



Word2Vec and GloVe

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

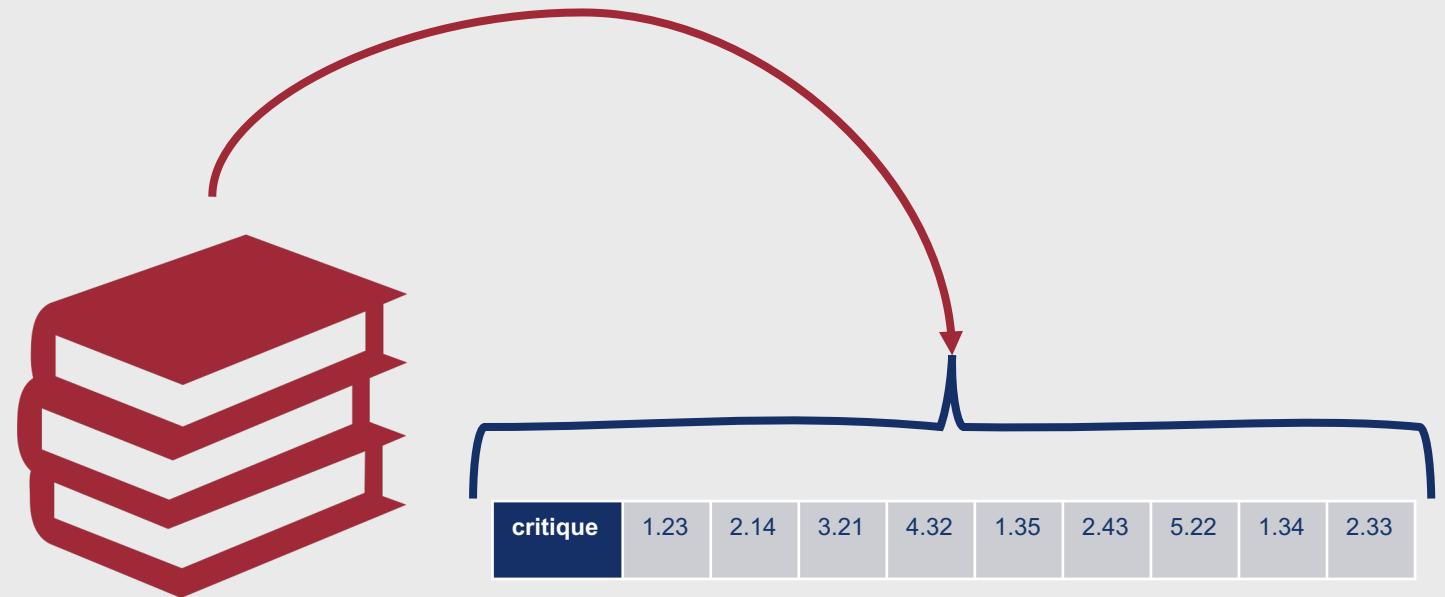
CS 521: Statistical Natural Language
Processing

Spring 2020

Many slides adapted from Jurafsky and Martin
(<https://web.stanford.edu/~jurafsky/slp3/>).

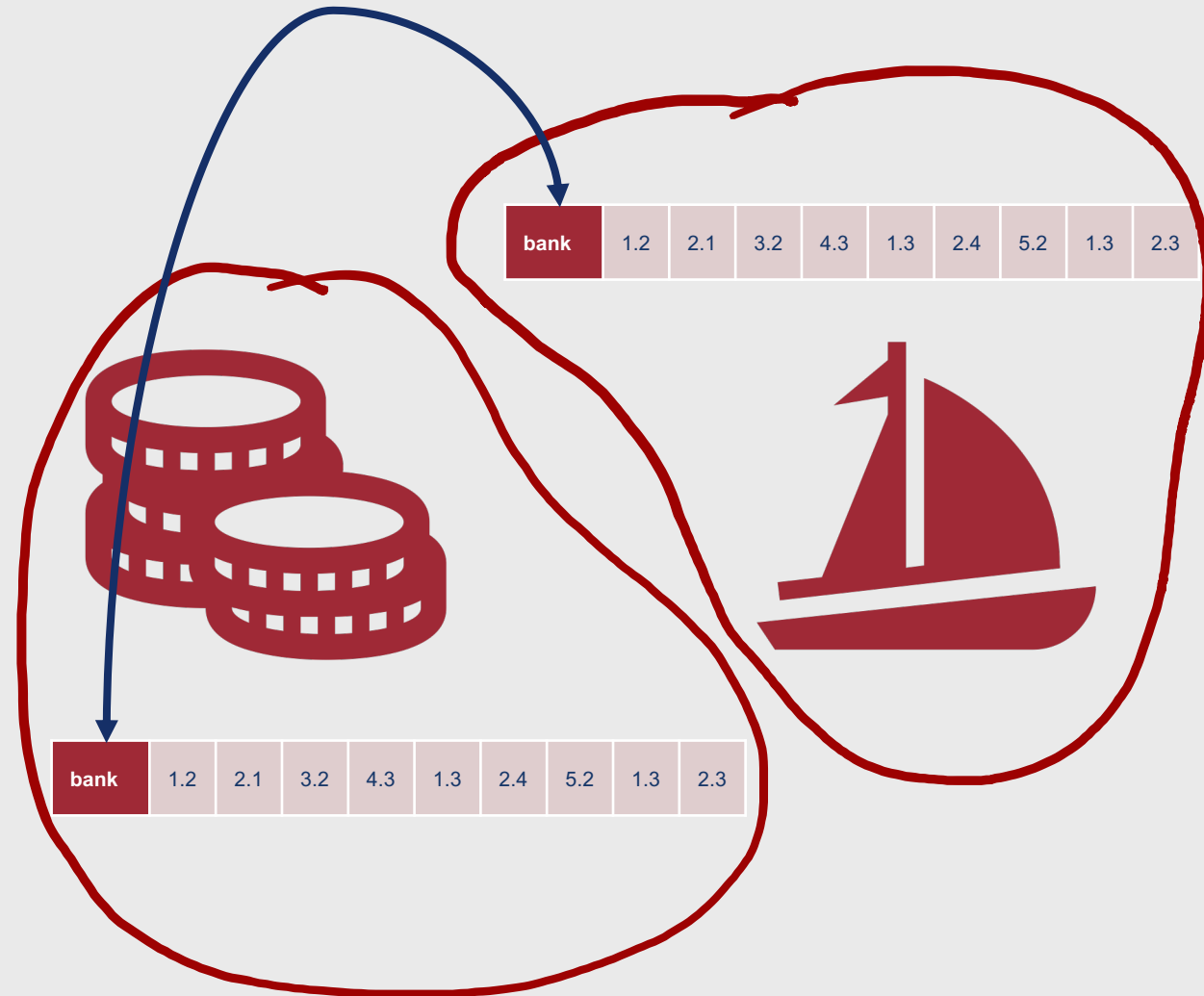
- Method for automatically learning dense word representations from large text corpora

What is Word2Vec?



Characteristics of Word2Vec

- Non-contextual
- Fast
- Efficient to train




Word2Vec

- Technically a tool for implementing word vectors:
 - <https://code.google.com/archive/p/word2vec>
- The algorithm that people usually refer to as *Word2Vec* is the **skip-gram** model with **negative sampling**

How does Word2Vec work?

- Instead of counting how often each word occurs near each context word, train a classifier on a **binary prediction task**
 - Is word w likely to occur near context word c ?
- The twist: **We don't actually care about the classifier!**
- We use the **learned classifier weights** from this prediction task as our word embeddings



None of this
requires
manual
supervision.

- Text (without any other labels) is framed as **implicitly supervised** training data
 - Given the question: Is word w likely to occur near context word c ?
 - If w occurs near c in the training corpus, the gold standard answer is “yes”
- This idea comes from **neural language modeling** (neural networks that predict the next word based on prior words)
- However, Word2Vec is simpler than a neural language model:
 - It makes **binary yes/no predictions** rather than predicting words
 - It trains a **logistic regression classifier** instead of a deep neural network

this sunday, watch the super bowl at 5:30 p.m.

c1 c2 t c3 c4

What does the classification task look like?

- Assume the following:
 - **Text fragment:** this sunday, watch the super bowl at 5:30 p.m.
 - **Target word:** super
 - **Context window:** ± 2 words

this sunday, watch the super bowl at 5:30 p.m.

c1 c2 t c3 c4

What does the classification task look like?

- **Goal:** Train a classifier that, given a tuple (t, c) of a target word t paired with a context word c (e.g., (super, bowl) or (super, laminator)), will return the probability that c is a real context word
 - $P(+ | t, c)$



How do we predict $P(+ | t, c)$?

