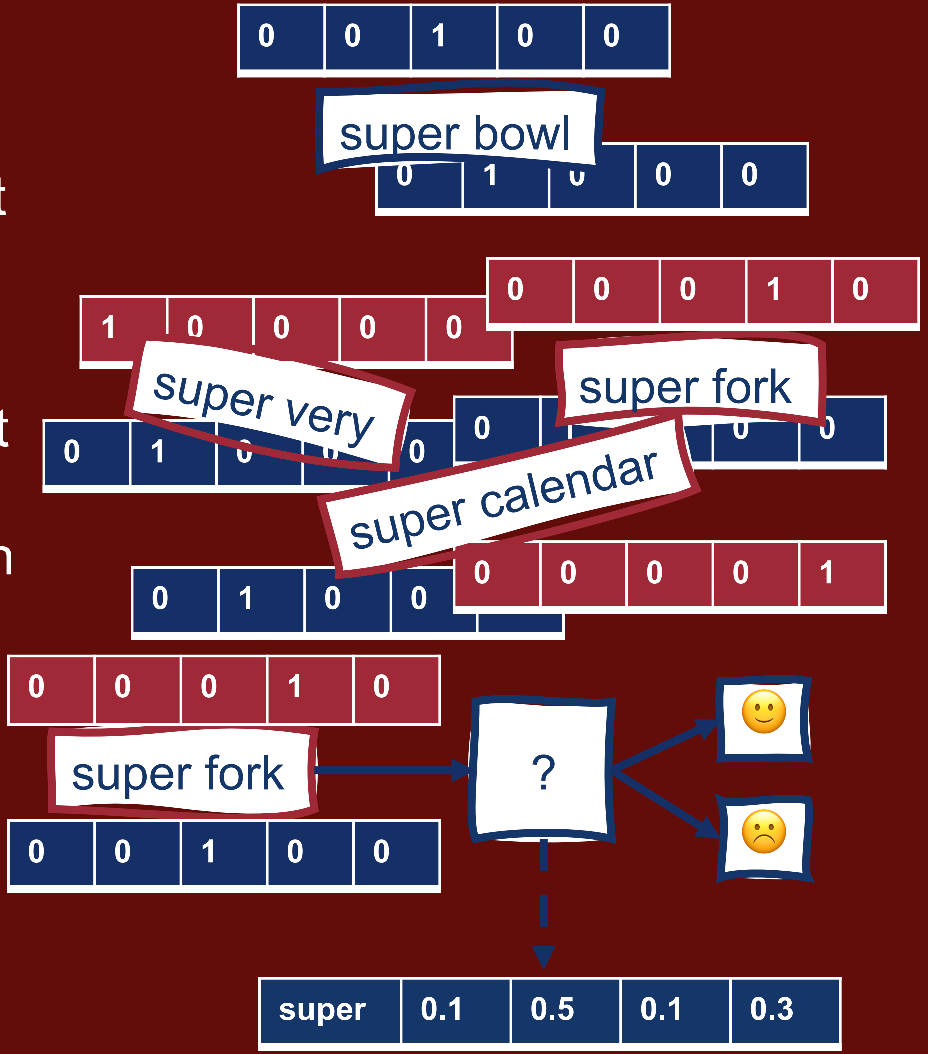
Word2Vec

Natalie Parde
UIC CS 421

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))
- 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
  - 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*) = $\frac{1}{1+e^{-t\cdot c}}$
- Then:
  - P(+ | *t*,*c*) = $\frac{1}{1+e^{-t\cdot c}}$
  - P(- | *t*,*c*) = 1 - P(+ | *t*,*c*) = $\frac{e^{-t\cdot c}}{1+e^{-t\cdot c}}$

We’re usually not just looking at words in isolation.

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

$$\boxed{\begin{aligned} P(+|t,c) &= \frac{1}{1+e^{-t\cdot c}} \\ P(-|t,c) &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}}$$

- 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….

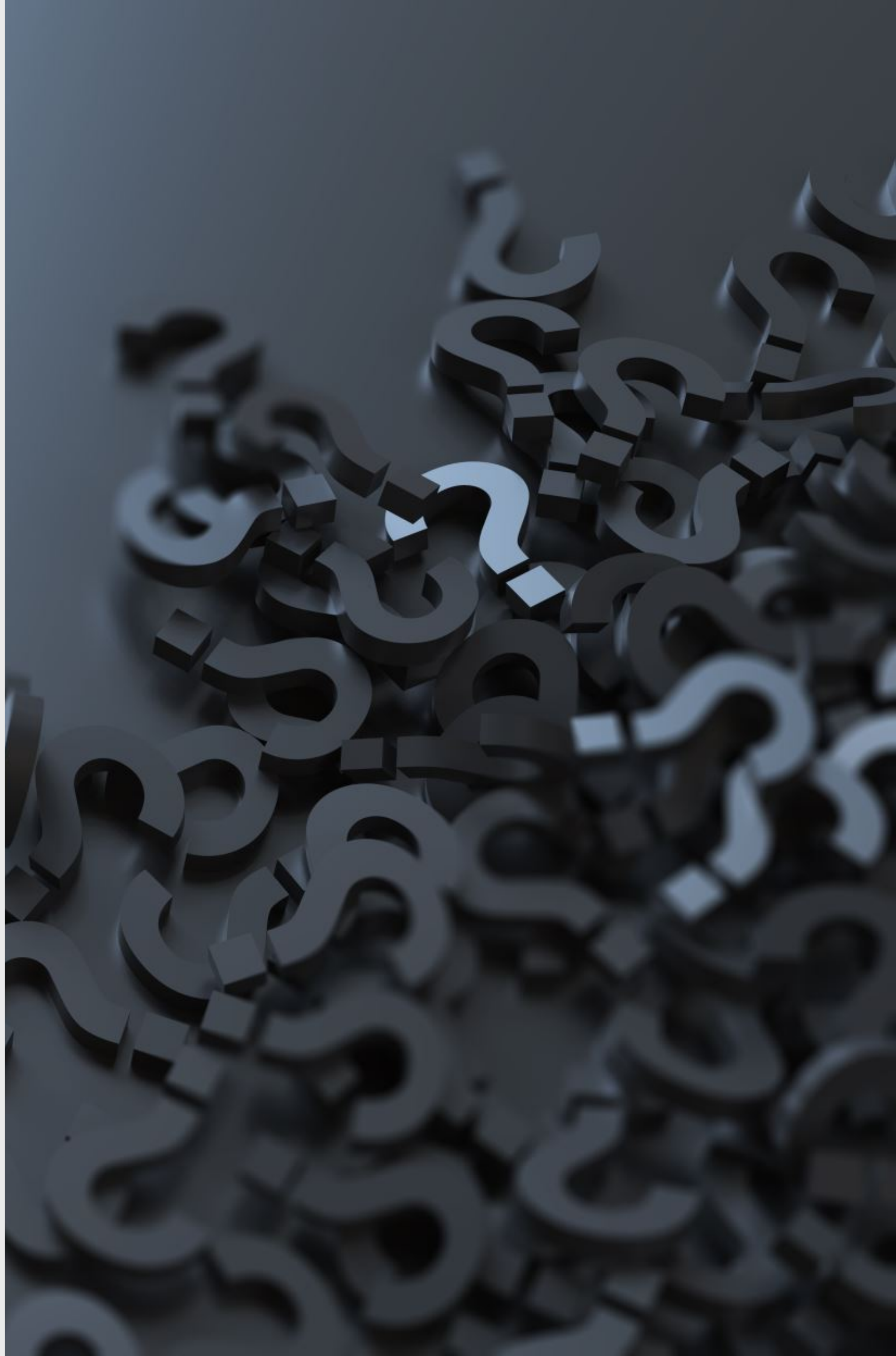
this	sunday,	watch	the	super	bowl	at	5:30	p.m.
		c1	c2	t	c3	c4		
		P(+|*super*, *watch*) = .7	P(+| *super*, *the*) = .5		P(+|*super*, *bowl*) = .9	P(+| *super* *at*) = .5		

