

# Outline

- Motivation
- word2vec
- Negative Sampling
- GloVe
- Applications

# Word Embeddings

- It is useful to have numerical representations for words
  - Computers only really understand numbers
- Similar words should have similar numerical representations!
- A word will always have the same embedding no matter its context!

# Word Embeddings

- It would not be great to have a single number for each word
  - The word 'set' has ~430 ways it can be used in English!
- Each word should thus be represented by an array of numbers
  - Hopefully, each possible use of a word is captured in this array, somehow
- Let's assume each word is represented with an array of 300 numbers

# Word Embeddings

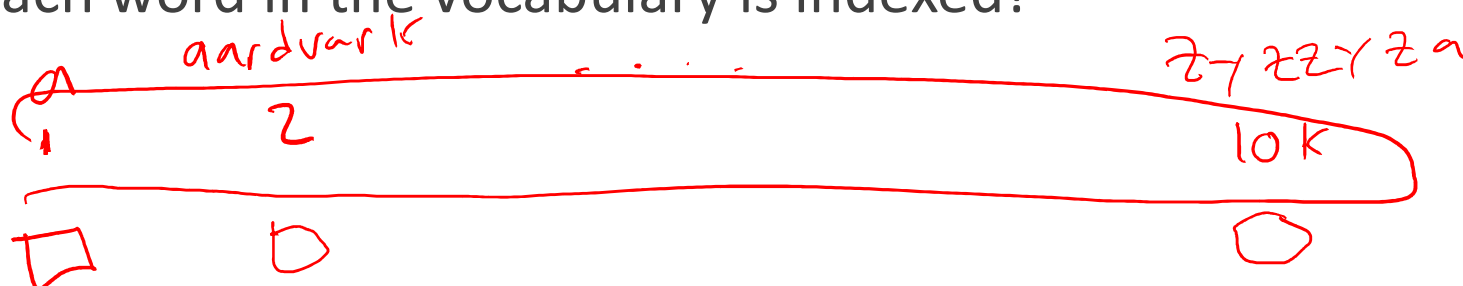
- How do we accomplish this?
  - Train a Neural Network model!
  - Goal of the model is to predict words
  - Train the embedding model using a large corpus of text
    - Something like all of Wikipedia
- Our model should learn:
  - Which words are similar
  - Which words appear together
- Extract some weights and biases from this network
  - These are our word embeddings!

# Large Language Models

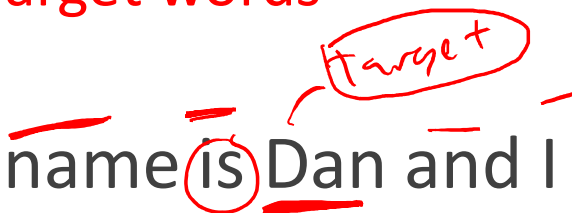
- There have been recent developments in word embeddings using Large Language Models (LLM)
  - The same word can have different embeddings, depending on the context
  - I set up a chess set to play a set of games.
- These are typically transformer-based embeddings
  - We'll talk about them later!

# Setup

- To train a model for word embeddings we need a clean large corpus of text
  - Assume you have a corpus that has been tokenized and separated into blocks of text
    - Separate based on sentence or paragraph or ...
- The number of tokens in this corpus is 10k
  - **Vocabulary**
  - Each word in the vocabulary is indexed!