- We base this decision on the similarity between the input vectors for t and c
- More similar vectors \rightarrow more likely that c occurs near t

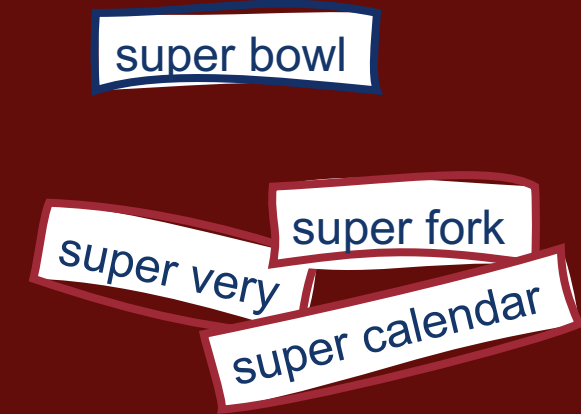
High-Level Overview: How Word2Vec Works

- Treat the target word w and a neighboring context word c as positive samples

super bowl

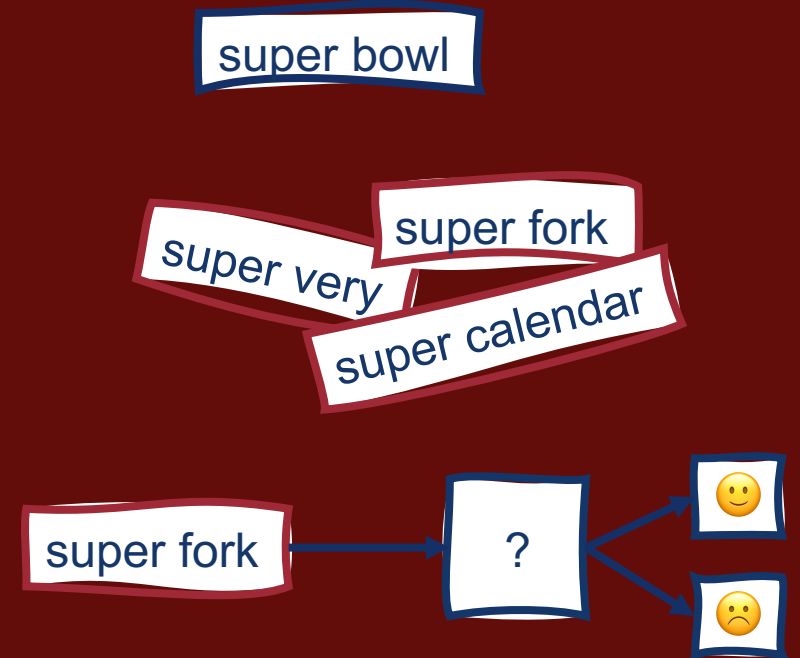
High-Level Overview: How Word2Vec Works

- Treat the target word w and a neighboring context word c as positive samples
- Randomly sample other words in the lexicon to get negative samples



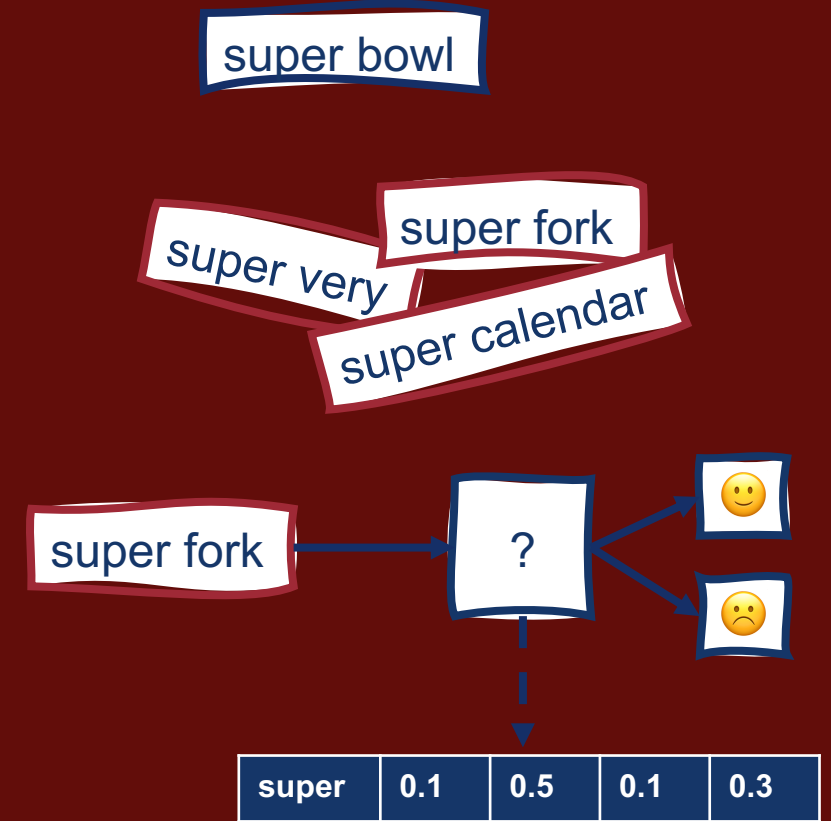
High-Level Overview: How Word2Vec Works

- Treat the target word w and a neighboring context word c as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Use logistic regression to train a classifier to distinguish between those two cases



High-Level Overview: How Word2Vec Works

- Treat the target word w and a neighboring context word c as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Use logistic regression to train a classifier to distinguish between those two cases
- Use the weights from that classifier as the word embeddings



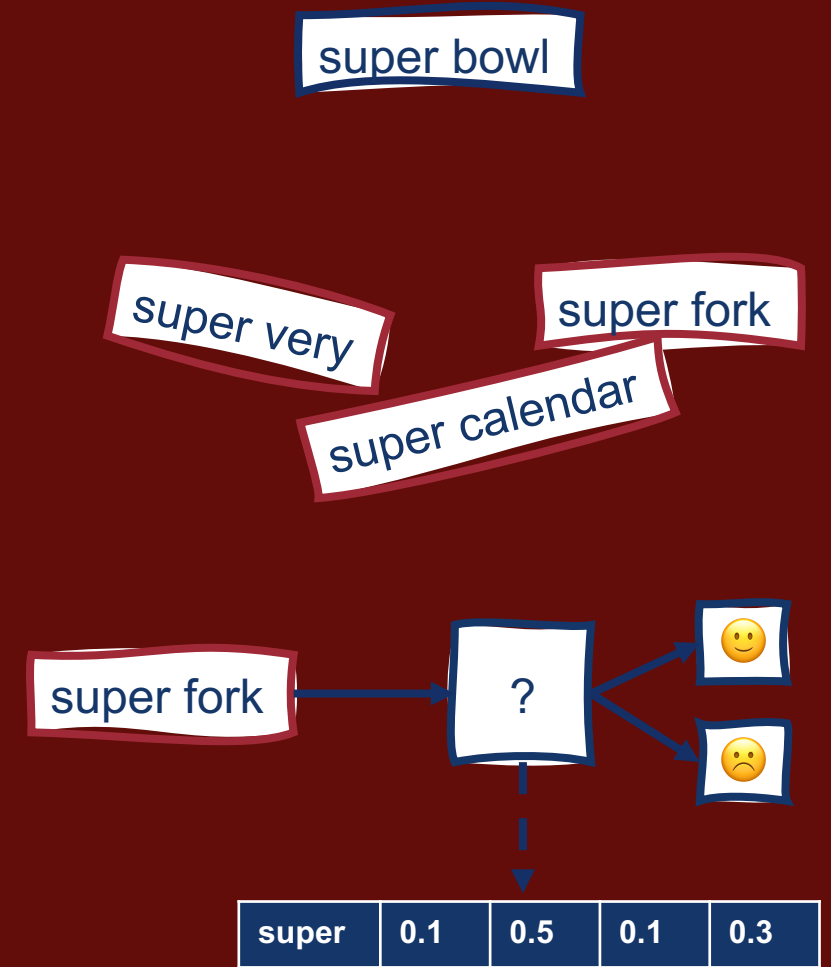


Just one more thing....

- Remember, words are represented using input vectors (somehow, we need to create these)
- We're using the similarity between those vectors to inform our classification decisions

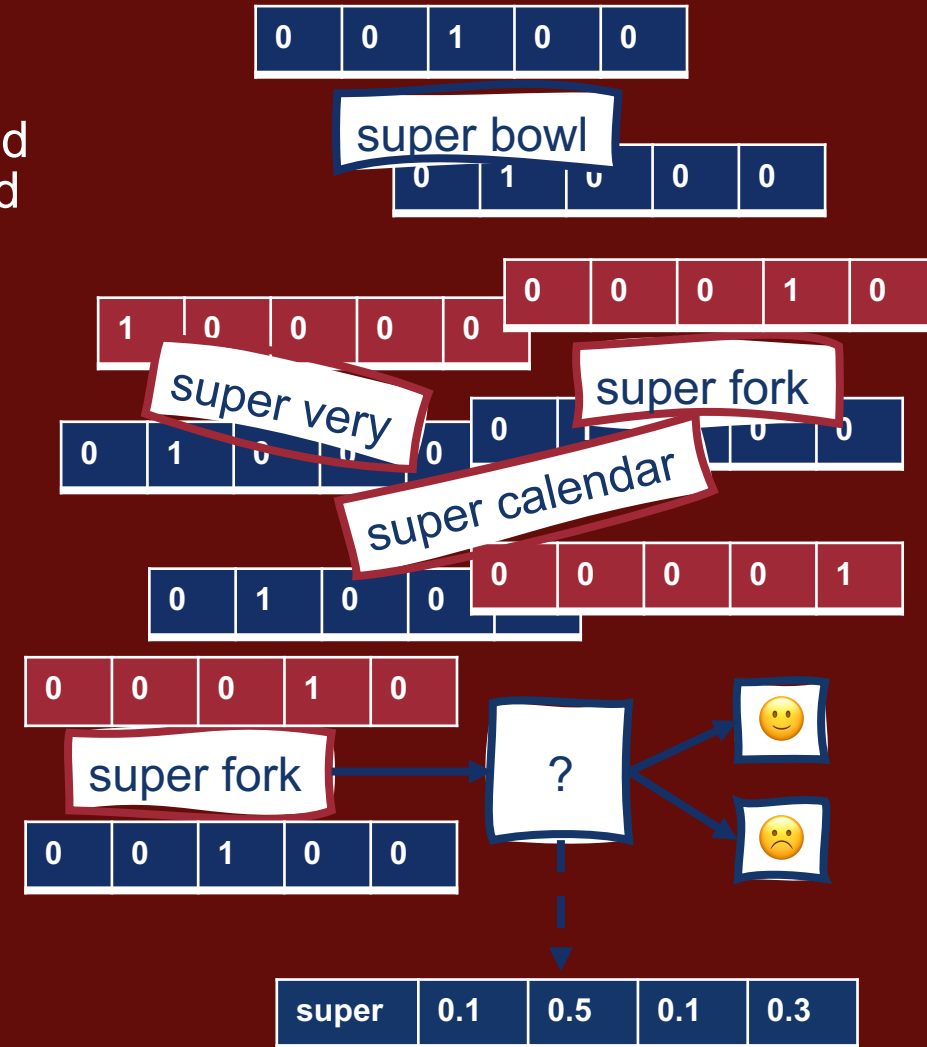
High-Level Overview: How Word2Vec Works

- Treat the target word w and a neighboring context word c as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Use logistic regression to train a classifier to maximize these probabilities to distinguish between positive and negative cases
- Use the weights from that classifier as the word embeddings



High-Level Overview: How Word2Vec Works

- Represent all words in a vocabulary as a vector
- Treat the target word w and a neighboring context word c as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Find the similarity for each (t,c) pair and use this to calculate $P(+|(t,c))$
- Use logistic regression to train a classifier to maximize these probabilities to distinguish between positive and negative cases
- Use the weights from that classifier as the word embeddings





How do we *compute* $P(+ | t, c)$?

