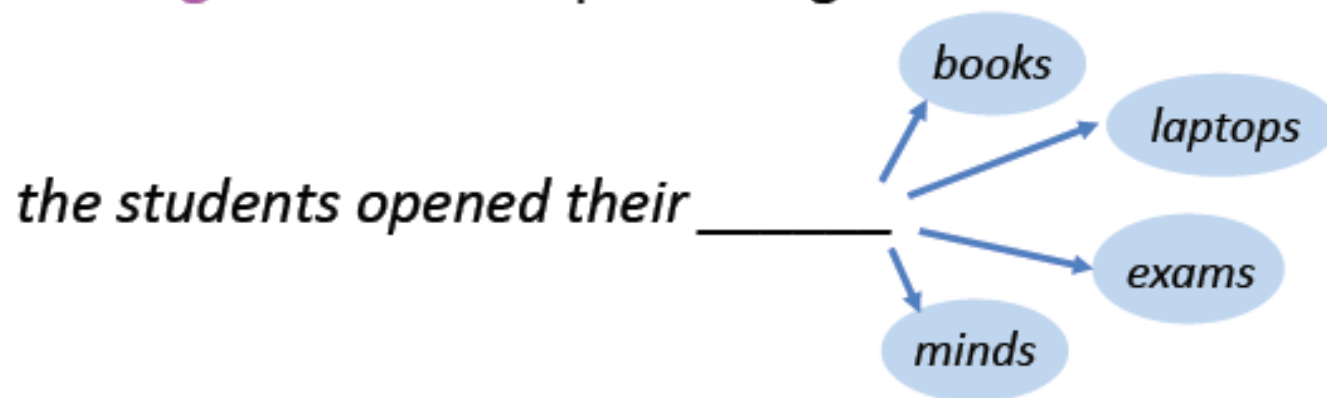


# Neural Language Models

# Language Modeling

- **Language Modeling** is the task of predicting what word comes next.



- More formally: given a sequence of words  $x^{(1)}, x^{(2)}, \dots, x^{(t)}$ , compute the probability distribution of the next word  $x^{(t+1)}$ :

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$$

where  $x^{(t+1)}$  can be any word in the vocabulary  $V = \{w_1, \dots, w_{|V|}\}$

- A system that does this is called a **Language Model**.

# Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.
- For example, if we have some text  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$ , then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) &= P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)}) \\ &= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) \end{aligned}$$

This is what our LM provides

# N-gram Language Models Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ students opened their \_\_\_\_\_  
discard condition on this

$$P(\mathbf{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
  - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
- “students opened their exams” occurred 100 times
  - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$

Should we have  
discarded the  
“proctor” context?

# Sparsity Problems with n-gram Language Models

## Sparsity Problem 1

**Problem:** What if “students opened their  $w$ ” never occurred in data? Then  $w$  has probability 0!

**(Partial) Solution:** Add small  $\delta$  to the count for every  $w \in V$ . This is called *smoothing*.

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

## Sparsity Problem 2


**Problem:** What if “students opened their” never occurred in data? Then we can’t calculate probability for any  $w$ !

**(Partial) Solution:** Just condition on “opened their” instead. This is called *backoff*.

**Note:** Increasing  $n$  makes sparsity problems worse. Typically we can’t have  $n$  bigger than 5.

# Storage Problems with n-gram Language Models

**Storage:** Need to store count for all  $n$ -grams you saw in the corpus.


$$P(\boldsymbol{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing  $n$  or increasing corpus  
increases model size!

