neurons as we are predicting neighbouring words one hot vectors.

[0.35, 0.005, 0.55, 0.095]

→ I live in Dhaka

③

prediction of NN = [0.35, 0.005, 0.55, 0.095]

Actual target (in) = [0, 0, 1, 0]

Error = Actual − target

= [−0.35, −0.005, 0.45, −0.095]

Now from this error update weight matrices.

Note: Softmax activation function converts a tuple of K real numbers into a probability distribution of K outcomes.

Output layer:

$$y = \begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \rightarrow \boxed{\text{Softmax}} \rightarrow \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Probabilities

sum = 1

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}$$

After training this Neural Network on our entire dataset, we will get optimum weight matrices: Embedding & Context Matrixes.

Let, for the above network, we get,

$EM = |V| \times k = 4 \times 3$

৪	1	2	1
live	0	1	2
in	1	4	3
Dhaka	1	2	1

→ Word vector for ৪.
→ Word vector for live
⋮

$\oplus$ [3 3 2] → Alternative word vector for ৪.

$CM = k \times |V| = 3 \times 4$

$\therefore CM^T = 4 \times 3$

৪	2	1	1
live	4	3	0
in	1	1	2
Dhaka	1	3	4

Note that: We can either use only EM for our word vector or use the summation of EM & CM for our word vectors.

Also, we are getting a $1 \times k$ dimension (1×3 in this example) vectors. So, how many neuron(k) we will keep in our hidden layer will determine the dimension of the vectors we get. Usually we keep $50 \leq k \leq 1000$.

Theoretically, this model should give us properly learnt word vectors. But there is a problem. The step (3) ↴

(3) Project to output vocabulary using softmax activation function.

is extremely computationally expensive. In proper dataset & training environment, $|V|$ can be from 50K to 1 million. This means the output layer has too many neurons & for each of them we need to compute softmax "for each training data."

This is where Computer Scientist of Czech, Mikolov entered in the game.

Mikolov reformulated the task. Instead of predicting neighbouring word, Mikolov converted it to a simple classification task.

Before:

ꕥ (live, in) Dhaka

After:

that means, given two words from the training data, we try to predict whether they are neighbour or not.

[Word2vec Training Steps]

Steps:

① Converting the training data

Example: Context window = 2

ꕥ live in Dhaka.....

input word	output word	target
ꕥ	live	1
ꕥ	in	1
live	ꕥ	1
live	in	1
live	Dhaka	1
⋮	⋮	⋮
ꕥ	Dhaka	0
ꕥ	ꕥ	0
Dhaka	ꕥ	0
ꕥ	else	0

② Negative Sampling:

After step 1, all of the target in training data is 1. All 1 in training data means all words in training data are neighbour which is not good for learning. So, we randomly add some words which are not neighbour with target = 0. This is called random sampling.

③ Initialize Embedding Matrix $(|v| \times k)$ & Context Matrix $(k \times |v|)$ with random values.