- This is based on vector similarity
- We can assume that vector similarity is proportional to the dot product between two vectors
 - $\text{Similarity}(t, c) \propto t \cdot c$

A dot product doesn't give us a probability though....

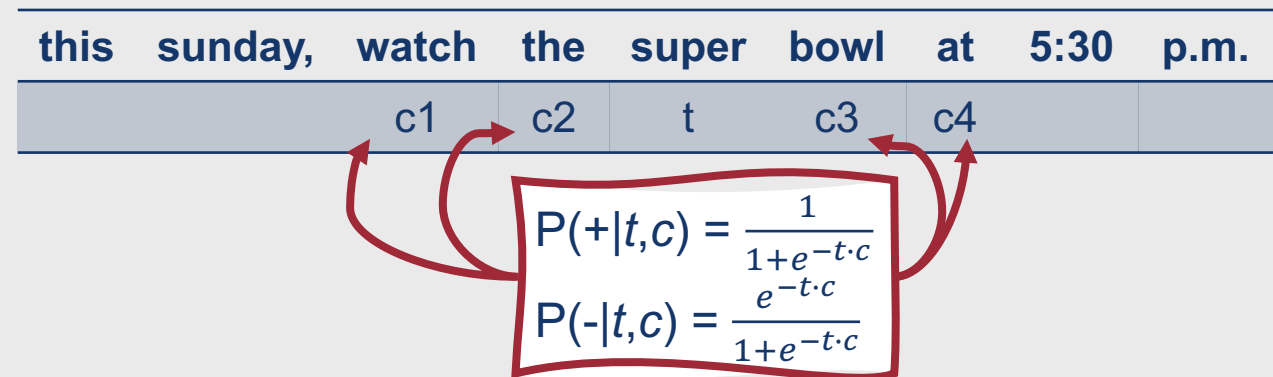
- How do we turn it into one?
 - **Sigmoid function** (just like we did with logistic regression!)
 - We can set:
 - $P(+|t,c) = \frac{1}{1+e^{-t \cdot c}}$



Just like with logistic regression, the sigmoid function doesn't automatically return a probability.

- What *does* it return?
 - A number between 0 and 1
- However, we can follow our approach from logistic regression to convert it to a probability---all we need to do is make sure the sum of the values returned for our two possible events (c is or is not a real context word) equals 1.0
 - $P(+ | t, c) = \frac{1}{1+e^{-t \cdot c}}$
 - $P(- | t, c) = 1 - P(+ | t, c) = \frac{e^{-t \cdot c}}{1+e^{-t \cdot c}}$

So far, we've been assuming we have a single context word.



- What if we're considering a window containing multiple context words?
 - Simplifying assumption: **All context words are independent**
 - So, we can just multiply their probabilities:
 - $P(+|t,c_{1:k}) = \prod_{i=1}^k \frac{1}{1+e^{-t \cdot c_i}}$, or
 - $\log P(+|t,c_{1:k}) = \sum_{i=1}^k \log \frac{1}{1+e^{-t \cdot c_i}}$



With this in mind....

this	sunday,	watch	the	super	bowl	at	5:30	p.m.
	c1	c2	t	c3	c4			
	$P(+ super, watch) = .7$	$P(+ super, the) = .5$		$P(+ super, bowl) = .9$	$P(+ super at) = .5$			

$$P(+|t, c_{1:k}) = .7 * .5 * .9 * .5 = .1575$$

- Given t and a context window of k words $c_{1:k}$, we can assign a probability based on how similar the context window is to the target word
- We do so by applying the logistic function to the dot product of the embeddings of t with each context word c

Computing $P(+ | t, c)$ and $P(- | t, c)$: ✓

- However, we still have some unanswered questions....
 - **How do we determine our input vectors?**
 - **How do we learn word embeddings** throughout this process (this is the real goal of training our classifier in the first place)?

Input Vectors

- Input words are typically represented as **one-hot vectors**
 - **Bag-of-words approach:** Place a “1” in the position corresponding to a given word, and a “0” in every other position

super

0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

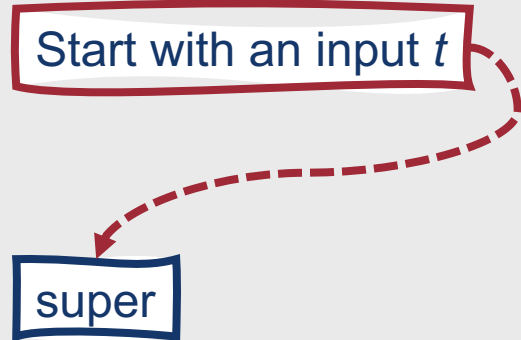
bowl

0	0	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

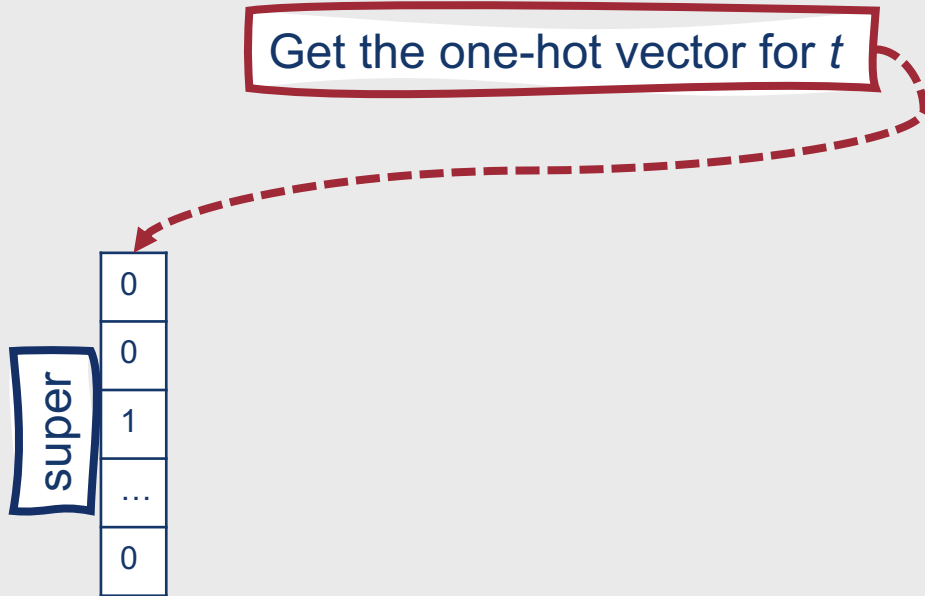
Learned Embeddings

- Embeddings are the weights learned for a two-layer classifier that predicts $P(+ | t, c)$
- Recall from our discussion of logistic regression:
 - $y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-w \cdot x + b}}$
- This is quite similar to the probability we're trying to optimize:
 - $P(+ | t, c) = \frac{1}{1+e^{-t \cdot c}}$

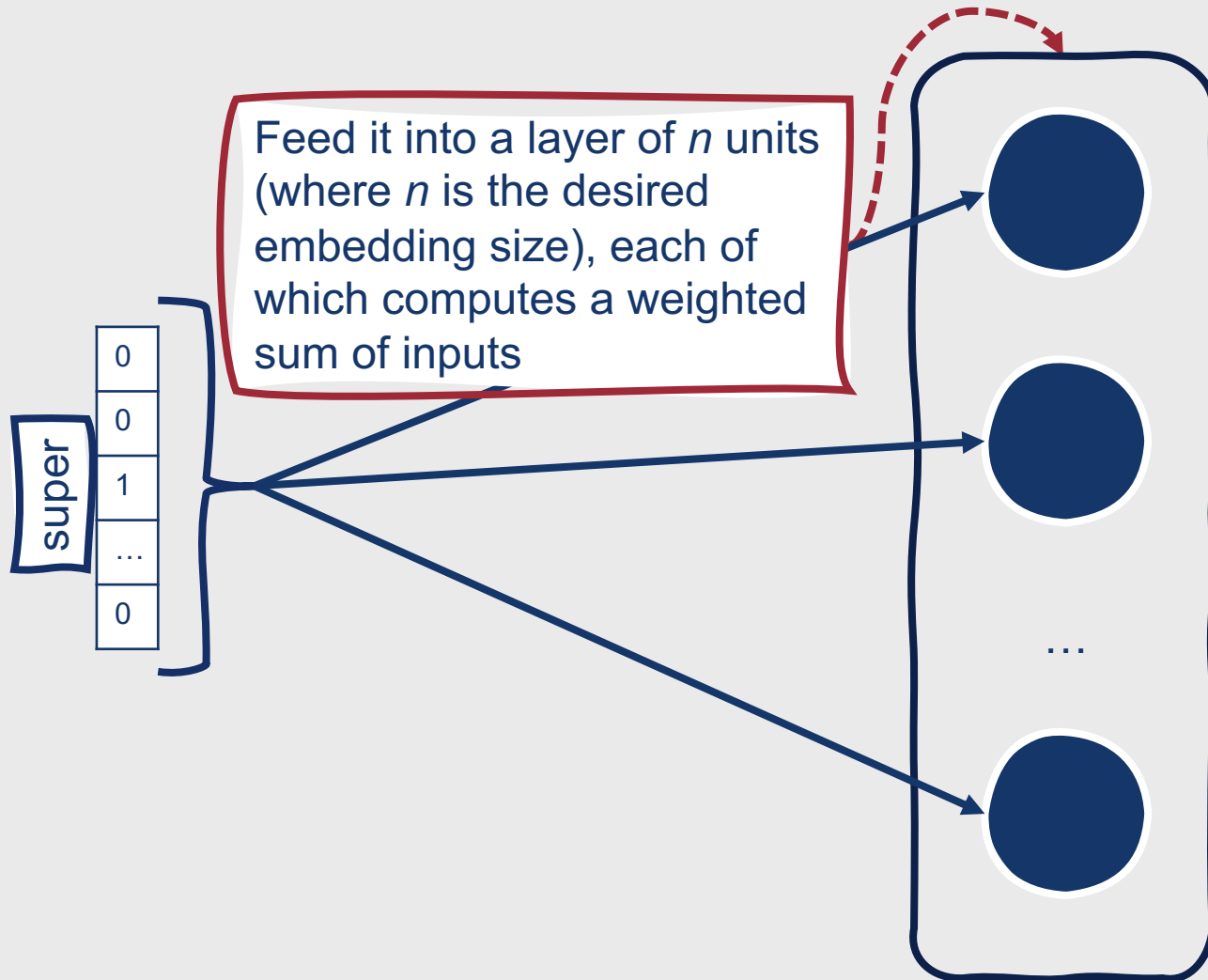
What does this look like?