# Setup

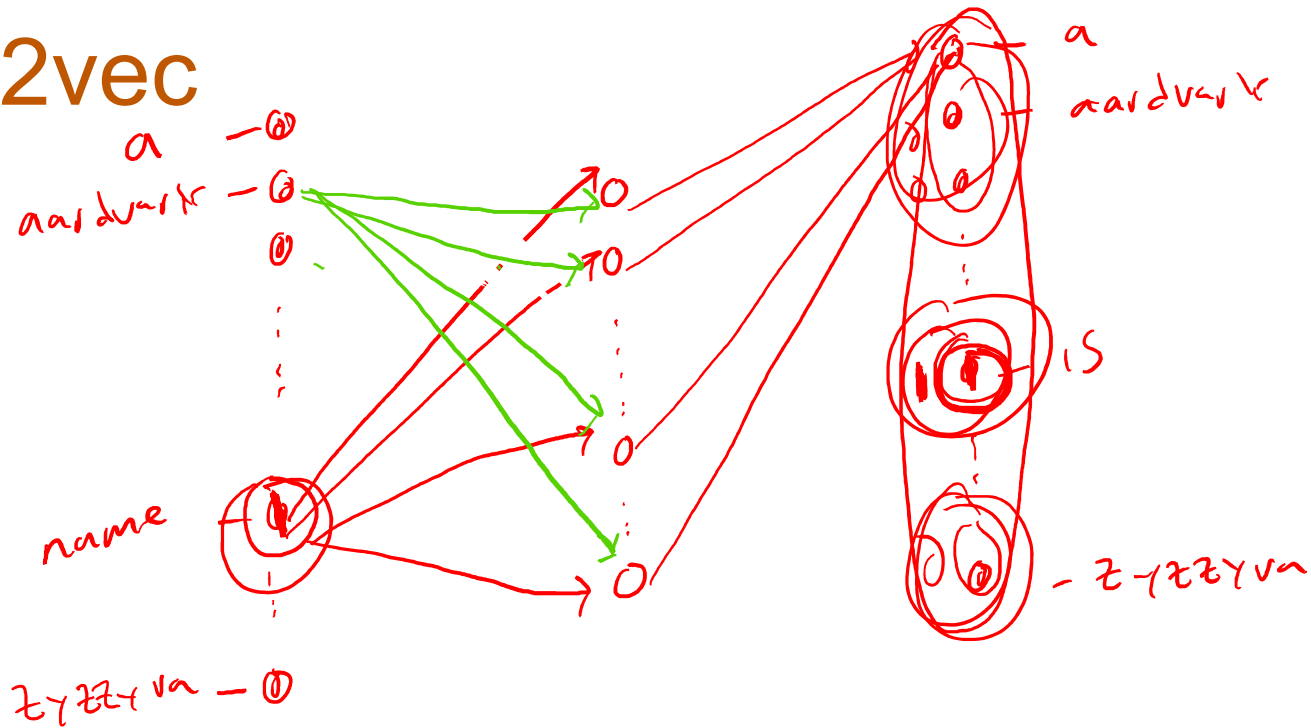
- We now want to train a model to get meaningful embeddings for each token (word) in the vocabulary
- To train the model we need to understand a couple things
  - Context words
  - Target words
- My name is Dan and I work at UT.  

- Sampling – SGD

## word2vec

- Build a neural network
  - 10k neurons on input layer
  - 300 neurons on hidden layer, no bias, no activation function
  - 10k neurons on output layer, softmax activation function
- All neural networks need data for inputs,  $x$ , and outputs,  $y$



# word2vec



- Eventually, the weights between the input layer and the hidden layer will contain the embeddings for each word