# n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop\*

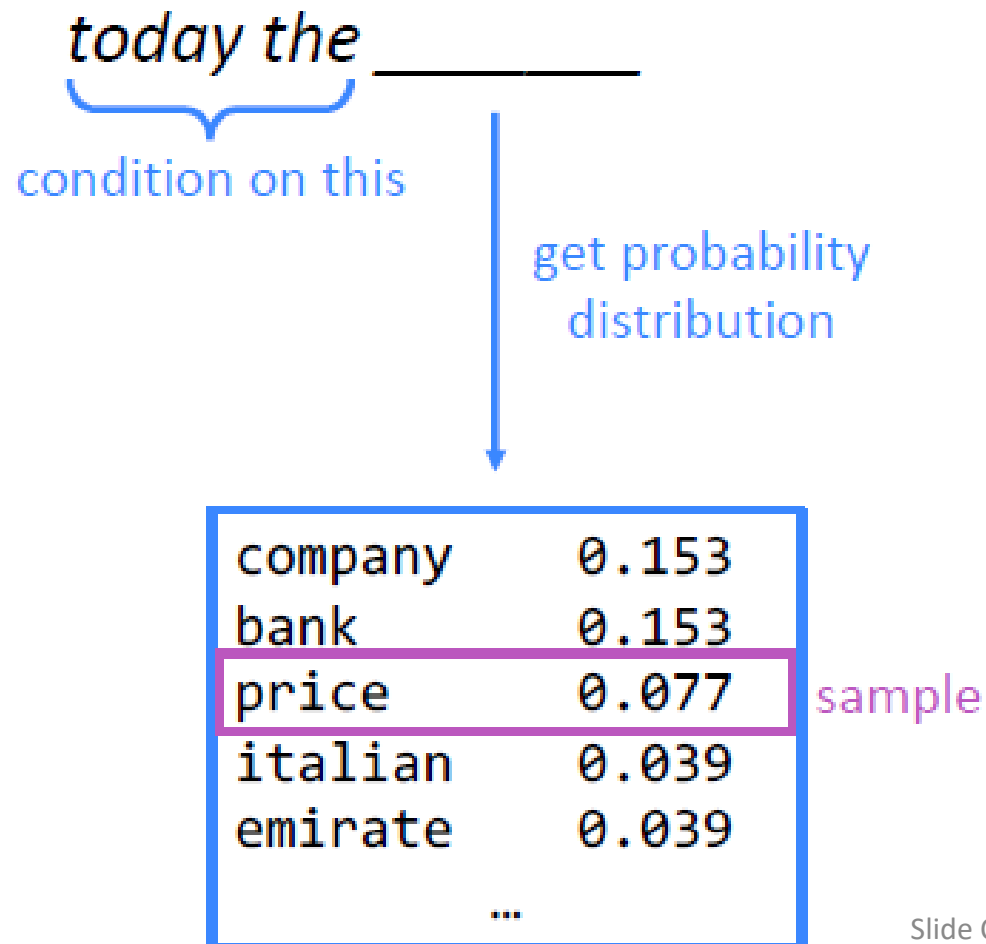
Business and financial news



Try for yourself: <https://nlpforhackers.io/language-models/>

# Generating text with a n-gram Language Model

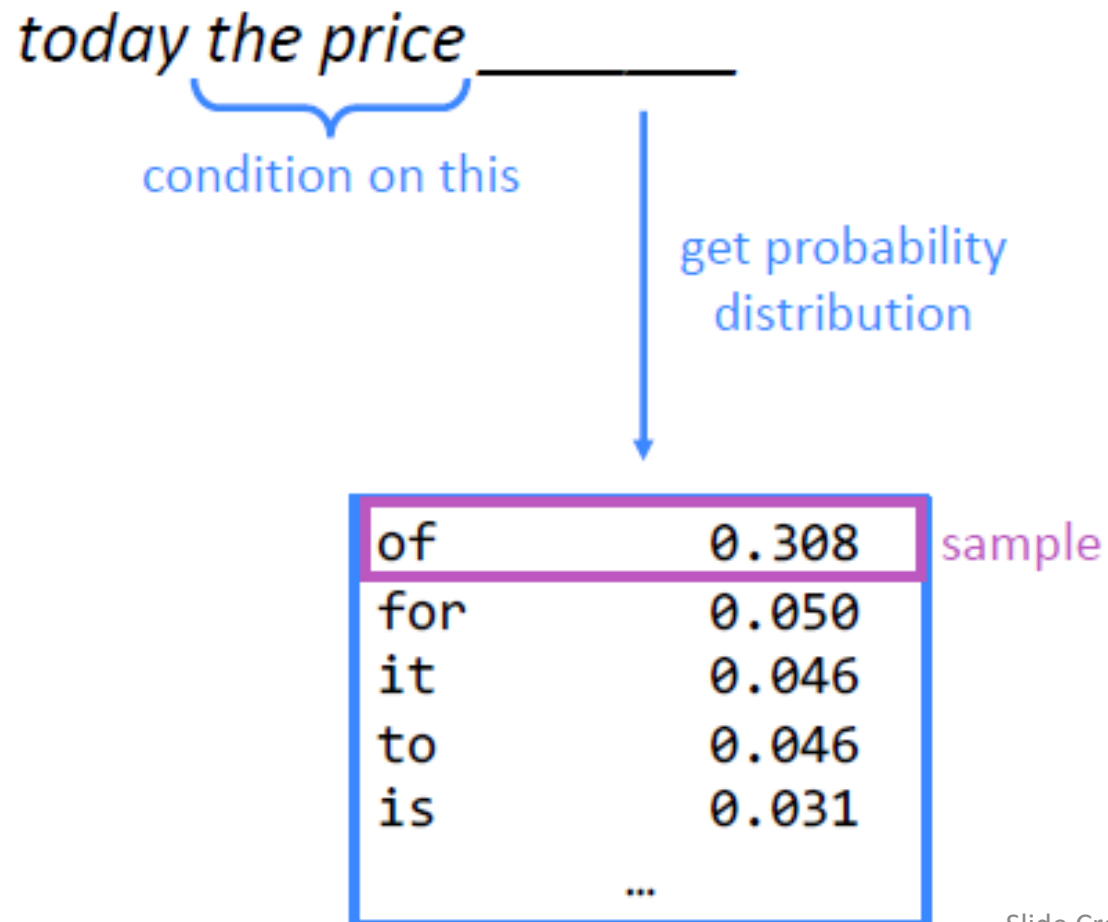
You can also use a Language Model to generate text.