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

- 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**
  - **Binary 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….

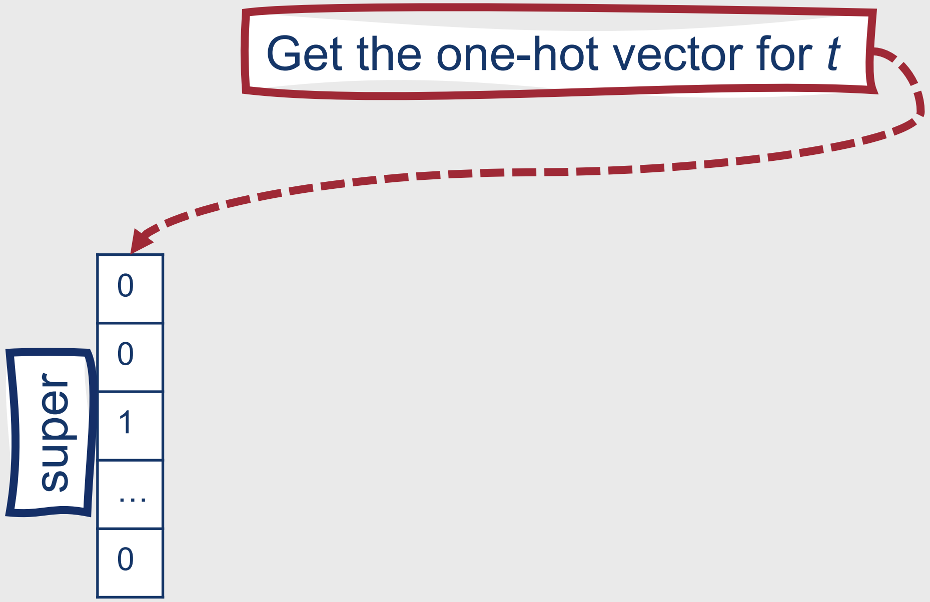
- Embeddings are the weights learned for a two-layer classifier that predicts P(+ | *t*,*c*)
- Recall from our discussion of logistic regression:
  - $y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-w \cdot x+b}}$
- This is quite similar to the probability we're trying to optimize:
  - P(+ | *t*,*c*)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

What does this look like?