# word2vec

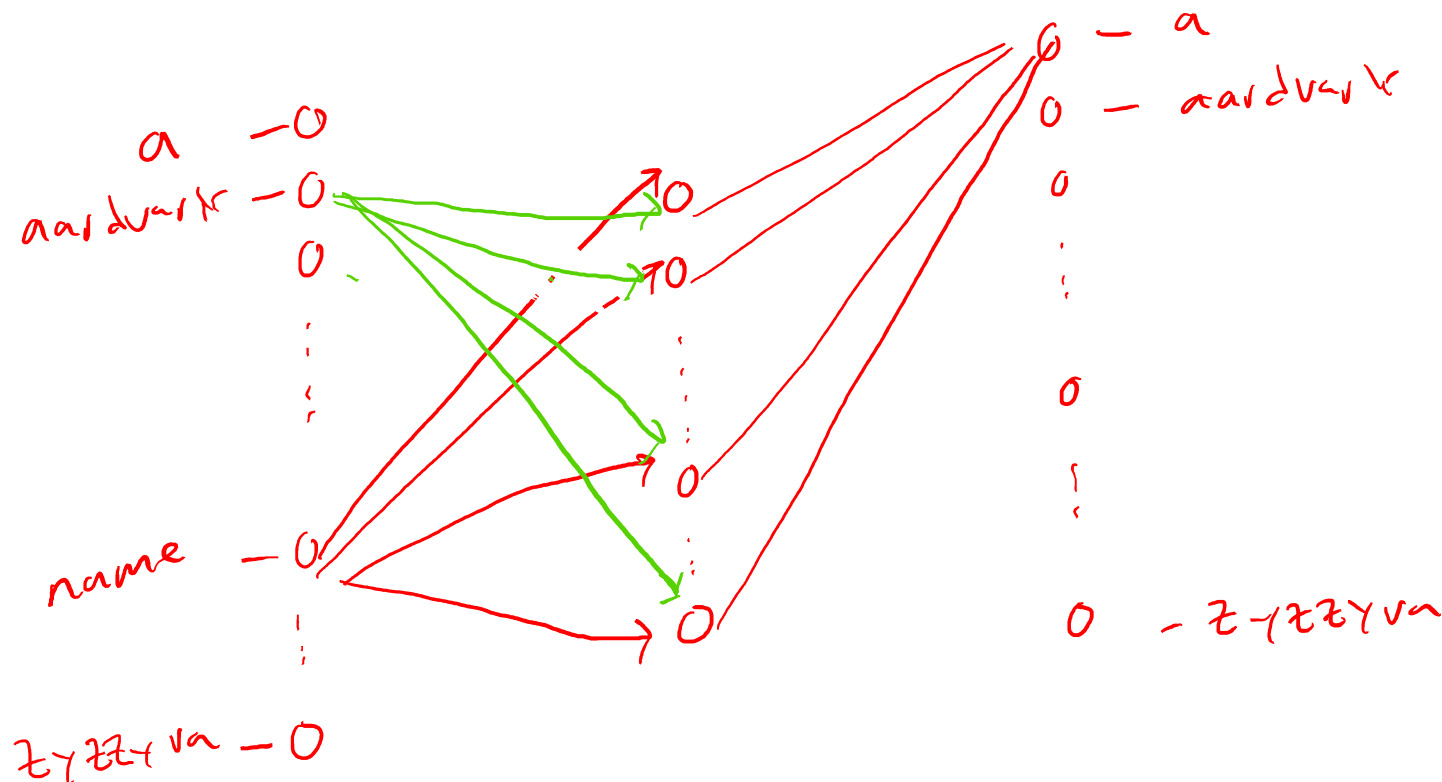
- The goal is to:
  - Plug in context words to the input
  - Have the neural network predict the target words
- Each output neuron will predict  $\hat{y}$ , the probability that each word is the target word for an input context

## word2vec

- We need to train this neural network
- Simple idea:
  - Randomly grab a word (index  $j$ ) from somewhere in the corpus, make it the context word
    - Set input neuron  $x_j = 1$ , all other input neurons are 0
  - Make the next word in the corpus the target word, index  $k$ 
    - Set  $y_k = 1$
  - Loss function for  $\hat{y}_k$  vs  $y_k$  and SGD!
  - Repeat for many context/target pairs

# word2vec

- My name is Dan and I work at UT.