④ Take input word vector from EM and output word vector from CM

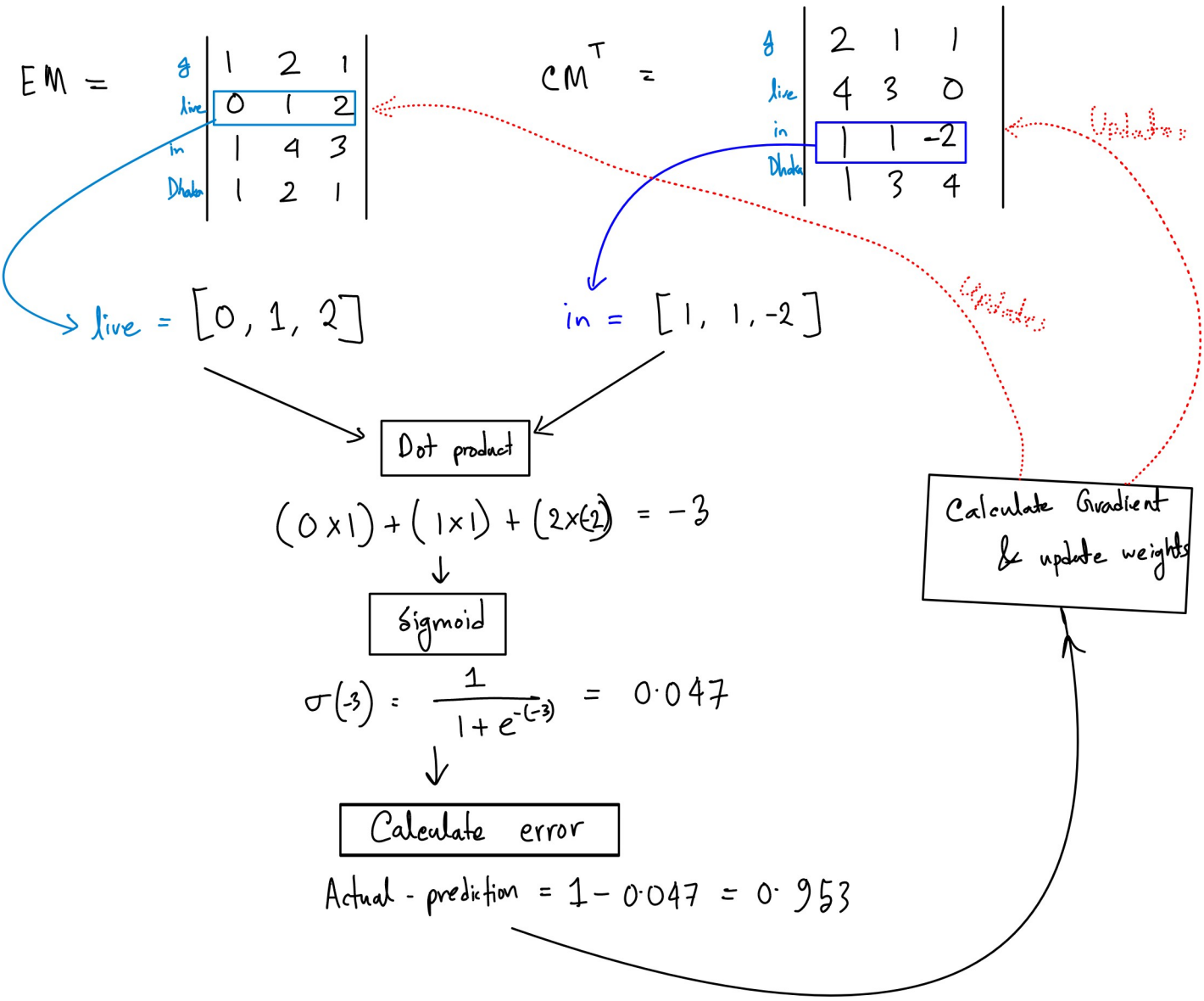
⑤ Dot product two vectors and pass them on sigmoid function for binary classification.

⑥ Calculate error and update the word vectors accordingly on both EM & CM.

Doing this for the entire dataset will give us the trained EM and CM.

Example: Let,

Training data = (live, in, 1)



This entire process gives us the updated weight Matrix EM & CM. Now we can use them as word vectors.

While working, we do not need to code from scratch. We can directly use gensims Word2vec.

We dont need to train Word2vec always. It is rarely necessary to train skipgram. Instead, we trained it once on massive data and use this pre trained embeddings directly.

Semantic Properties of Embeddings

* Different type of similarity or association

Based on the context window, word association changes:

Smaller window size (2 – 15) embeddings

→ high similarity score indicates that word are interchangable

Example: Hogwart is similar to Sunnydale

(both are school/academy)

→ Synonym or similar word stays close in the vector space.

Larger window size (15 – 50 or more) embeddings

→ high similarity score indicates relatedness.

Example: Hogwart is more similar to dumbledore, malfoy etc.

(As they are from same book/stories.

* Analogy / Relational Similarity:

Embeddings captures relational meanings. Therefore, we can do analogies like:

$$\overrightarrow{King} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{Queen}$$

$$\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italy} \approx \overrightarrow{Rome}$$

Historical Semantic Context:

The data we use reflects on our embedding. So, we can see how the meaning of word changes through time.

We can visualize this using PCA or t-SNE which compresses the high dimensional word vectors into lower dimension for plotting. (Ex: 300D to 2D)

Bias in Embeddings:

As embeddings are learnt from training data, it captures the bias from it as well.

Example:
- cosine(man, attractive) = 0.3085
- cosine(woman, attractive) = 0.4111
- cosine(dumb, American) = 0.4118
- cosine(dumb, European) = 0.2658

Also,

he is a nurse. she is a farmer.