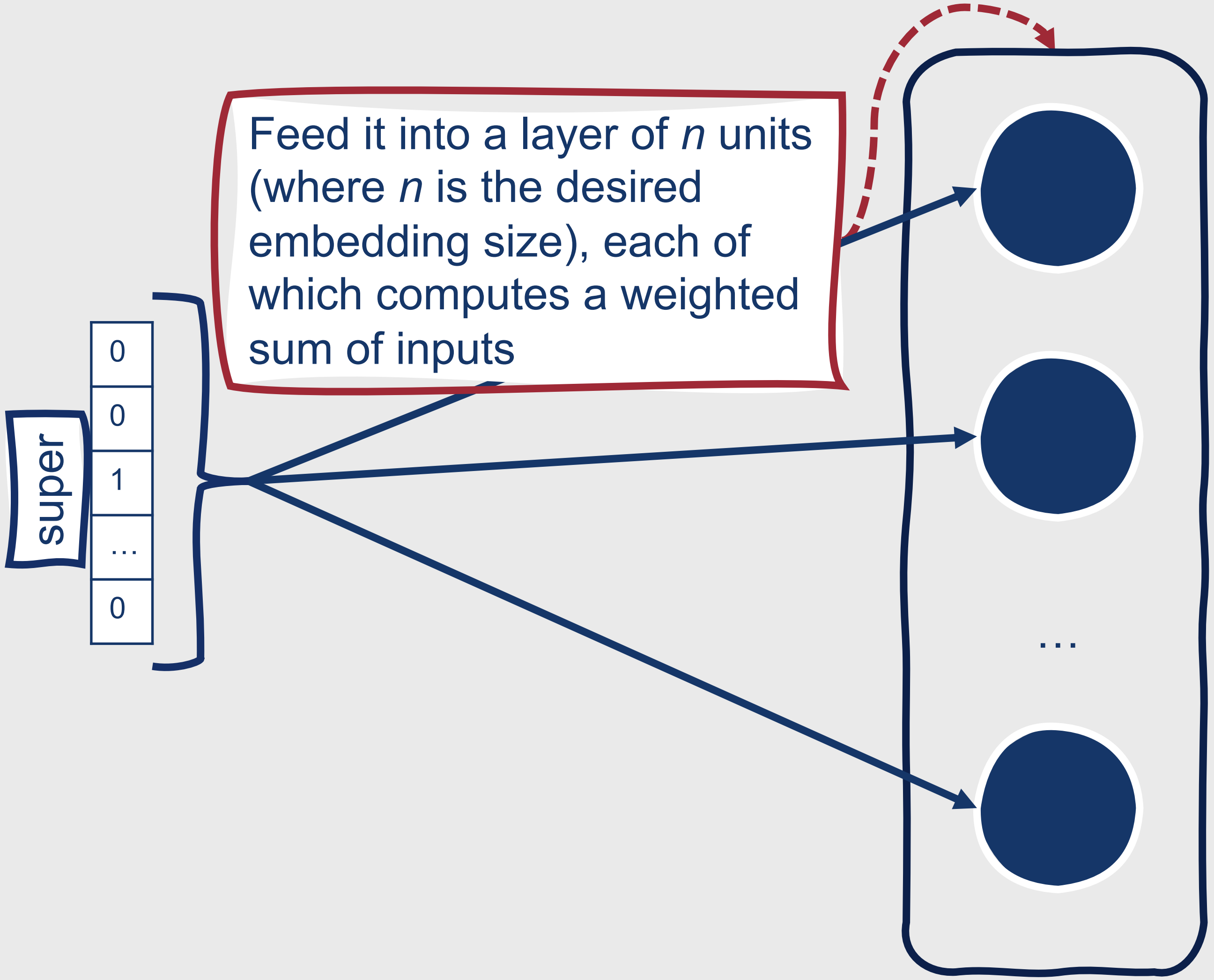
Start with an input *t*

super

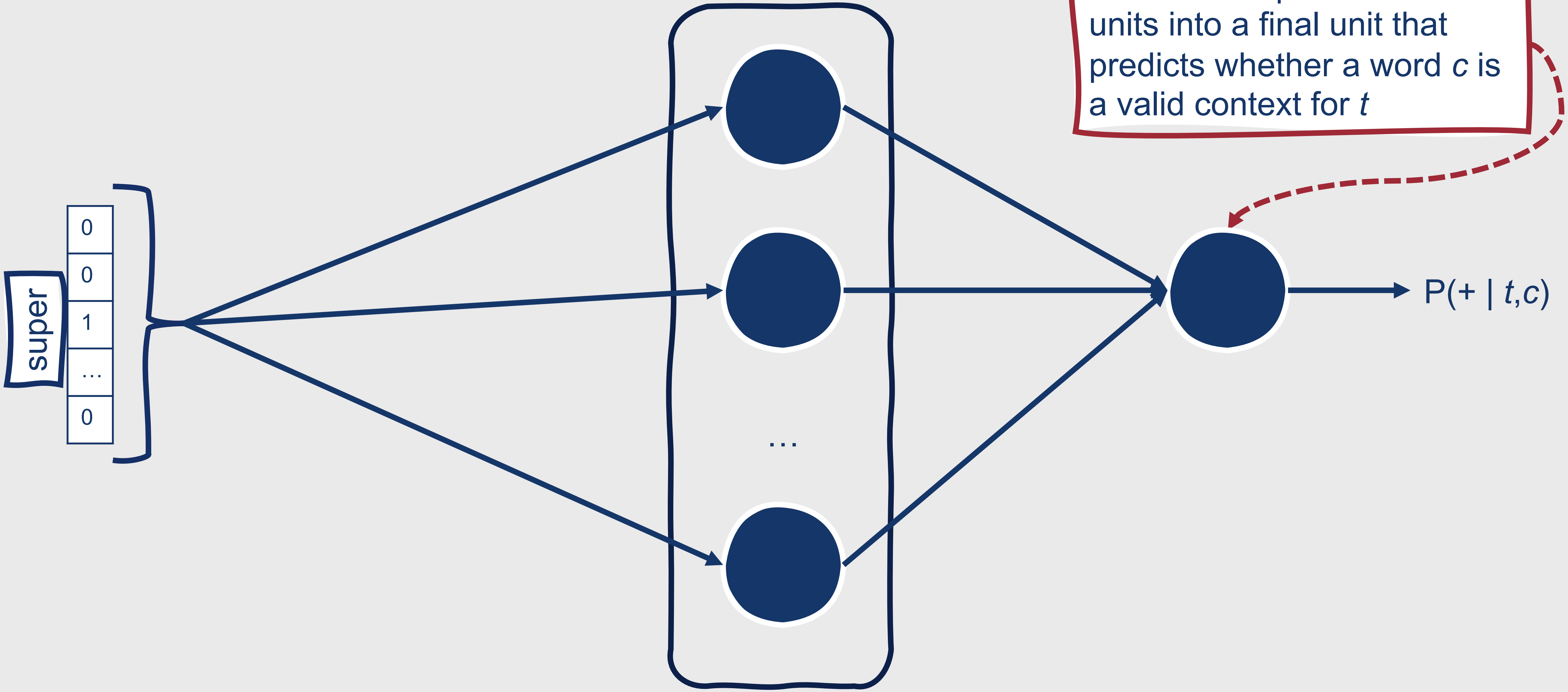
What does this look like?



What does this look like?