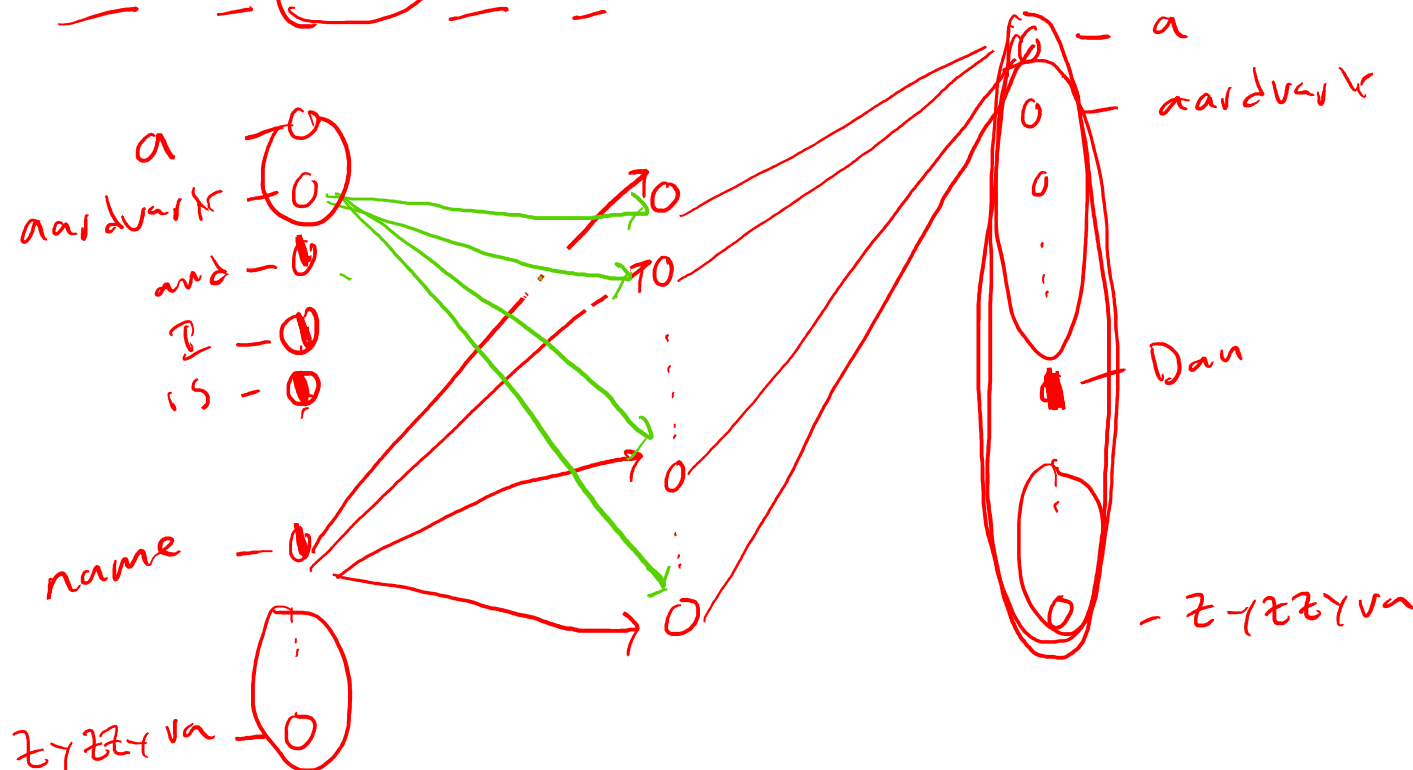
## word2vec – Continuous Bag of Words (CBOW)

- With just 2 sequential words, we can't learn that much context
- Randomly pick a target word from the corpus, index  $k$ 
  - Set  $y_k = 1$
- Take a few words (hyperparameter) before and after the target as the context
  - For every word,  $j$ , in the context, set input neuron  $j = 1$ , all other input neurons = 0

# CBOW

- My name is Dan and I work at UT.
 