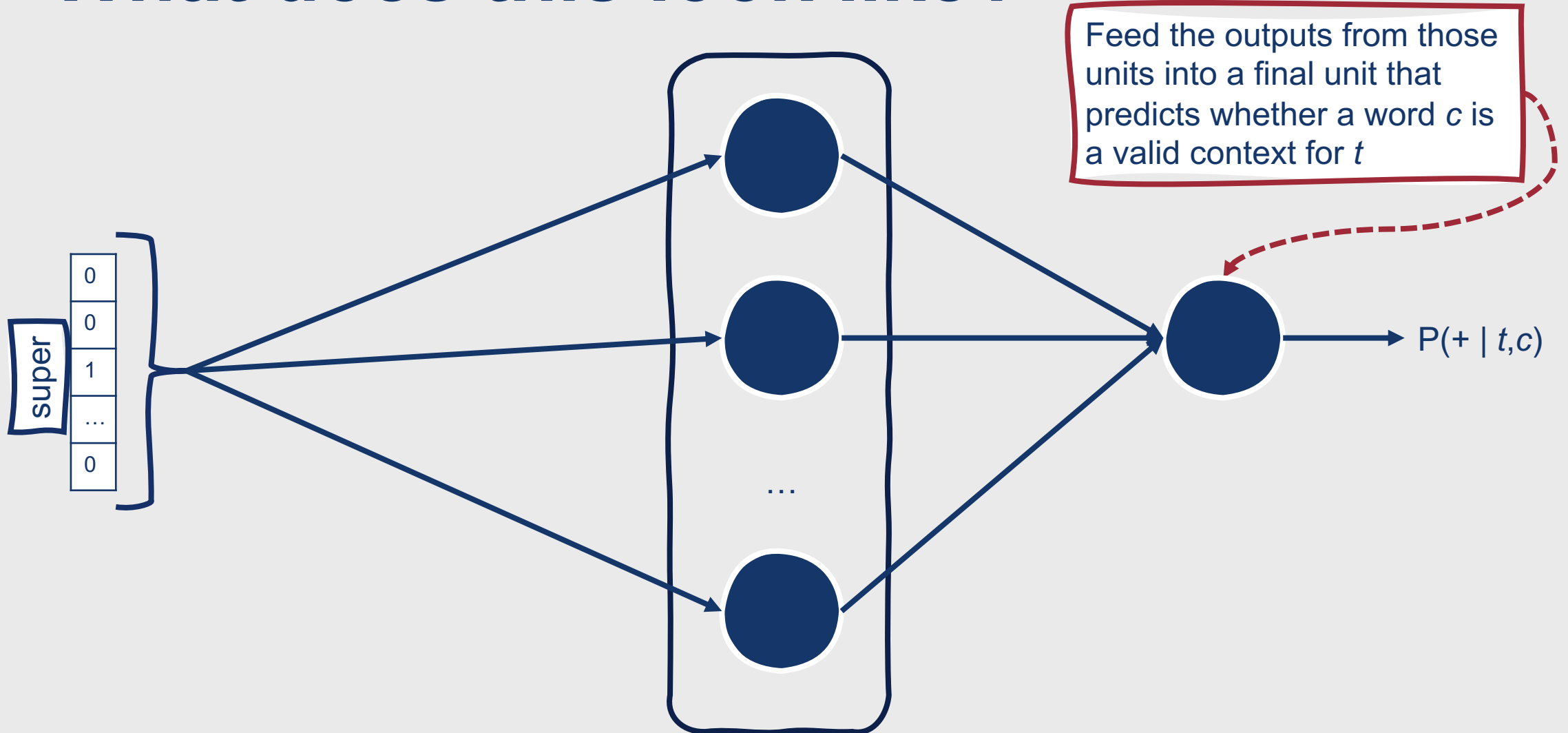
What does this look like?



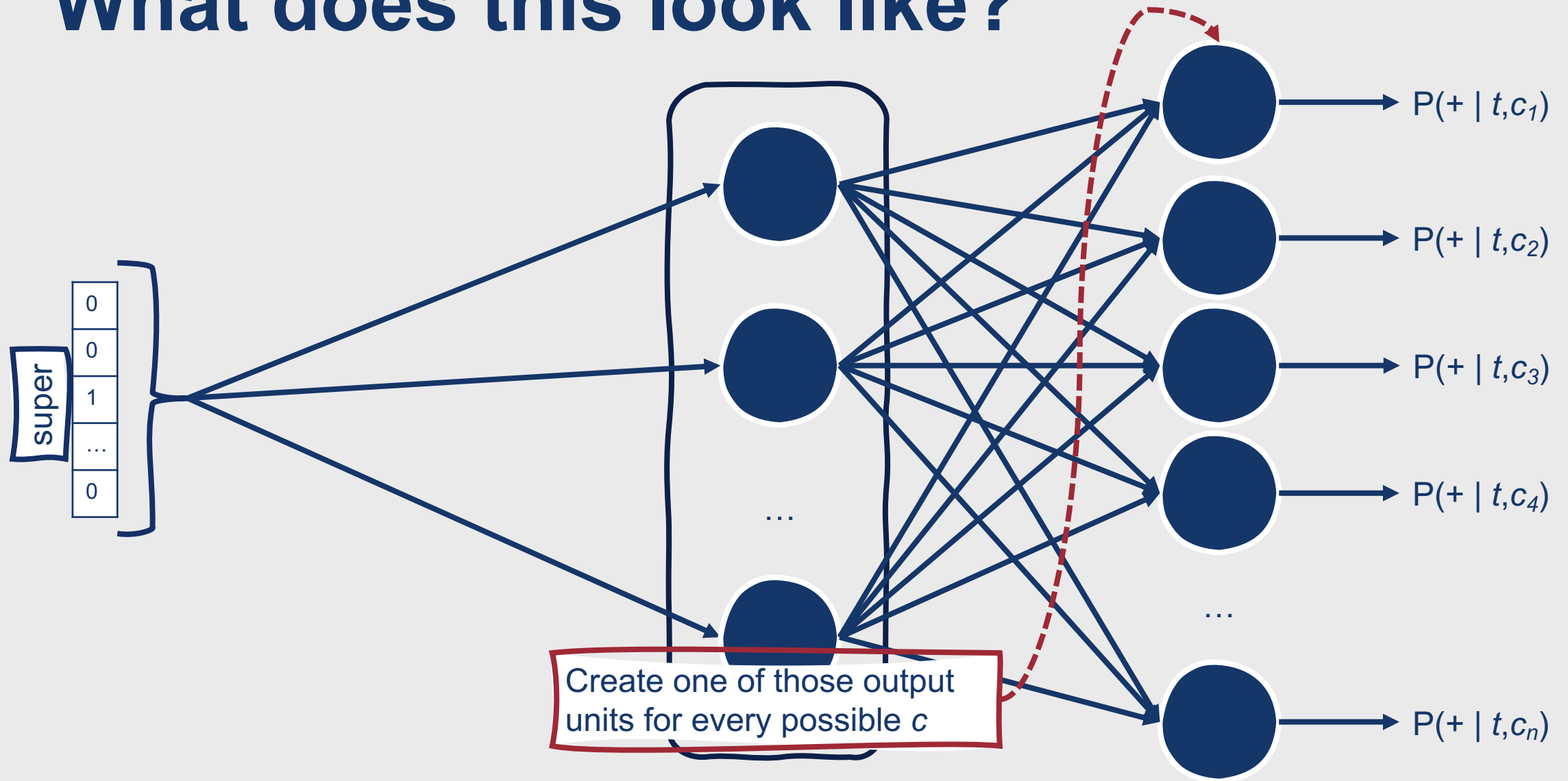
What does this look like?