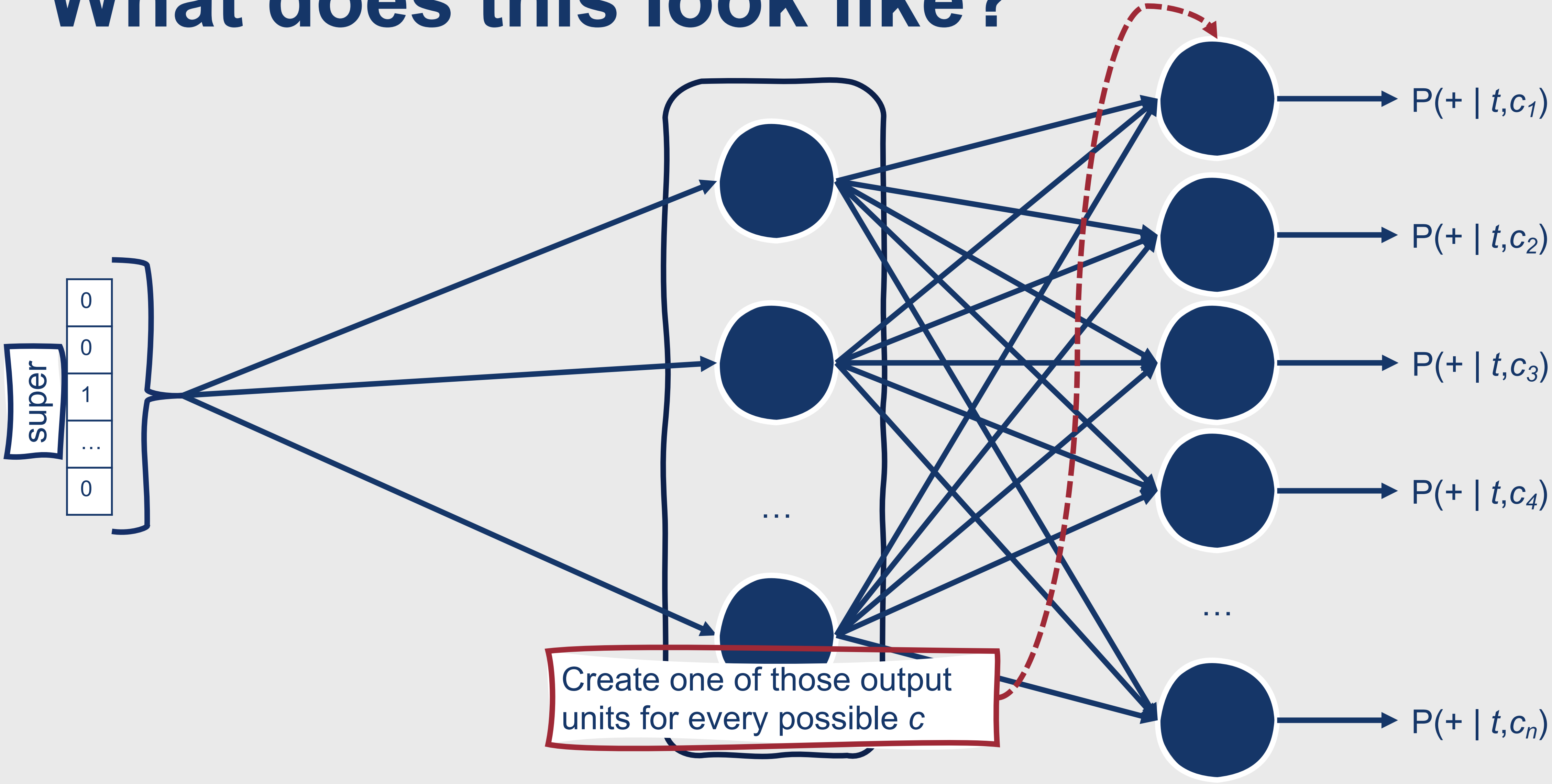


What does this look like?

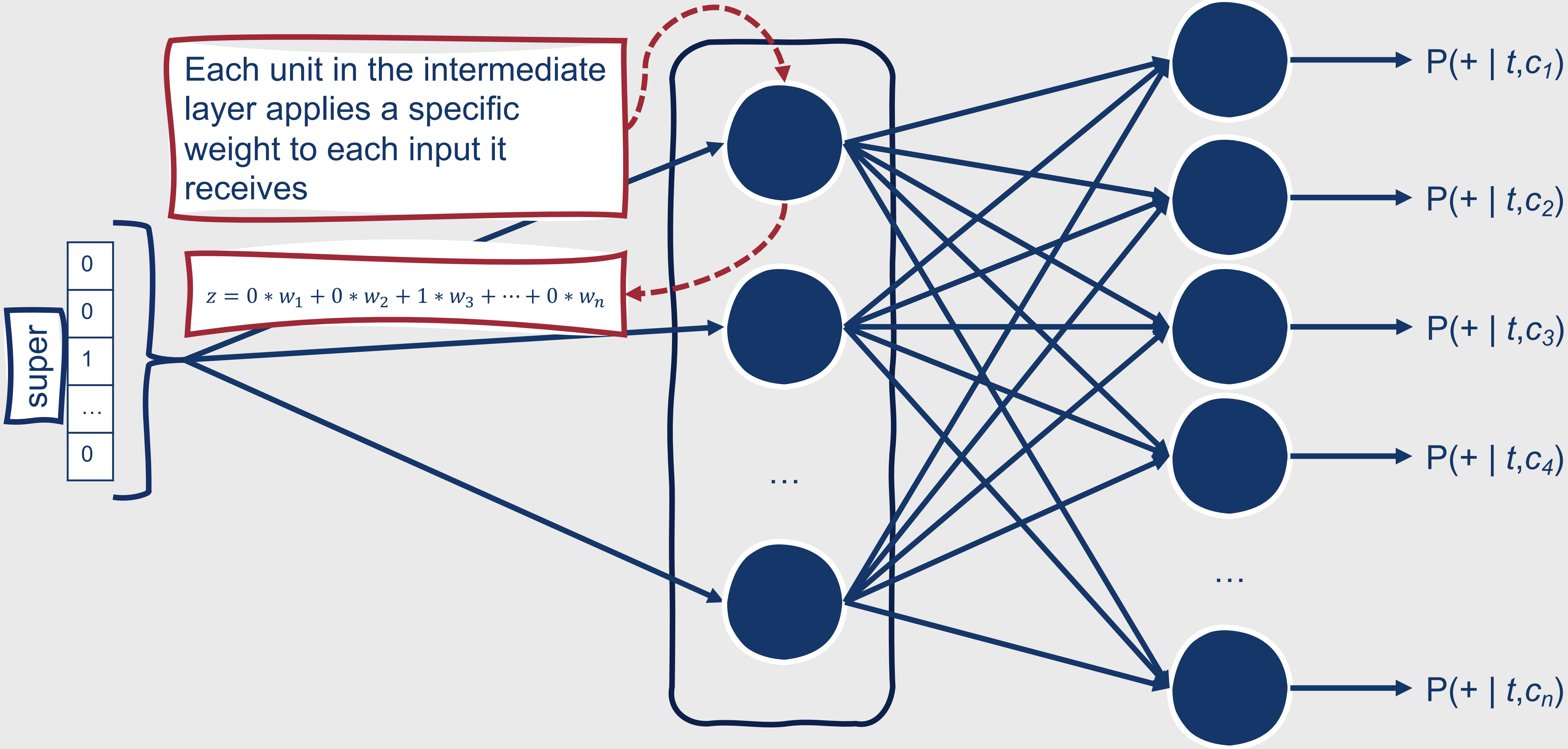


Feed the outputs from those units into a final unit that predicts whether a word *c* is a valid context for *t*