100    300    Target 100 200

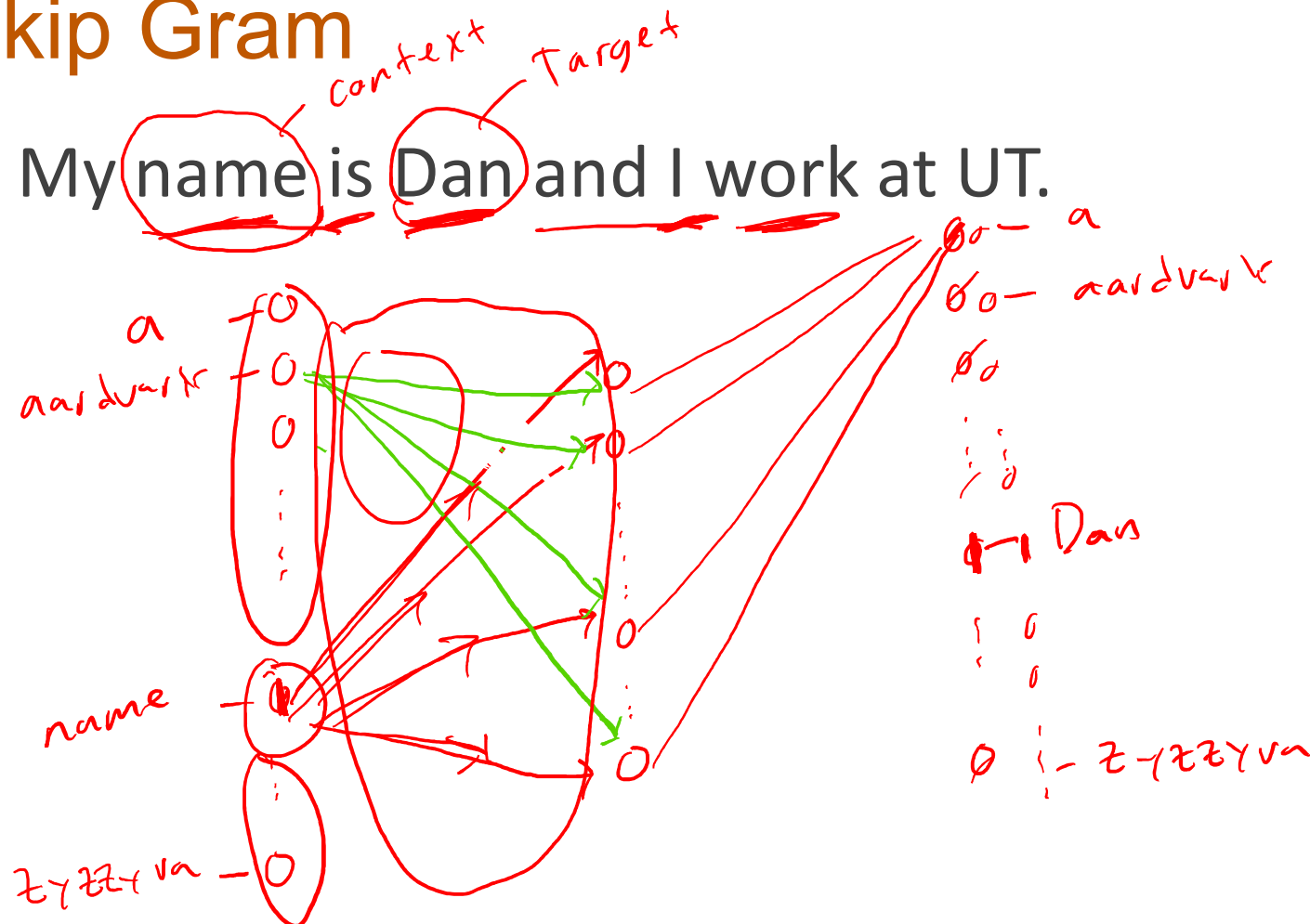


## word2vec – Skip Gram

- Randomly pick a word from the corpus to be the target word, index  $k$ 
  - Set  $y_k = 1$
- Randomly pick a word within some range of the target to be the context word, index  $j$ 
  - Set  $x_j = 1$
- SGD

# Skip Gram

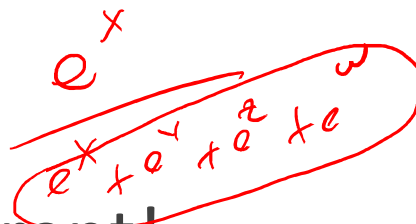
- My name is Dan and I work at UT.





## word2vec

- word2vec has  $\sim 10k \times 300 \times 2 = 6M$  parameters to estimate
- Softmax is very slow!
- Let's try something different!


$$e^x$$
$$e^{x \cdot w_1} + e^{x \cdot w_2} + e^{x \cdot w_3} + e^{x \cdot w_4}$$

# Negative Sampling

- Are 2 words part of a context/target pair?
- Build a neural network
  - 10k neurons on input layer
  - 300 neurons on hidden layer, no bias, no activation function
  - 10k neurons on output layer, sigmoid activation function
- For each target word, we're asking 10k yes/no questions!