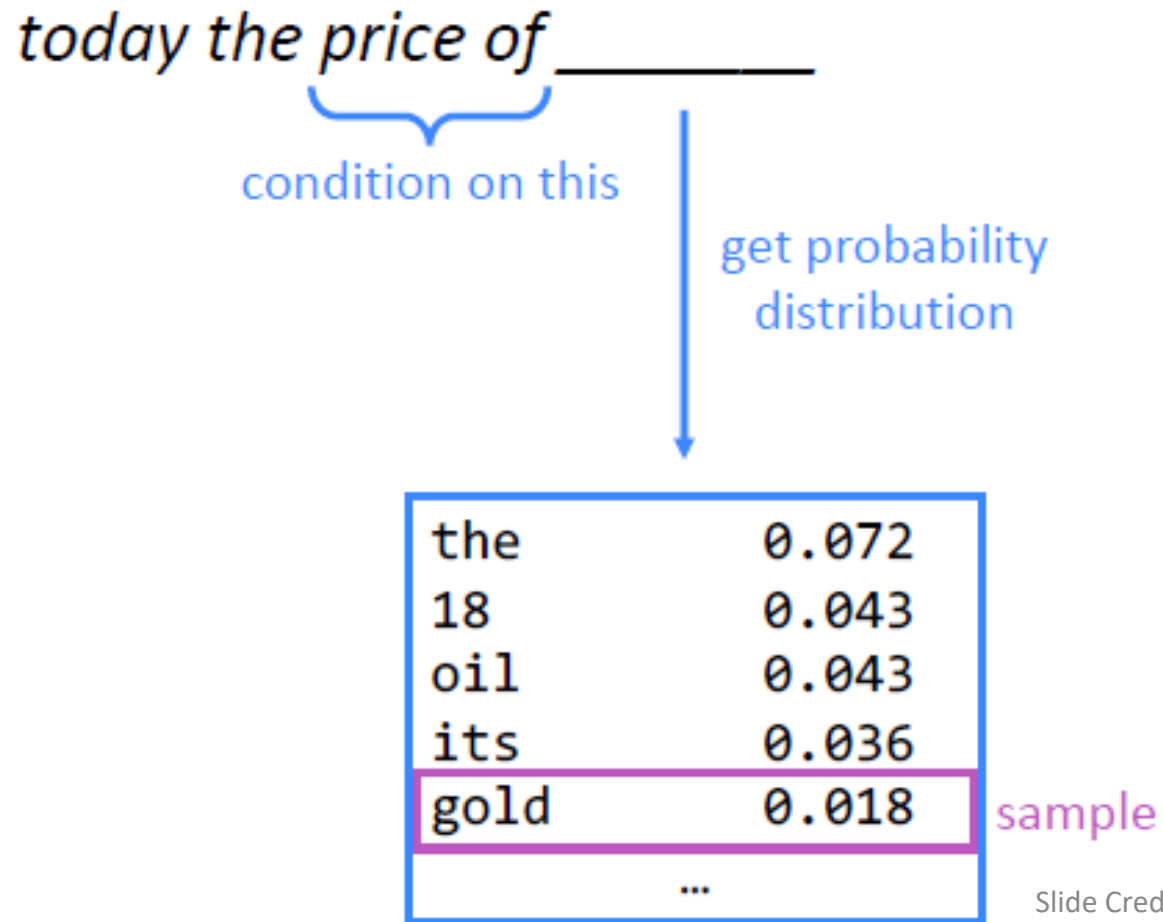
# Generating text with a n-gram Language Model

You can also use a Language Model to generate text.