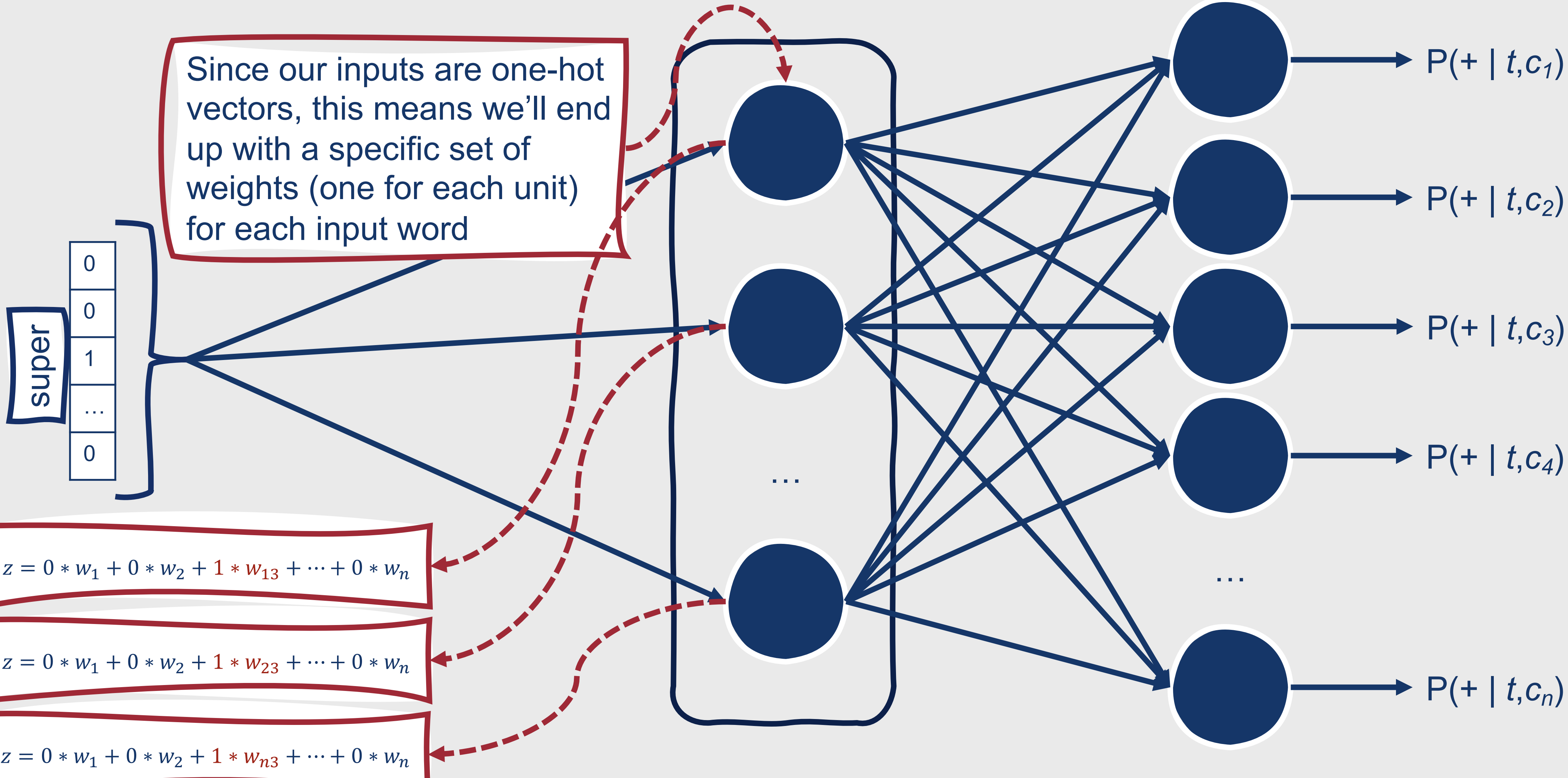
What does this look like?



Behind the scenes….



Behind the scenes….