19

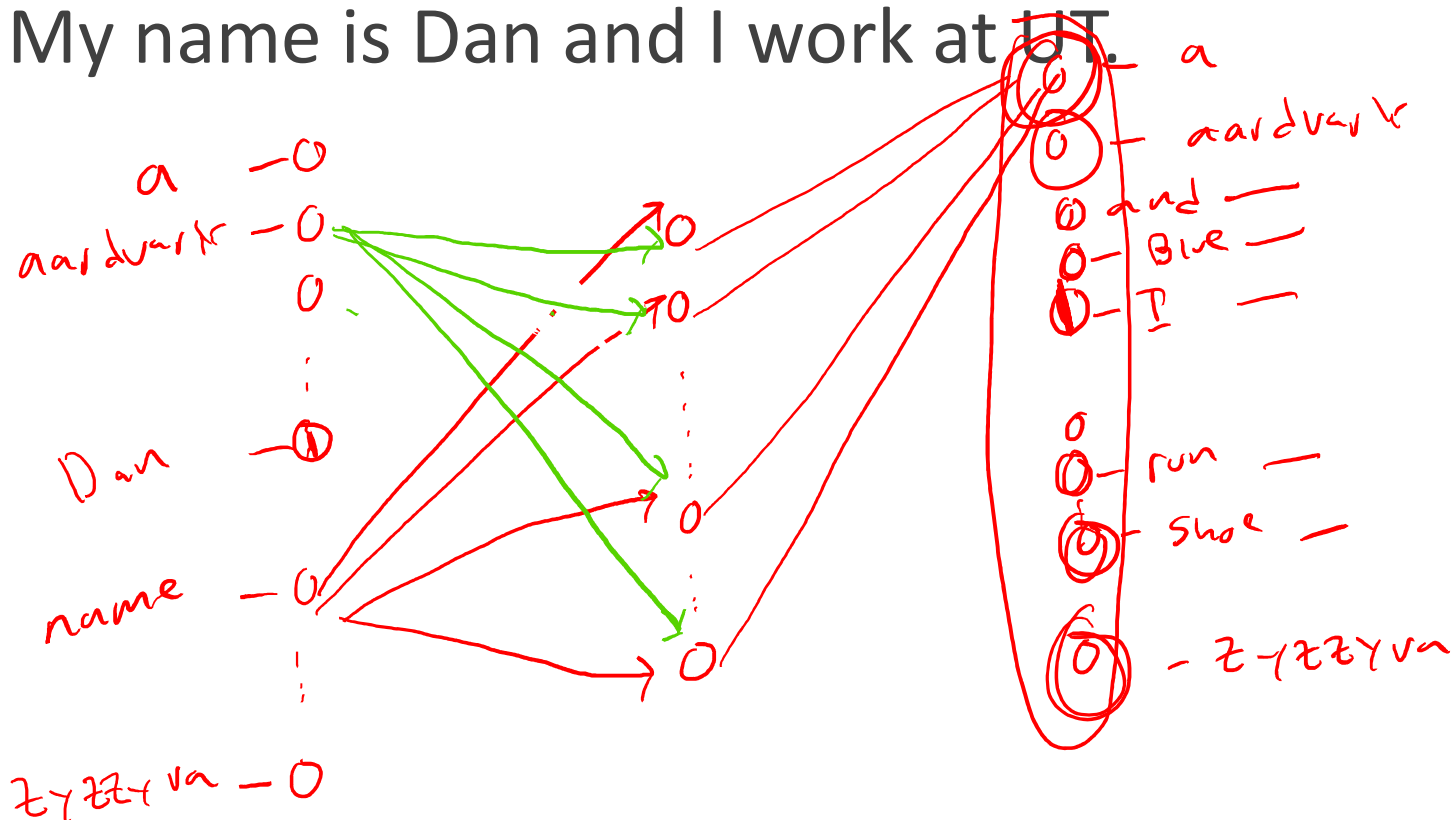
# Negative Sampling

- Randomly pick a target word from the corpus
- Within some range of that word, pick a context word at random
- Randomly pick 4 other words from the vocabulary
- My name is Dan and I work at UT.