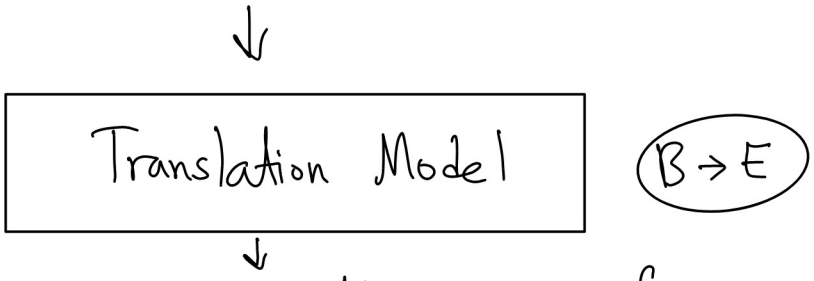
↓

Translation model (E → B)

↓

সে একজন নার্স। সে একজন কৃষক।

↓



She is a nurse. He is a farmer.

Static Word Embedding:

So far, the word vectors we learnt about has a fixed vector after training. They ignore the context in that sentence.

Ex: The river bank, A bank deposite

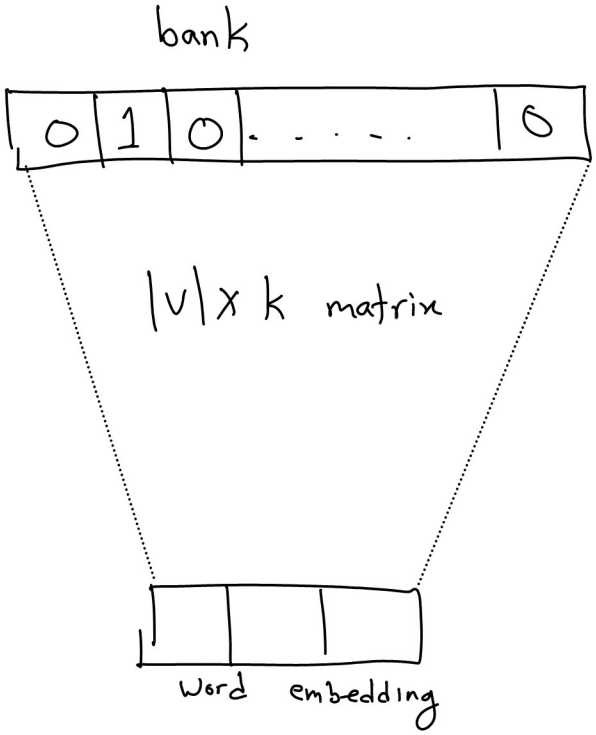
They shouldn't be similar, Based on context, they should be different.

Static Embedding captures both the meaning of money bank and river bank. But this is not accurate.

Solution ?

Contextual Word Embeddings :

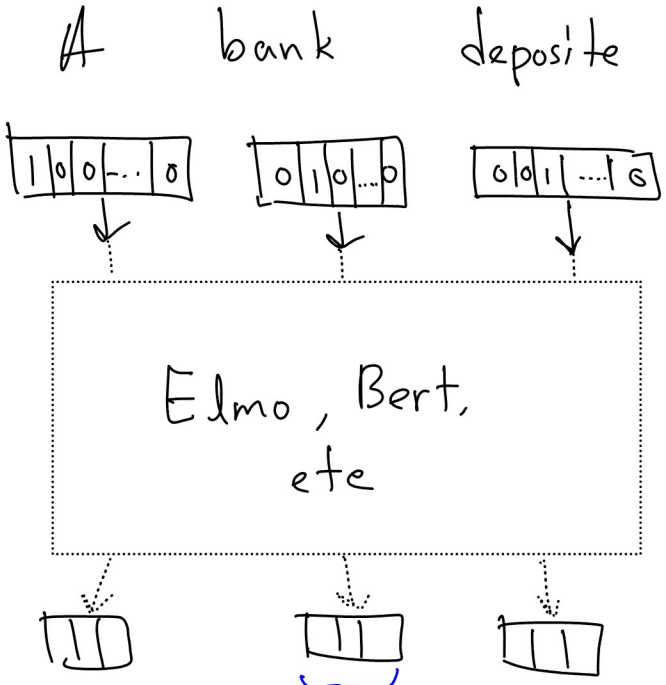
Static Word Embedding :