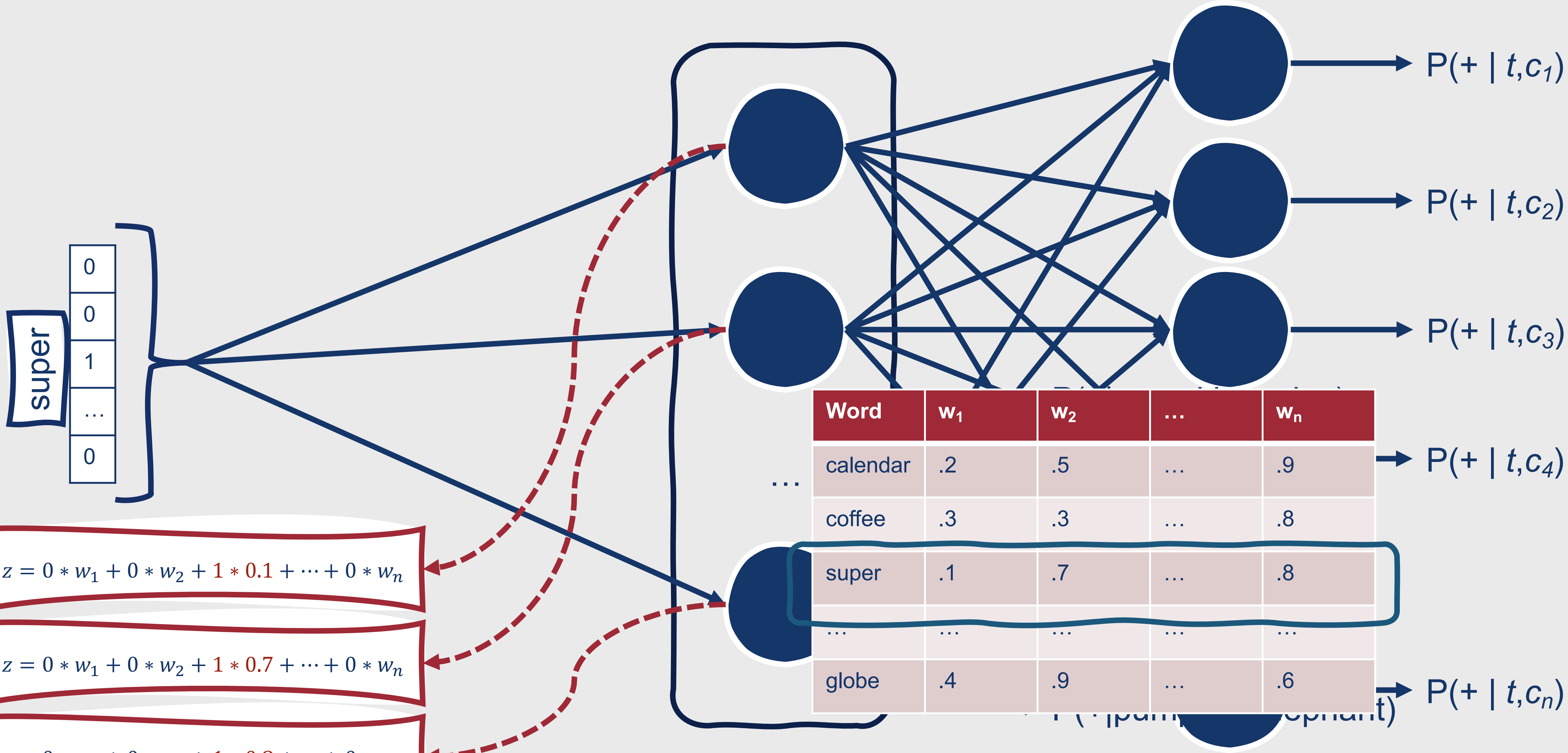
Since our inputs are one-hot vectors, this means we'll end up with a specific set of weights (one for each unit) for each input word

$z = 0 * w_1 + 0 * w_2 + 1 * w_{13} + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * w_{23} + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * w_{n3} + \cdots + 0 * w_n$

These are the weights we're interested in!



Word	w_1	w_2	…	w_n
calendar	.2	.5	…	.9
coffee	.3	.3	…	.8
super	.1	.7	…	.8
…	…	…	…	…
globe	.4	.9	…	.6

$$z = 0 * w_1 + 0 * w_2 + 1 * 0.1 + \cdots + 0 * w_n$$

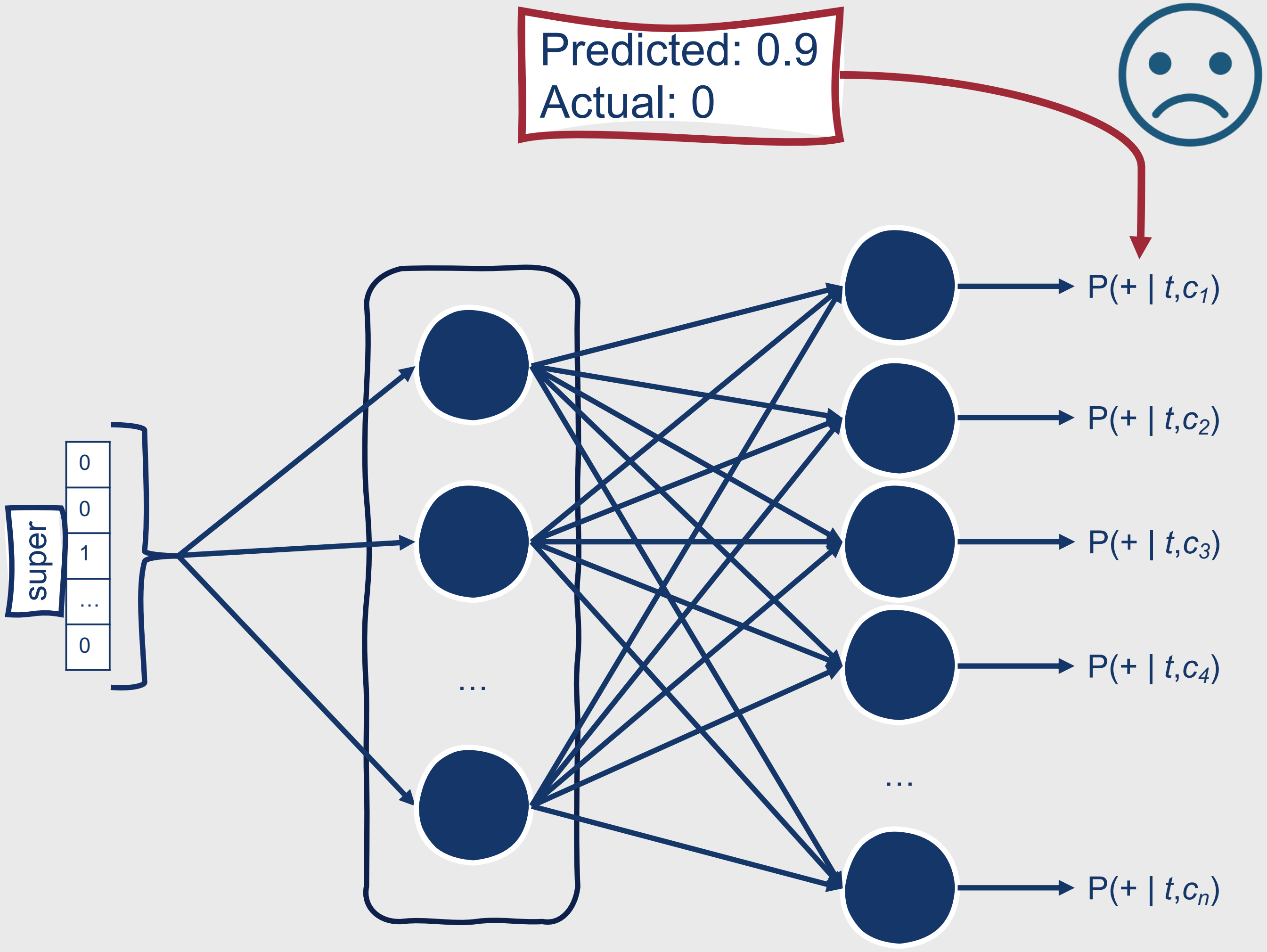
$$z = 0 * w_1 + 0 * w_2 + 1 * 0.7 + \cdots + 0 * w_n$$