Target	Context	yes/no
<u>Dan</u>	<u>I</u>	1
- Dan	shoe	0
- Dan	run	0
- Dan	and	0
- Dan	blue	0

# Negative Sampling

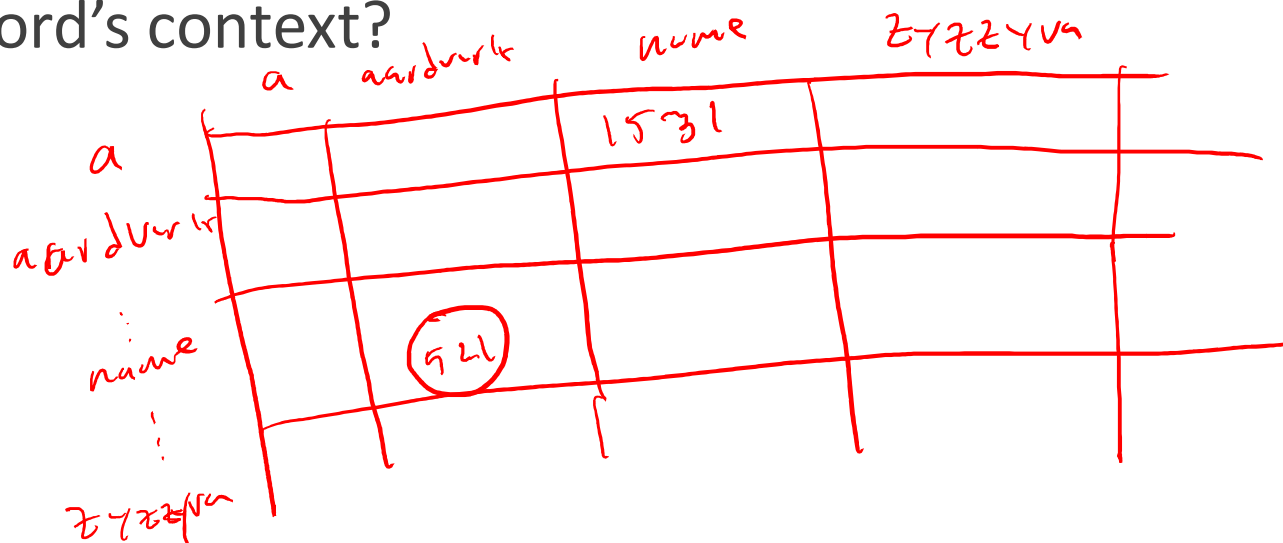
- My name is Dan and I work at UT.



# GloVe

- word2vec and negative sampling only look locally at individual contexts
- GloVe is short for Global Vectors
- How many times does each word in the vocabulary show up in every other word's context?