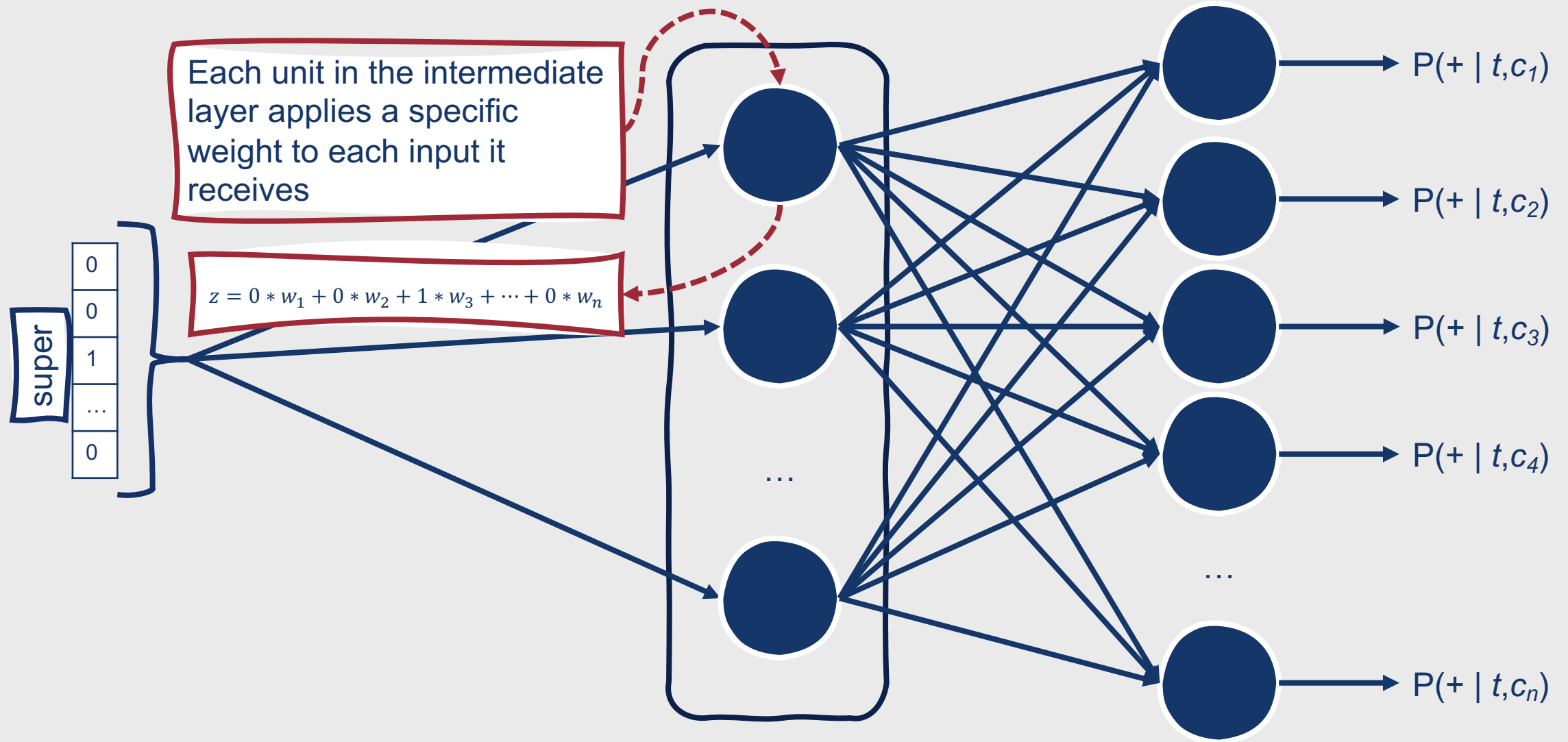
What does this look like?



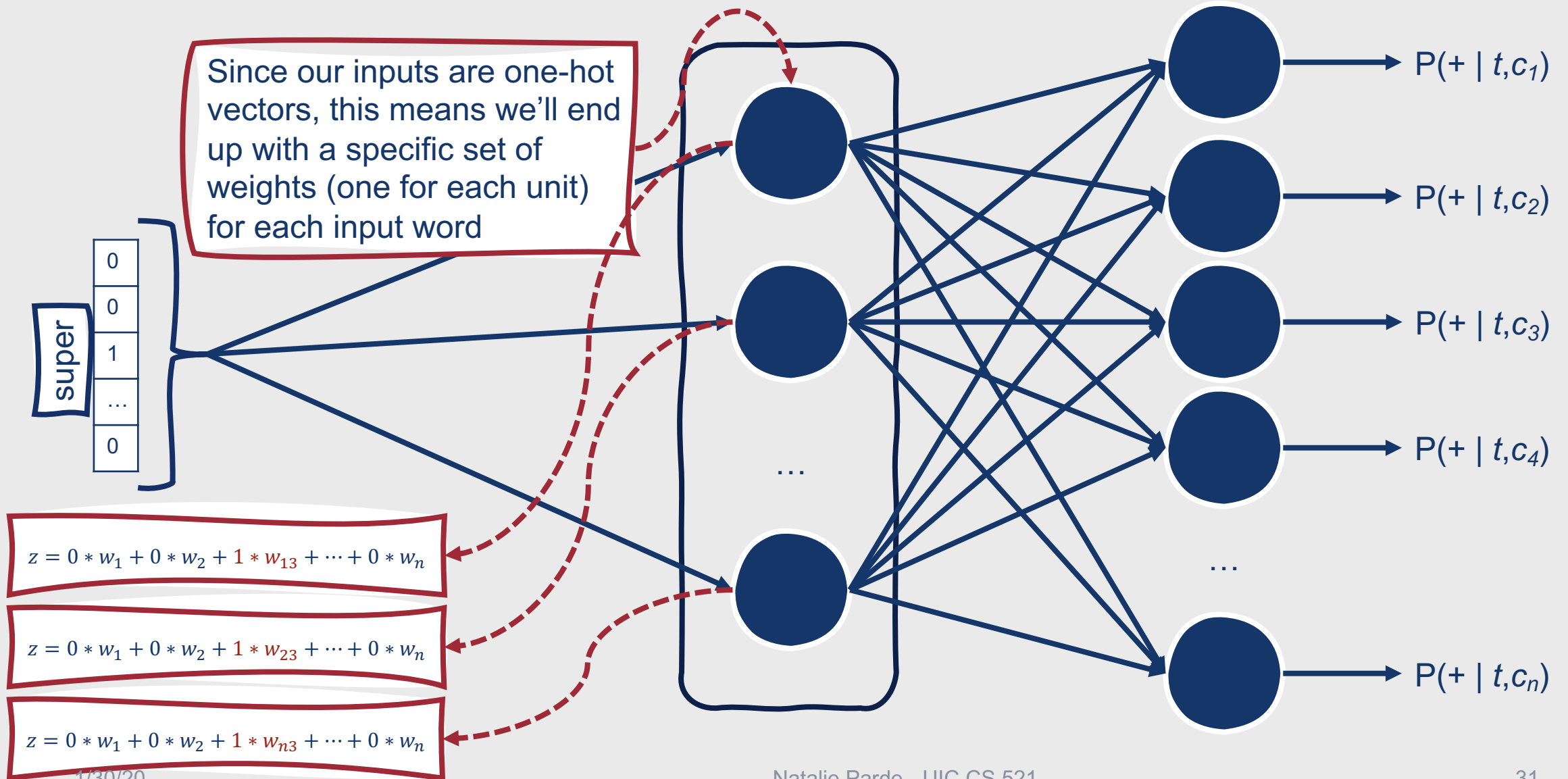
What does this look like?



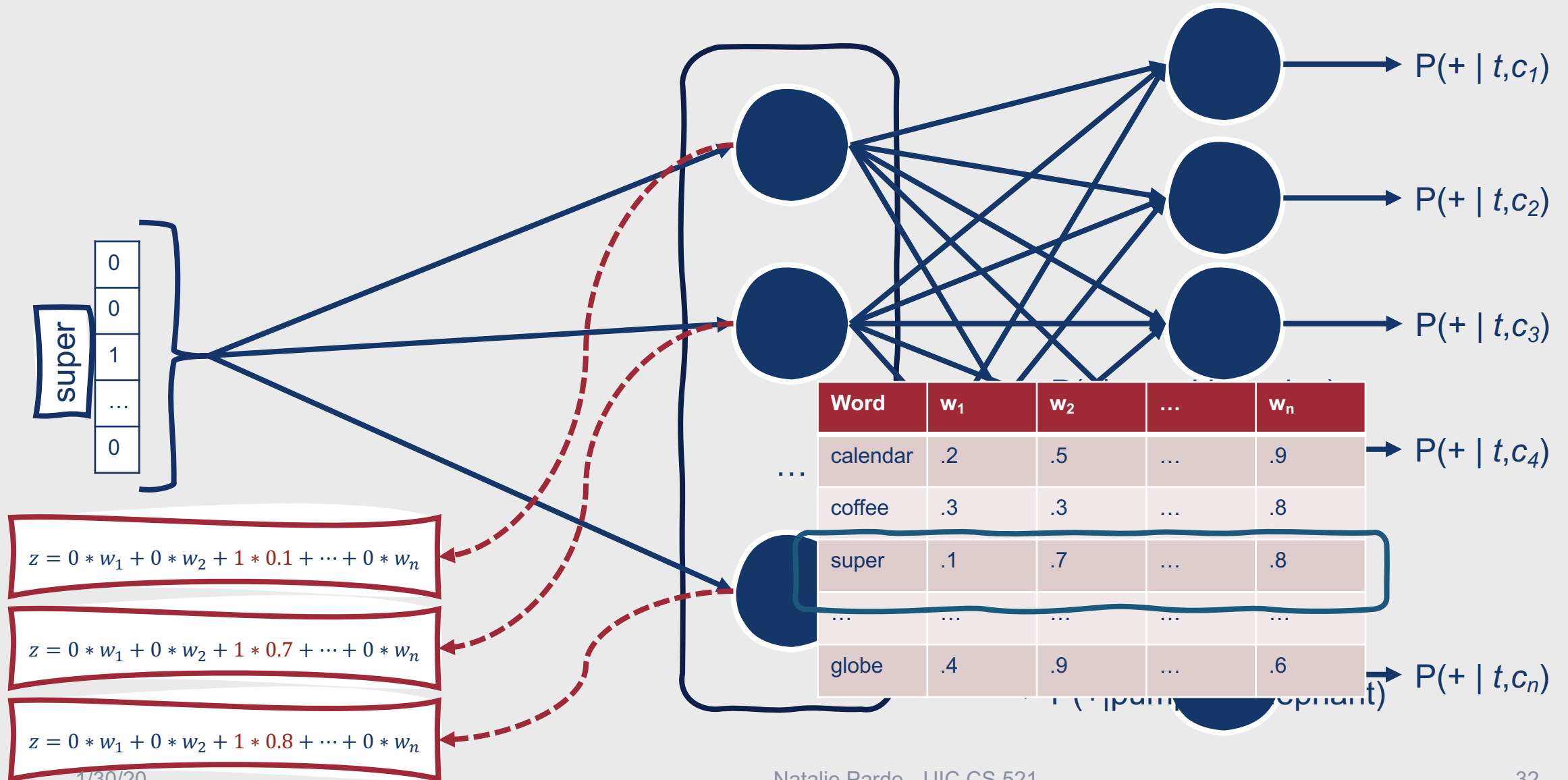
Behind the scenes....



Behind the scenes....



These are the weights we're interested in!