$$z = 0 * w_1 + 0 * w_2 + 1 * 0.8 + \cdots + 0 * w_n$$

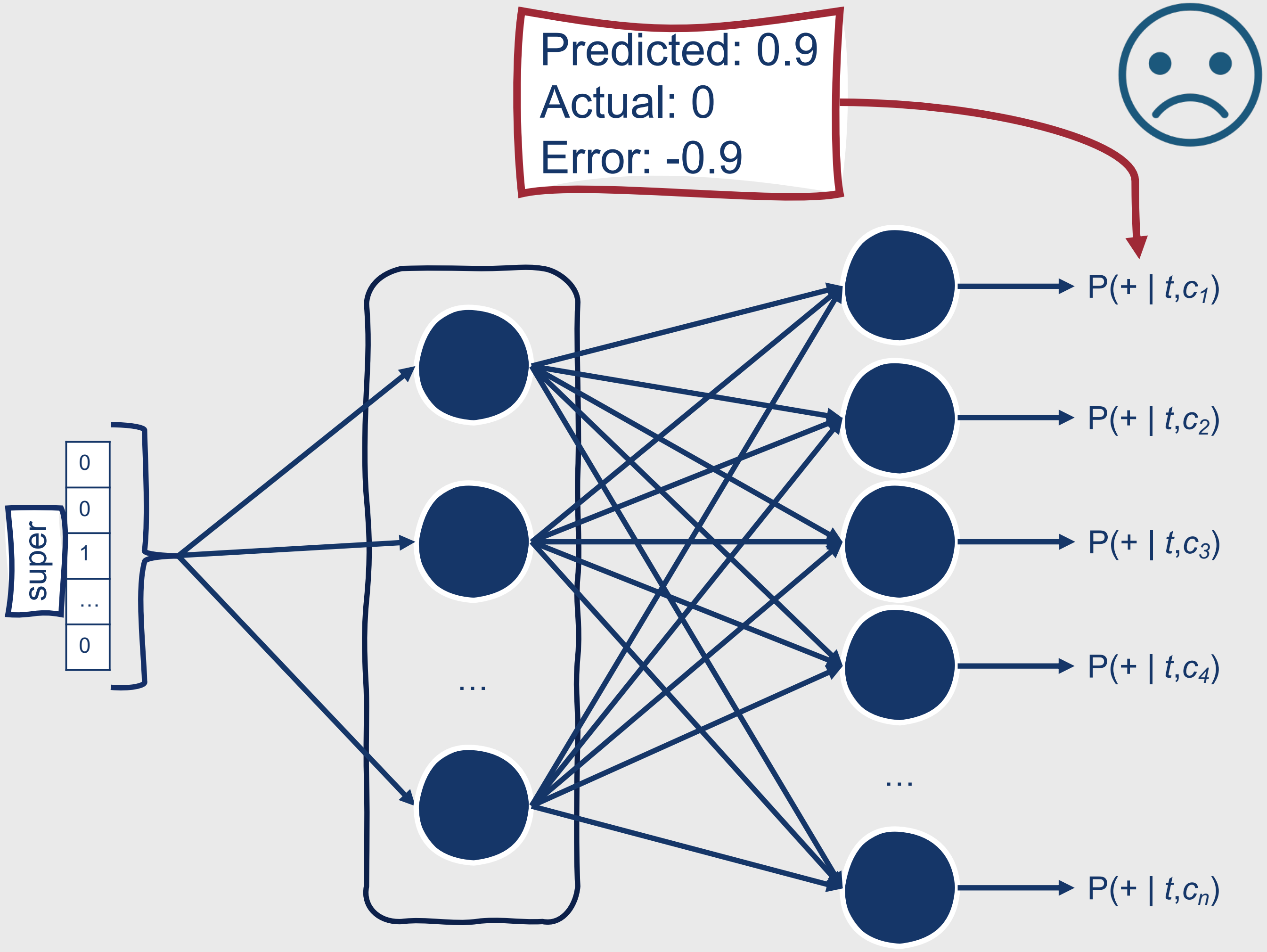
How do we optimize these weights over time?