Input 1 word

Output 1 embedding

Contextual Word Embedding :



Input N word

Output N embedding

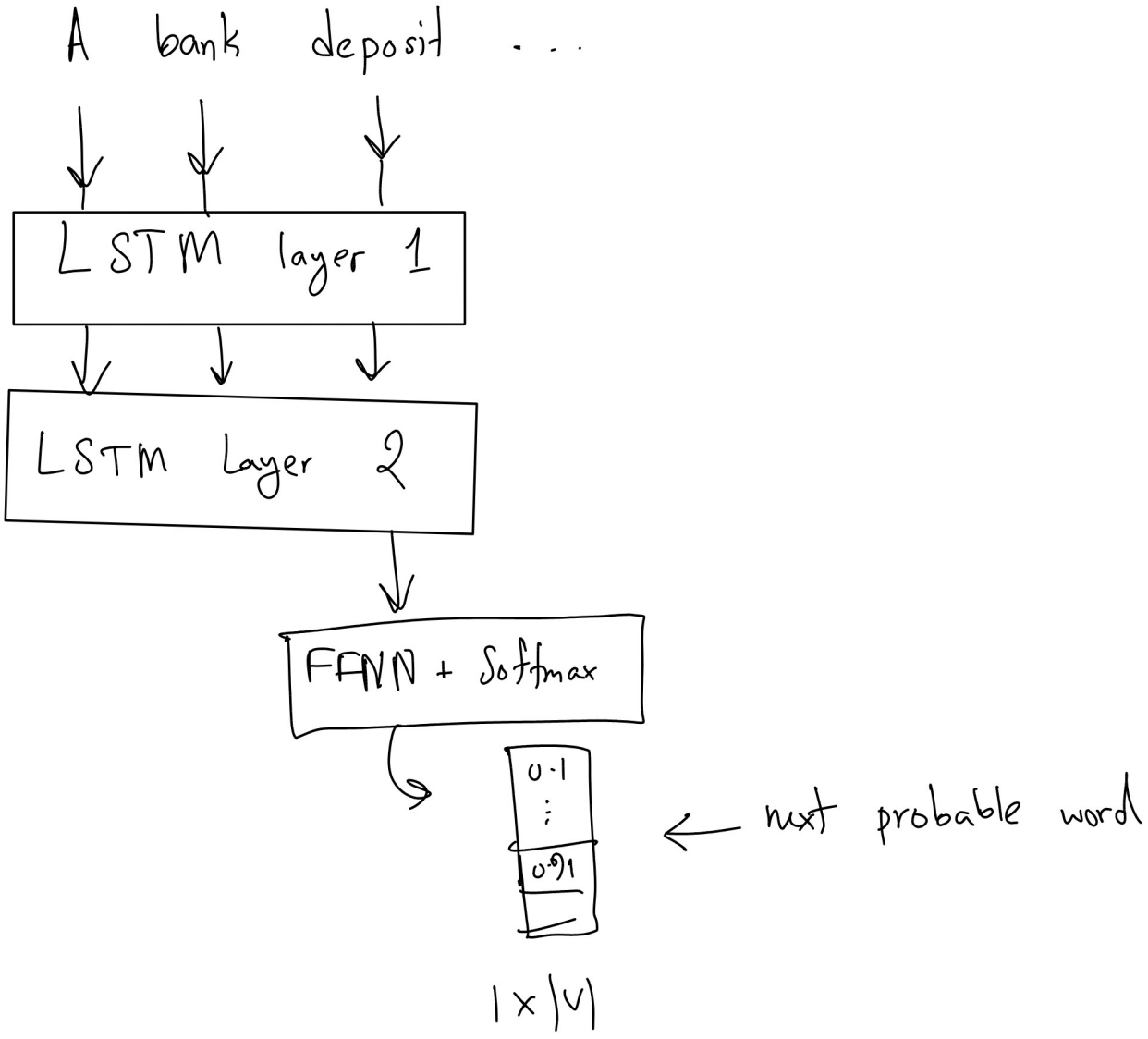
This 'bank' embedding has the context of neighbour words. So, it has the properties of money bank.

Language Model:

→ Given a sequence of word, predict the next word.

→ Unsupervised, great for learning word representation.

Elmo's task:



How to use contextual embedding?

① Train on Unlabeled data } Pre training

② Extract word vectors to use as feature

③ Fine tune it on labeled data. } finetuning.