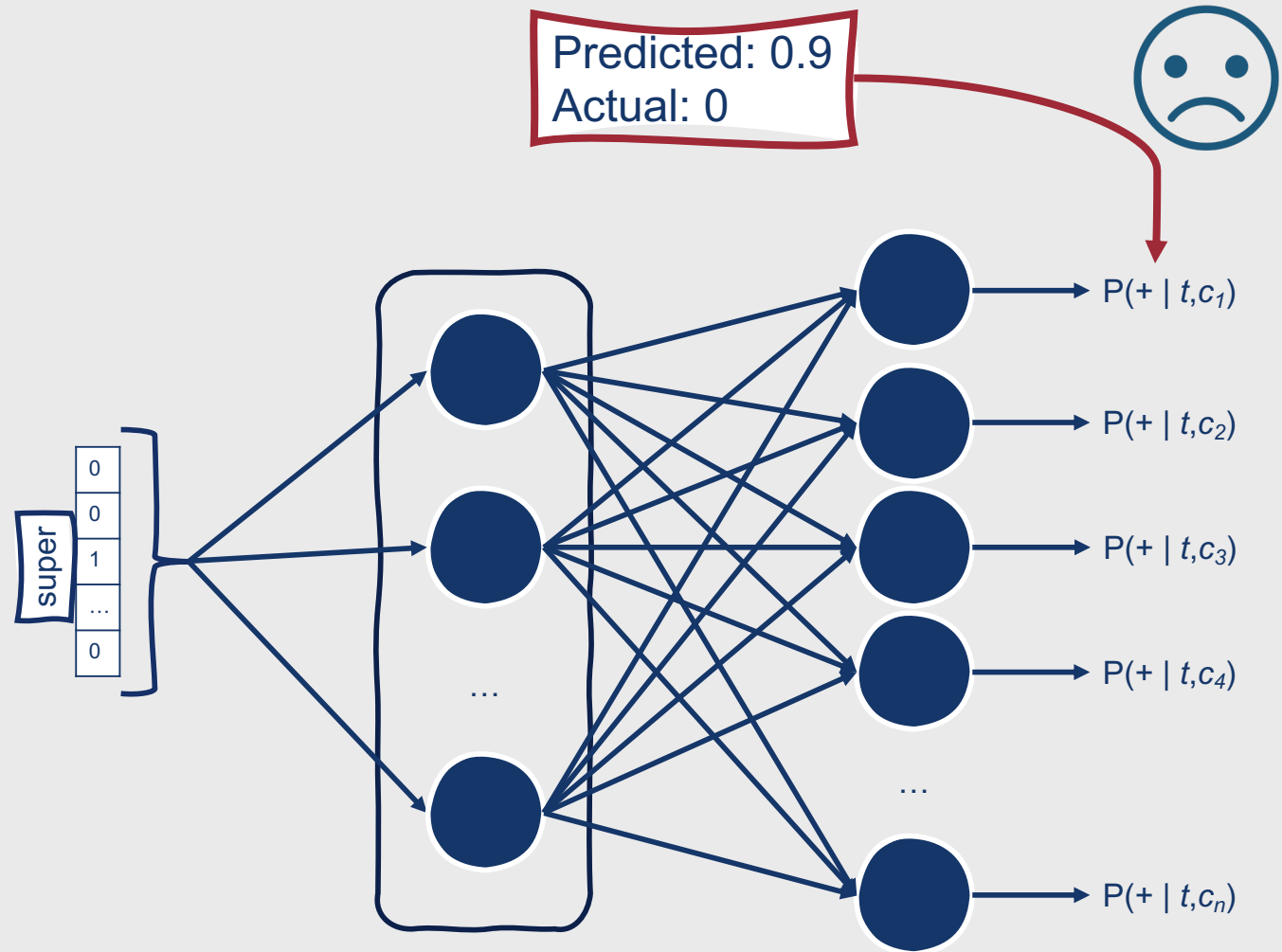




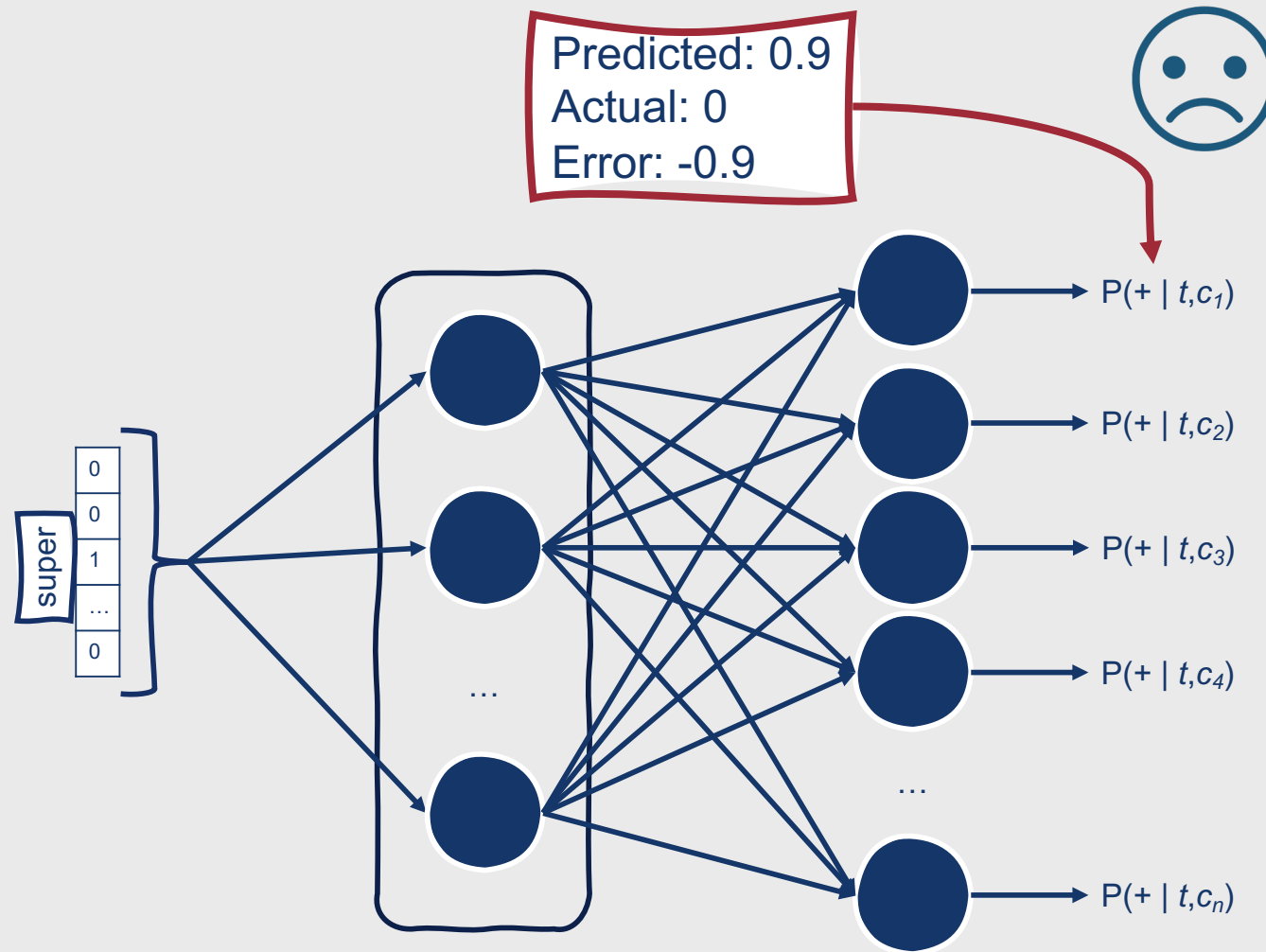
How do we optimize these weights over time?

- The weights are **initialized to some random value** for each word
- They are then iteratively updated to be more similar for words that occur in similar contexts in the training set, and less similar for words that do not
 - Specifically, we want to find weights that maximize $P(+|t,c)$ for words that occur in similar contexts and minimize $P(+|t,c)$ for words that do not, given the information we have at the time
- Note that throughout this process, we're actually maintaining two sets of hidden weight vectors
 - One for the input (the target words)
 - One for the output (the context words)

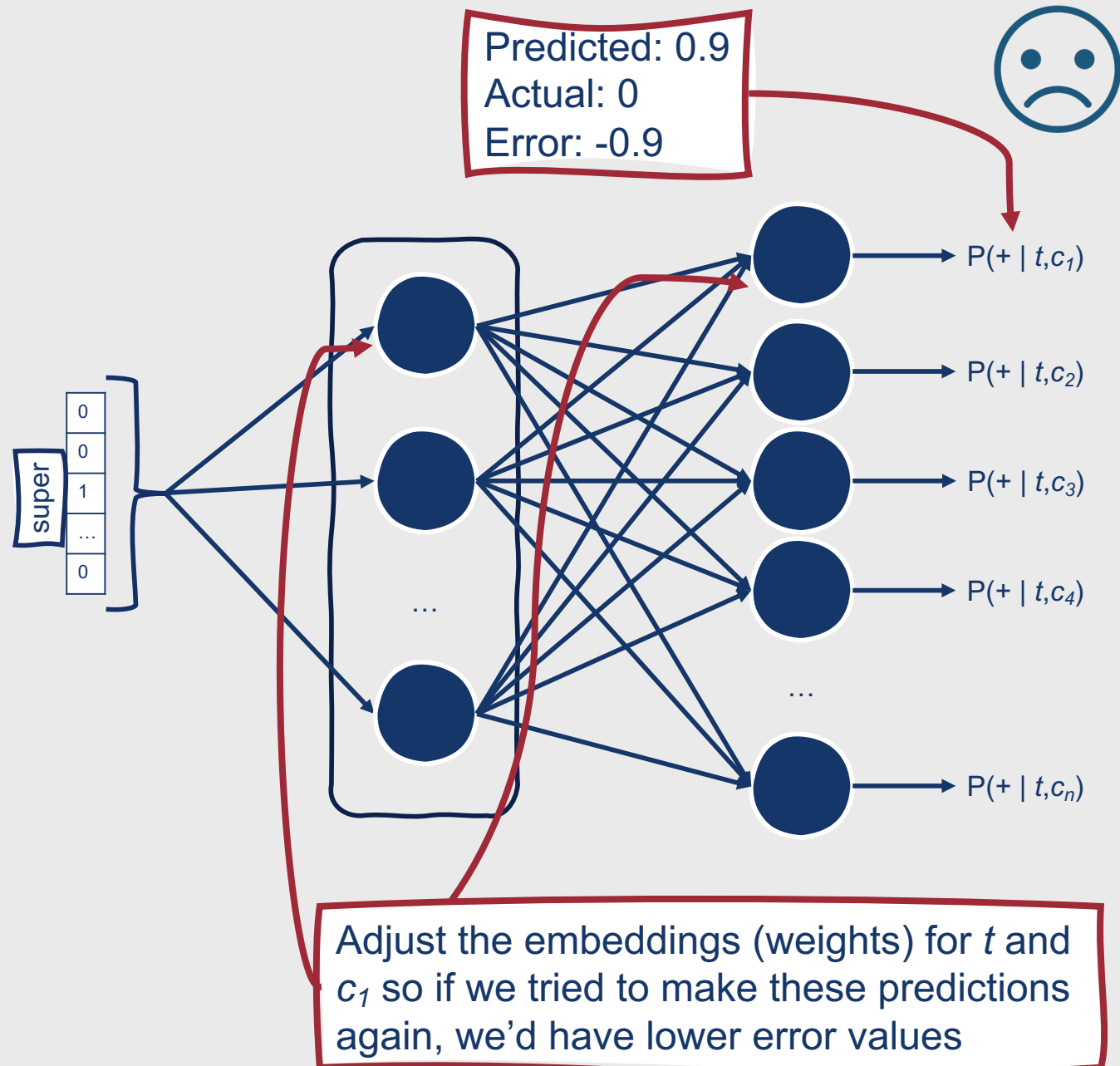
Since we initialize our weights randomly, the classifier's first prediction will almost certainly be wrong.