- 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

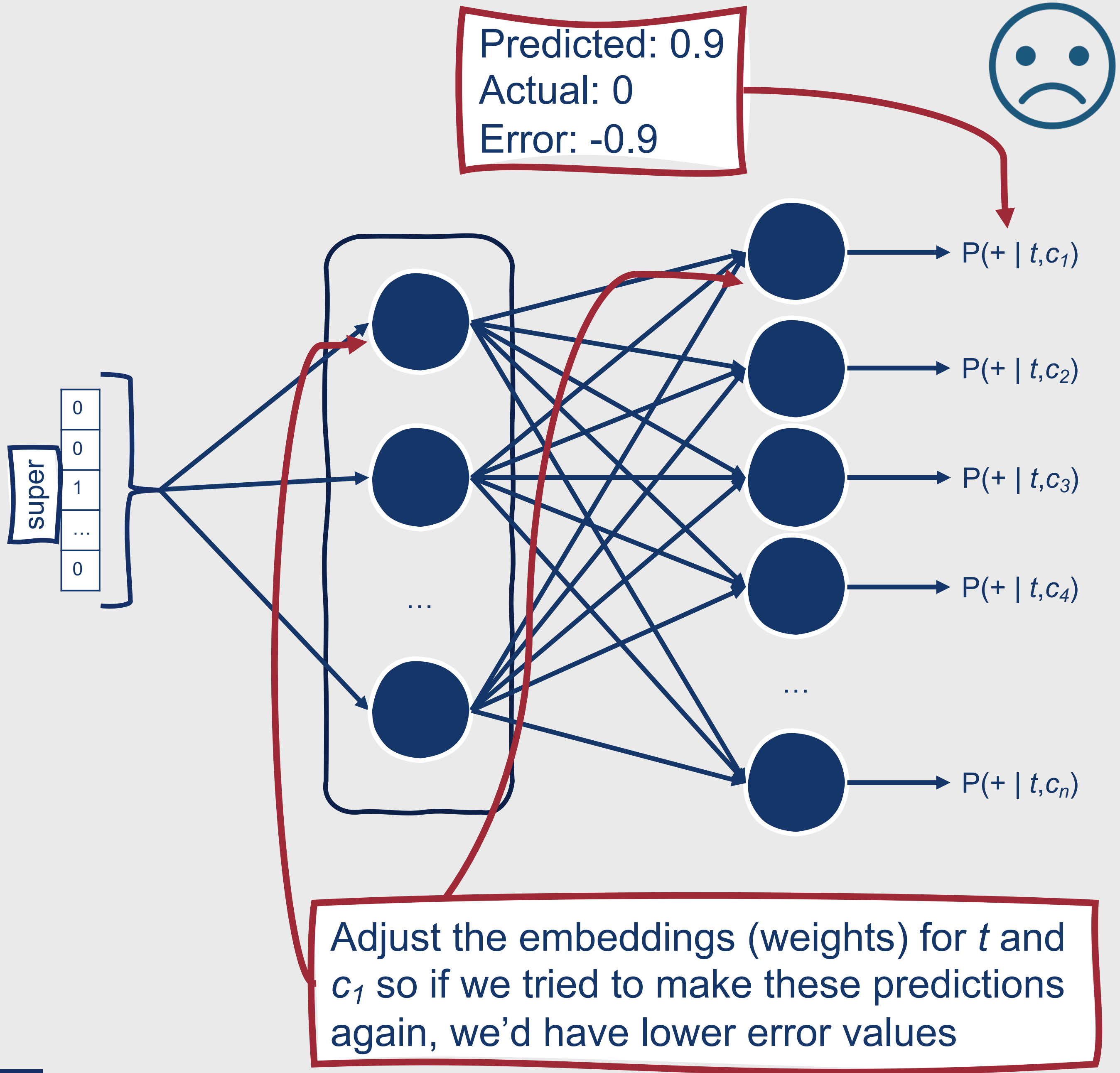
Since we initialize our weights randomly, the classifier's first prediction will almost certainly be wrong.



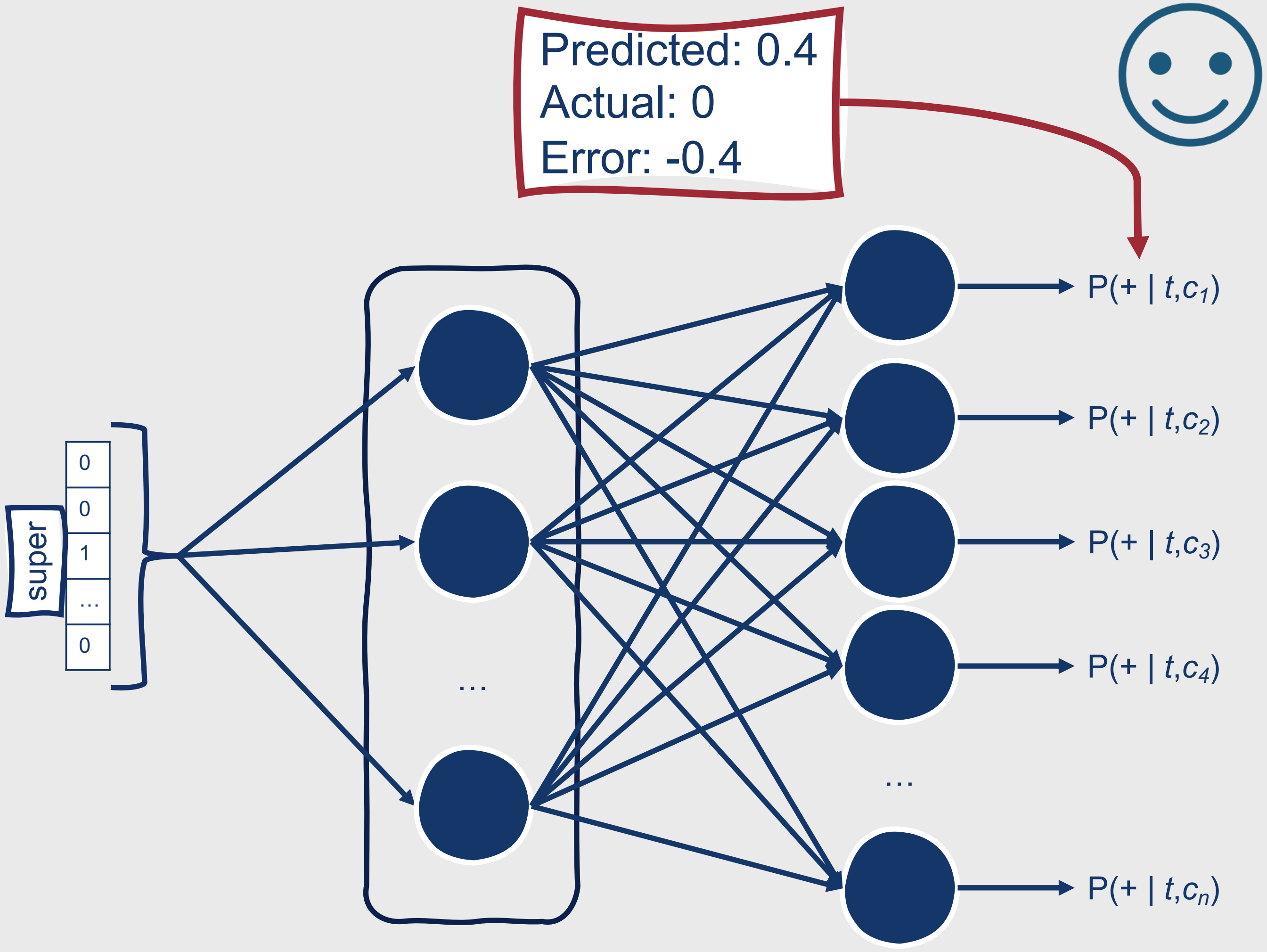
However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



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.



What is our 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**

What is our 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

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

What is our training data?

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:
    - $p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha}$

What is our 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