- $y_{jk}$



	a	aardvark	name	zyzzva
a			1531	
aardvark				
name		521		
zyzzva				

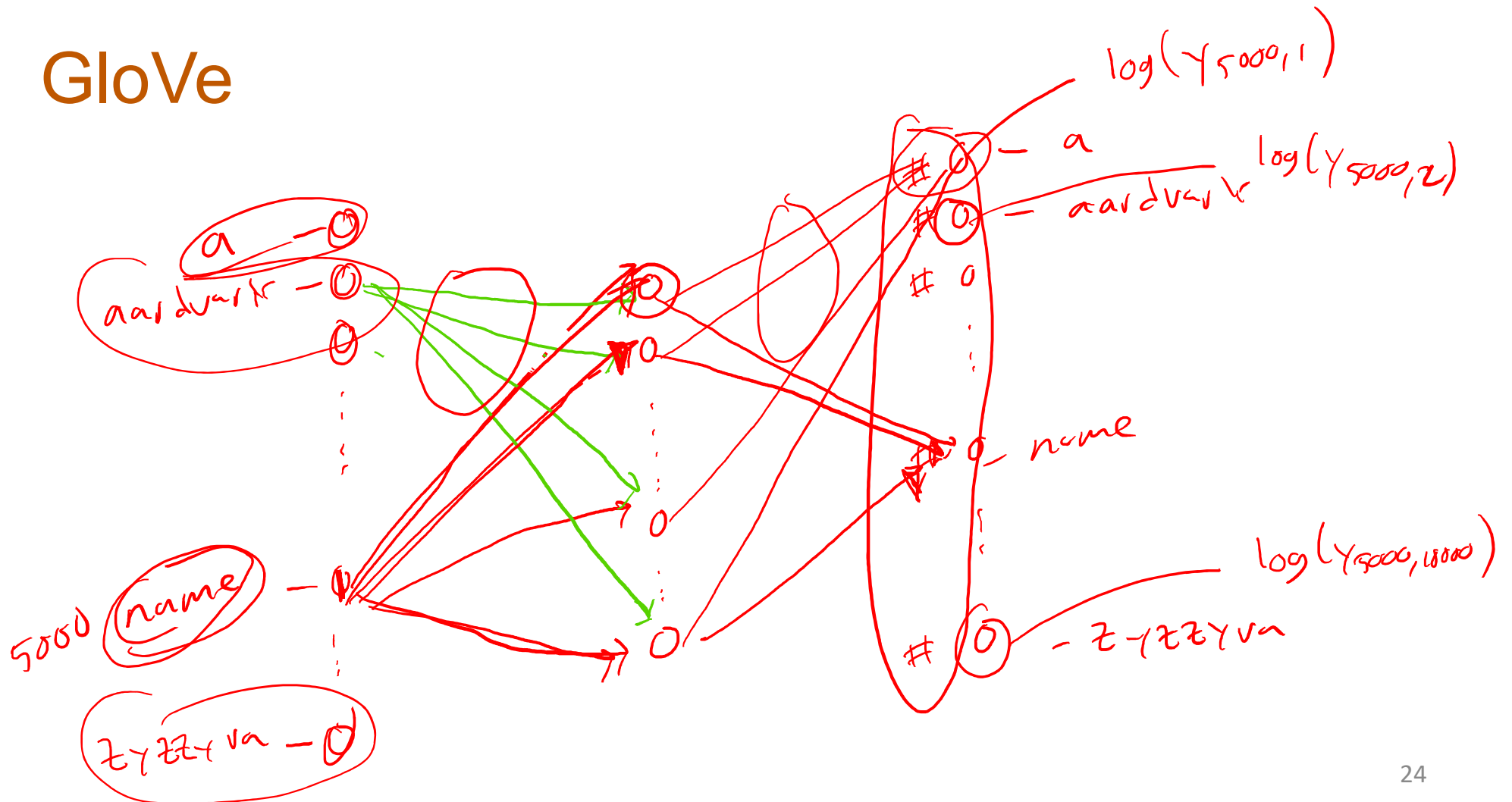
# GloVe

- Build a neural network
  - 10k neurons on input layer
  - 300 neurons on hidden layer, no activation function
  - 10k neurons on output layer, no activation function
- For each target word, compare the output of NN to

*word*

$$\log(y_{jk})$$

# GloVe



# Word Embeddings

- Training our own word embeddings can be slow
- Fortunately, many people have done this already!
  - We can find other people's embeddings online
  - We will use the GenSim package in python for this



# Implication

- Each word is now embedded so that
  - Similar words have similar embeddings
  - Words that appear together have similar embeddings
- That means we can measure the “distance” between words

Dan — name  
 $\begin{pmatrix} 5 \\ -2 \\ \vdots \\ 0.3 \end{pmatrix}$  —  $\begin{pmatrix} 4.8 \\ -1.3 \\ \vdots \\ 0.8 \end{pmatrix}$

$$\frac{5 * 4.8 + -2 * -1.3 + \dots + 0.3 * 0.8}{\sqrt{5^2 + (-2)^2 + \dots} * \sqrt{4.8^2 + (-1.3)^2 + \dots}}$$

# Application

- Text classification
  - Take output of embedding and use it as input for ML model
- Average the embeddings for each word in a block of text
  - Sentiment analysis

# Application

- Semantic Synonym Search
  - Compare distance between words in vocabulary
- Search for 'cabin'
  - Find results for 'house' too!

# Semantic Sentence Search

- Average the embeddings for each word in a block of text
  - Stocks with high volatility are dangerous.
  - Equities that have large variability seem risky.
- We will see an application of this
- Later we will embed blocks of text, instead of just individual words

# Summary

- There are several methods to create word embeddings
- They all fit some sort of machine learning model to get the embeddings
- The models are fit using a large corpus of text
- Word embeddings can be used for many downstream tasks
  - Like semantic search