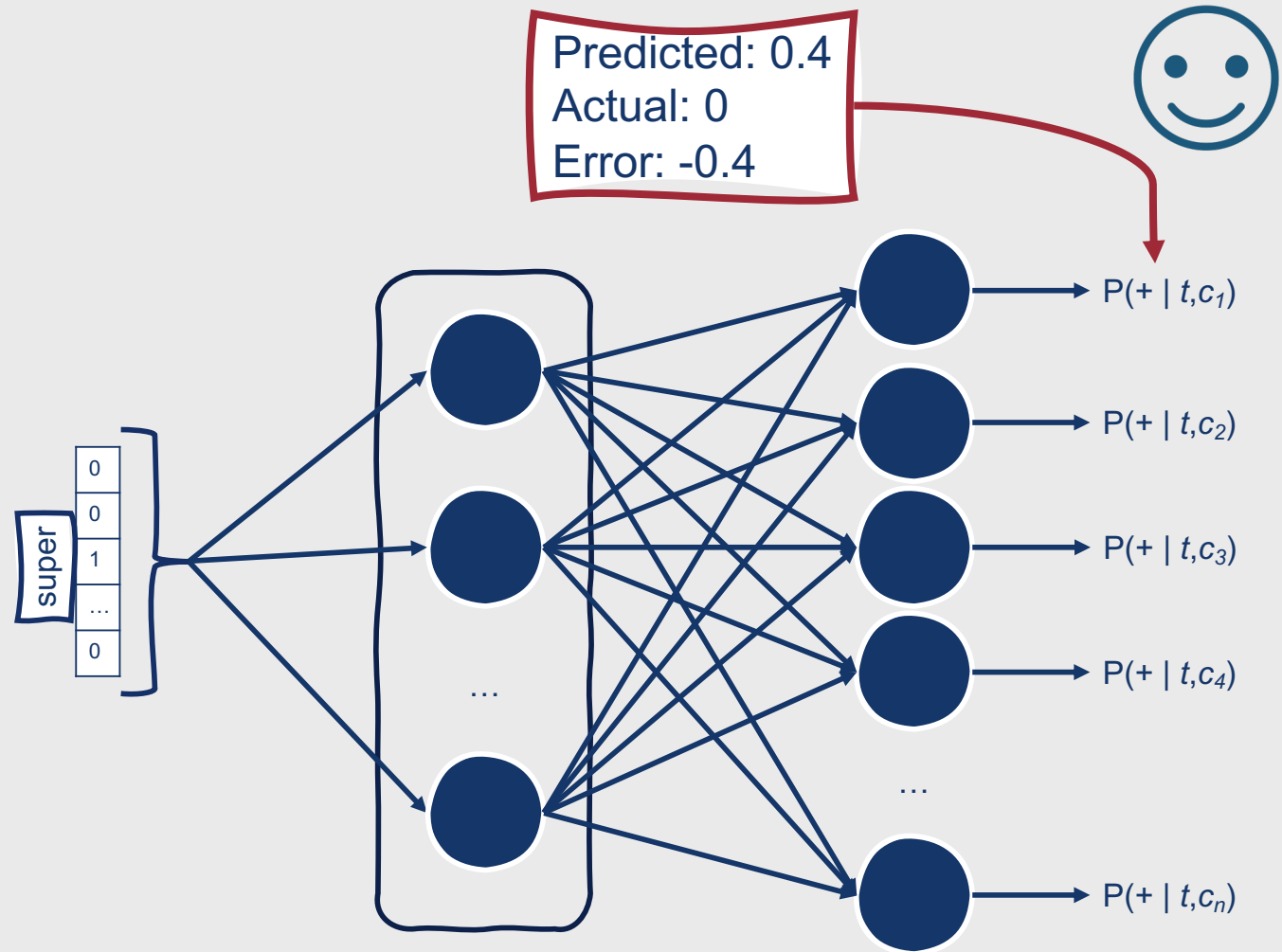
However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.





Training Data

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- We are able to assume that all occurrences of words in similar contexts in our training corpus are **positive samples**



Training Data

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- However, we also need negative samples!
- In fact, Word2Vec uses more negative than positive samples (the exact ratio can vary)
- We need to create our own negative examples



Training Data

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- How to create negative examples?
 - Target word + “noise” word that is sampled from the training set
 - Noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight:
 - $$p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha}$$



Training Data

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

Negative Examples

t	c
super	calendar
super	exam
super	loud
super	bread
super	cellphone
super	enemy
super	penguin
super	drive

- How to create negative examples?
 - Often, $\alpha = 0.75$ to give rarer noise words slightly higher probability of being randomly sampled
- Assuming we want twice as many negative samples as positive samples, we can thus randomly select noise words according to weighted unigram frequency



Learning Skip-Gram Embeddings

- It is with these samples that the algorithm:
 - Maximizes the similarity of the (target, context) pairs drawn from positive examples
 - Minimizes the similarity of the (target, context) pairs drawn from negative examples
- It does so by applying stochastic gradient descent to optimize a cross-entropy loss function
 - The target and context weight vectors are the parameters being tuned

Learning Skip-Gram Embeddings

Even though we're maintaining two embeddings for each word during training (the target vector and the context vector), we only need one of them

When we're finished learning the embeddings, we can just discard the context vector

Alternately, we can add them together to create a **summed embedding** of the same dimensionality, or we can **concatenate them into a longer embedding** with twice as many dimensions



Context window size can impact performance!

- Because of this, context window size is often tuned on a development set
- Larger window size → more required computations (important to consider when using very large datasets)



What if we want to predict a target word from a set of context words instead?

- **Continuous Bag of Words (CBOW)**
 - Another variation of Word2Vec
- Very similar to skip-gram model!
- The difference:
 - Instead of learning to predict a context word from a target word vector, you learn to **predict a target word from a set of context word vectors**

Skip-Gram vs. CBOW Embeddings

In general, skip-gram embeddings are good with:

- Small datasets
- Rare words and phrases

CBOW embeddings are good with:

- Larger datasets (they're faster to train)
- Frequent words

Are there any other variations of Word2Vec?

- **fasttext**
 - An extension of Word2Vec that also incorporates **subword models**
 - Designed to better handle unknown words and sparsity in language

fasttext

- Each word is represented as:
 - Itself
 - A bag of constituent n-grams

`super` = `<super>` + `<su, sup, upe, per, er>`



fasttext

- Skip-gram embedding is learned for each constituent n-gram
- Word is represented by the sum of all embeddings of its constituent n-grams
- Key advantage of this extension?
 - Allows embeddings to be predicted for unknown words based on subword constituents alone

Source code available online:
<https://fasttext.cc/>

Word2Vec and fasttext embeddings are nice ...but what's another alternative?

- Word2Vec is an example of a **predictive** word embedding model
 - Learns to predict whether words belong in a target word's context
- Other models are **count-based**
 - Remember co-occurrence matrices?
- GloVE combines aspects of both predictive and count-based models





Global Vectors for Word Representation (GloVe)

- Co-occurrence matrices quickly grow extremely large
- Intuitive solution to increase scalability?
 - Dimensionality reduction!
 - However, typical dimensionality reduction strategies may result in too much computational overhead
- GloVe learns to predict weights in a lower-dimensional space that correspond to the co-occurrence probabilities between words

GloVe

- Why is this useful?
 - Predictive models → black box
 - They work, but why?
 - GloVe models are easier to interpret
- GloVe models also encode the ratios of co-occurrence probabilities between different words ...this makes these vectors useful for word analogy tasks

How does GloVe work?

	c_1	...	c_n
t_1	123	...	456
...
t_n	0	...	789

Build a huge word-context
co-occurrence matrix

How does GloVe work?

	c_1	...	c_n
t_1	123	...	456
...
t_n	0	...	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

Scaler biases for t_i and c_j

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Vector for t_i

Vector for c_j

Co-occurrence count for $t_i c_j$

How does GloVe work?

	c_1	...	c_n
t_1	123	...	456
...
t_n	0	...	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^V \sum_{j=1}^V f(X_{ij}) (w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Weighting function:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{max}}\right)^\alpha, & X_{ij} < XMAX \\ 1, & \text{otherwise} \end{cases}$$

How does GloVe work?

	c_1	...	c_n
t_1	123	...	456
...
t_n	0	...	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^V \sum_{j=1}^V \underbrace{f(X_{ij})}_{\text{weight}} (w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Minimize the cost function to learn ideal embedding values for w_i and w_j

How does GloVe work?

	c_1	...	c_n
t_1	123	...	456
...
t_n	0	...	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^V \sum_{j=1}^V \underbrace{f(X_{ij})}_{\text{weight}} (w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

0.4 0.7 1.2 4.3 0.9 6.7 1.3 0.5 0.7 5.3

Minimize the cost function to learn ideal embedding values for w_i and w_j



In sum, GloVe is a log-bilinear model with a weighted least-squares objective.

- Why does it work?
 - Ratios of co-occurrence probabilities have the potential to encode word similarities and differences
 - These similarities and differences are useful components of meaning
- GloVe embeddings perform particularly well on analogy tasks



Which is best ... Word2Vec or GloVe?

- It depends on your data!
- In general, Word2Vec and GloVe produce similar embeddings
- Word2Vec → slower to train but less memory intensive
- GloVe → faster to train but more memory intensive
- As noted earlier, Word2Vec and GloVe both produce context-independent embeddings

Evaluating Vector Models

Extrinsic Evaluation

- Add the vectors as features in a downstream NLP task, and see whether and how this changes performance relative to a baseline model
- Most important evaluation metric for word embeddings!
 - Word embeddings are rarely needed in isolation
 - They are almost solely used to boost performance in downstream tasks

Intrinsic Evaluation

- Performance at predicting word similarity



Evaluating Performance at Predicting Word Similarity

- Compute the **cosine similarity** between vectors for pairs of words
- Compute the **correlation** between those similarity scores and word similarity ratings for the same pairs of words manually assigned by humans
- Corpora for doing this:
 - WordSim-353
 - SimLex-999
 - TOEFL Dataset
 - *Levied* is closest in meaning to: (a) imposed, (b) believed, (c) requested, (d) correlated

Other Common Evaluation Tasks

Semantic Textual Similarity

- Evaluates the performance of sentence-level similarity algorithms, rather than word-level similarity

Analogy

- Evaluates the performance of algorithms at solving analogies
 - Athens is to Greece as Oslo is to (Norway)
 - Mouse is to mice as dollar is to (dollars)



Semantic Properties of Embeddings

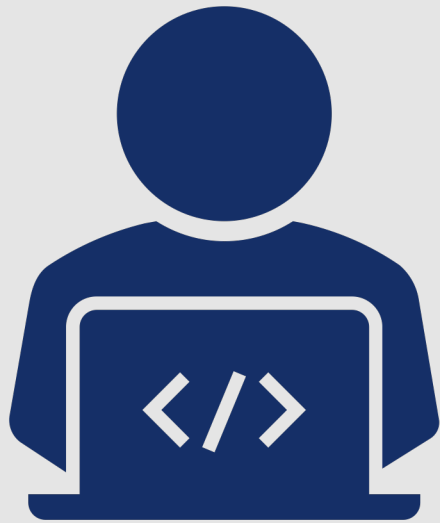
- Major advantage of dense word embeddings: Ability to capture elements of meaning
- Context window size impacts what type of meaning is captured
 - Shorter context window → more **syntactic representations**
 - Information is from immediately nearby words
 - Most similar words tend to be semantically similar words with the same parts of speech
 - Longer context window → more **topical representations**
 - Information can come from longer-distance dependencies
 - Most similar words tend to be topically related, but not necessarily similar (e.g., waiter and menu, rather than spoon and fork)



Analogy

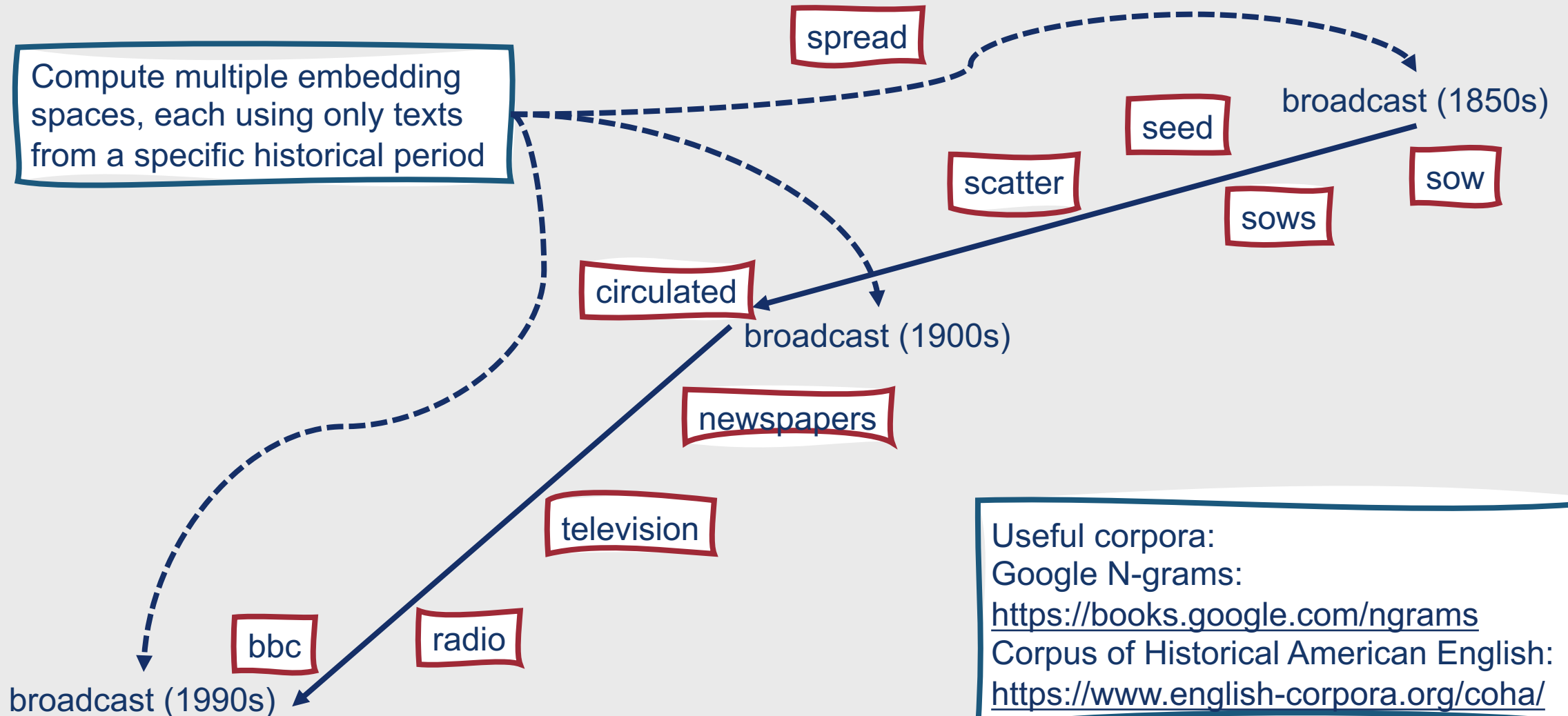
- Word embeddings can also capture **relational meanings**
- This is done by computing the offsets between values in the same columns for different vectors
- Famous examples (Mikolov et al., 2013; Levy and Goldberg, 2014):
 - king - man + woman = queen
 - Paris - France + Italy = Rome

Word embeddings have many practical applications.



- Incorporated as features in nearly every modern NLP task
- Useful for computational social science
 - Studying word meaning over time
 - Studying implicit associations between words

Embeddings and Historical Semantics



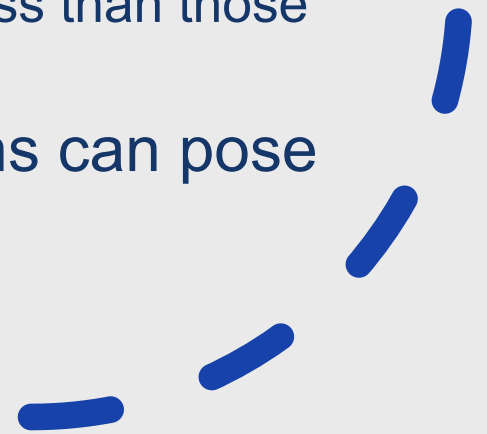
Unfortunately, word embeddings can also end up reproducing implicit biases and stereotypes latent in text.



- Recall: king - man + woman = queen
- Word embeddings trained on news corpora also produce:
 - man - computer programmer + woman = homemaker
 - doctor - father + mother = nurse
- Very problematic for real-world applications (e.g., resume scoring models)

Bias and Embeddings

- Caliskan et al. (2017) identified the following known, harmful implicit associations in GloVe embeddings:
 - African-American names were more closely associated with unpleasantness than European-American names
 - Male names were more closely associated with mathematics than female names
 - Female names were more closely associated with the arts than male names
 - Names common among older adults were more closely associated with unpleasantness than those common among younger adults
- Thus, learning word representations can pose ethical dilemmas!



How do we keep the useful associations present in word embeddings, but get rid of the harmful ones?

- Recent research has begun examining ways to **debias** word embeddings by:
 - Developing transformations of embedding spaces that remove gender stereotypes but preserve definitional gender
 - Changing training procedures to eliminate these issues before they arise
- Although these methods reduce bias, they do not eliminate it
- Increasingly active area of study:
 - <https://fatconference.org/2020/>



Summary: Word2Vec and GloVe

- **Word2Vec** is a **predictive** word embedding approach that learns word representations by training a classifier to predict whether a **context word** should be associated with a given **target word**
- **Fasttext** is an extension of Word2Vec that also incorporates **subword models**
- **GloVe** is a **count-based** word embedding approach that learns an optimized, lower-dimensional version of a co-occurrence matrix
- Word embedding models are best evaluated extrinsically, in downstream tasks, but can also be evaluated based on their ability to predict **word similarity** or solve **analogies**
- Dense word embeddings encode many interesting semantic properties
 - Positive: Useful **word associations** and **synonymy**
 - Negative: **Biases** and **stereotypes**
- Developing ways to **debias** word embeddings or avoid bias entirely is an increasingly active area of research