# Generating text with a n-gram Language Model

You can also use a Language Model to generate text.



# Generating text with a n-gram Language Model

You can also use a Language Model to generate text.

*today the price of gold \_\_\_\_\_*

# Generating text with a n-gram Language Model

You can also use a Language Model to generate text.

*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

Surprisingly grammatical!

...but **incoherent**. We need to consider more than three words at a time if we want to model language well.

But increasing  $n$  worsens sparsity problem,  
and increases model size...

# Advantages of Neural Language Models:

- Don't need smoothing,
- Can handle much longer histories,
- Can generalize over contexts of similar words.

# How to build a *neural* Language Model?

Recall the Language Modeling task:

- Input: sequence of words  $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
- Output: prob dist of the next word  $P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$

How about a window-based neural model?

# A fixed-window neural Language Model

~~as the proctor started the clock~~

discard

the students opened their

fixed window

# A fixed-window neural Language Model

output distribution

$$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$$

hidden layer

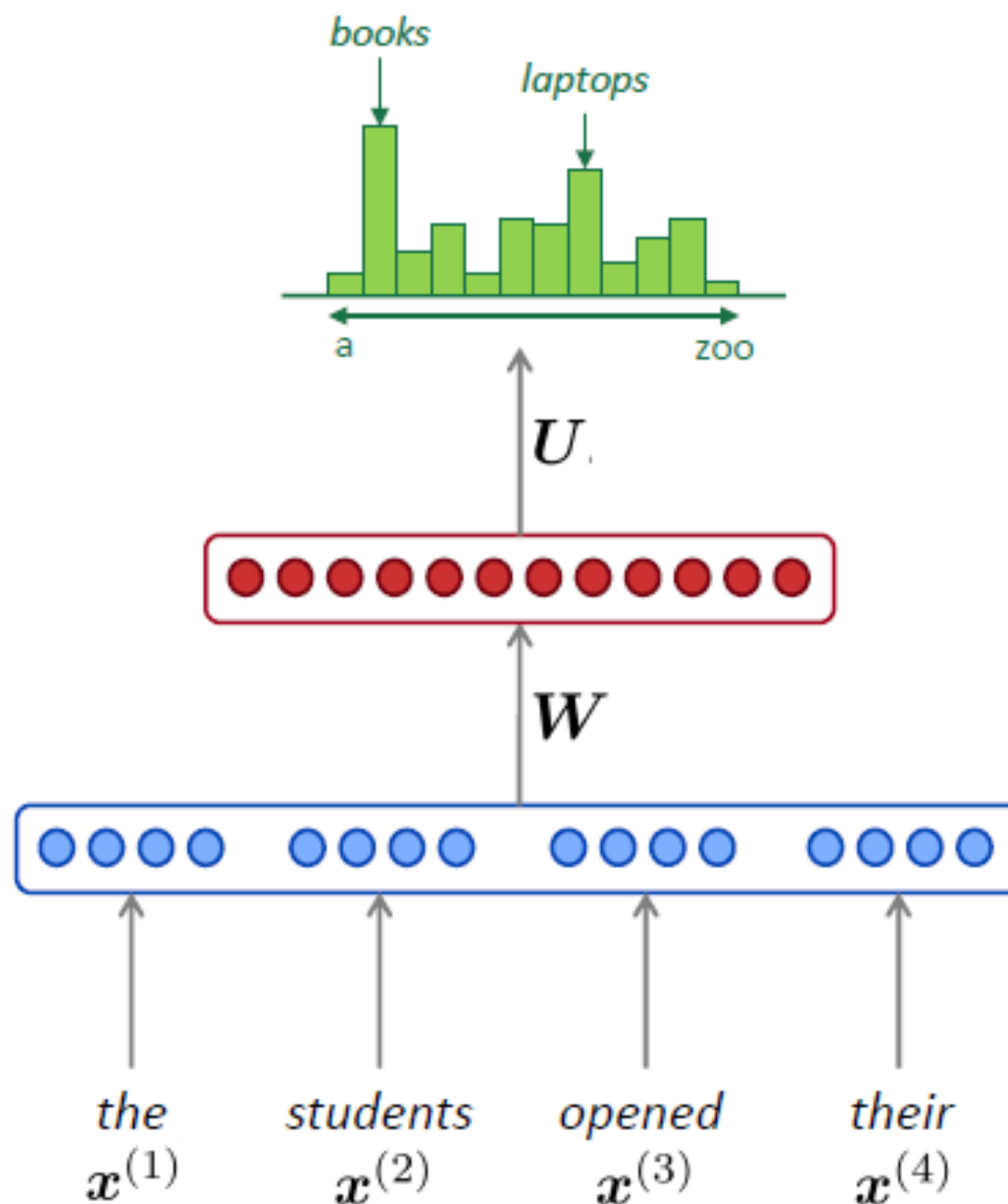
$$h = f(We + b_1)$$

concatenated word embeddings

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

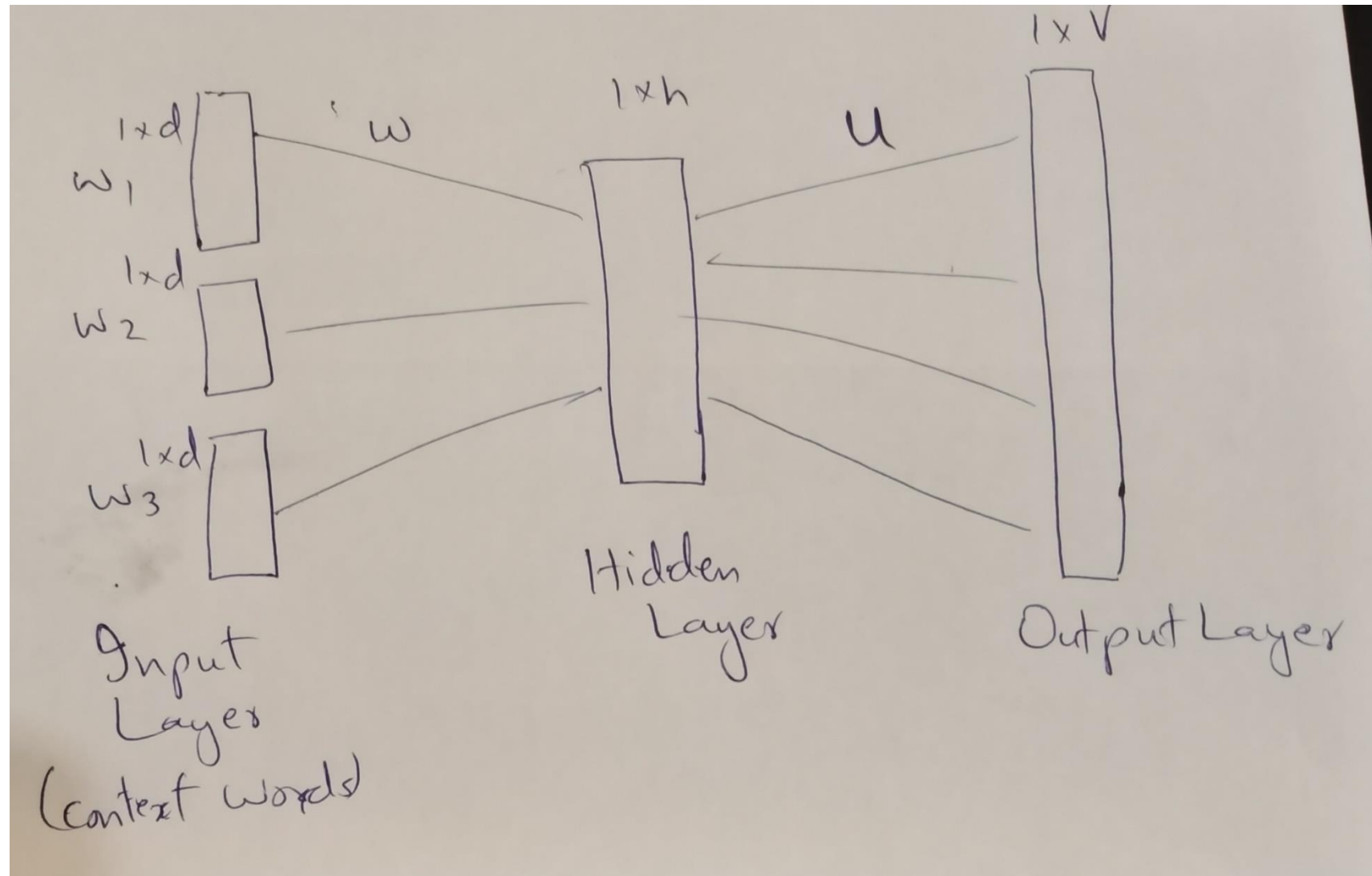
words / one-hot vectors

$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$

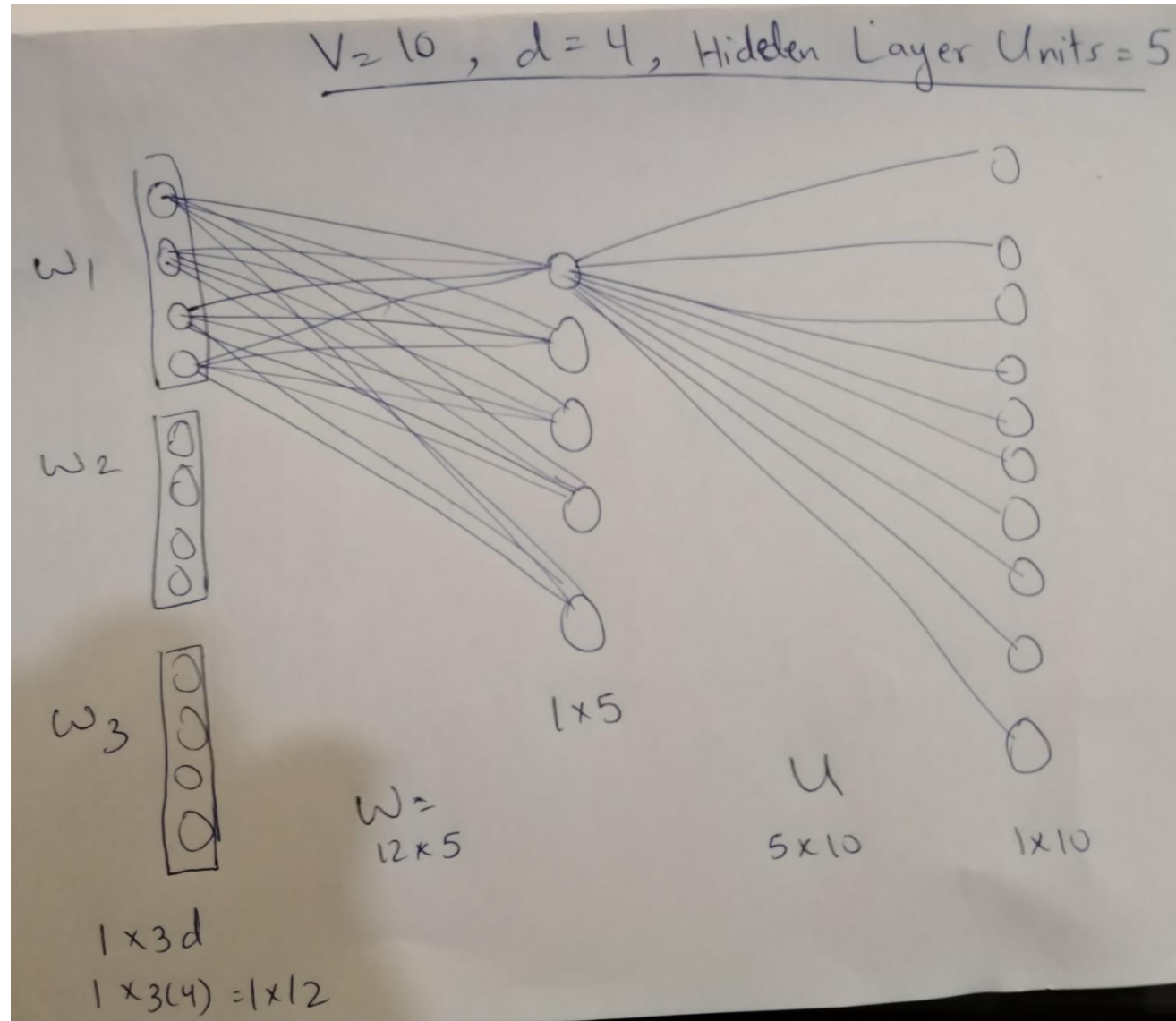




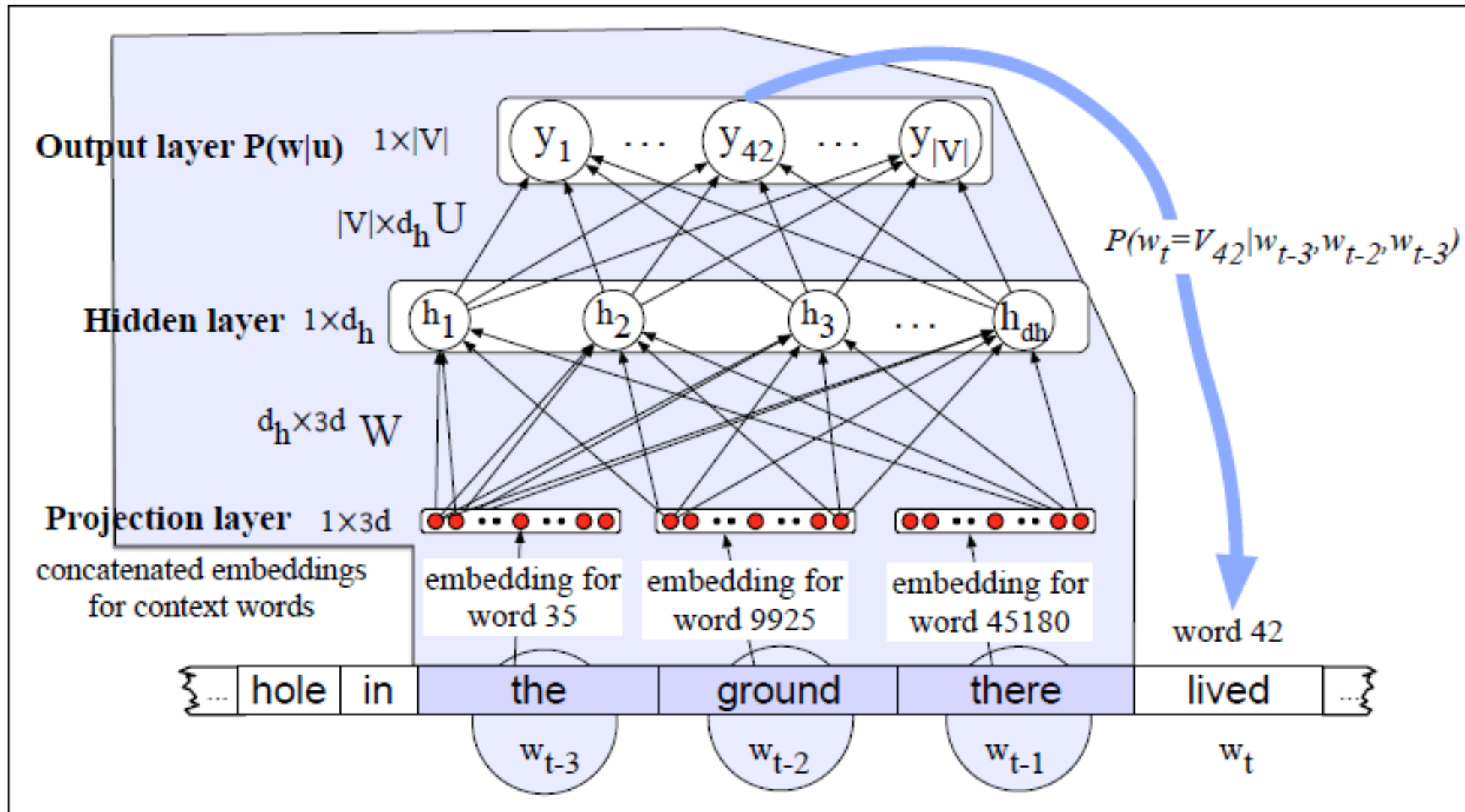
# High Level Architecture of Neural LM



# High Level Architecture of Neural LM

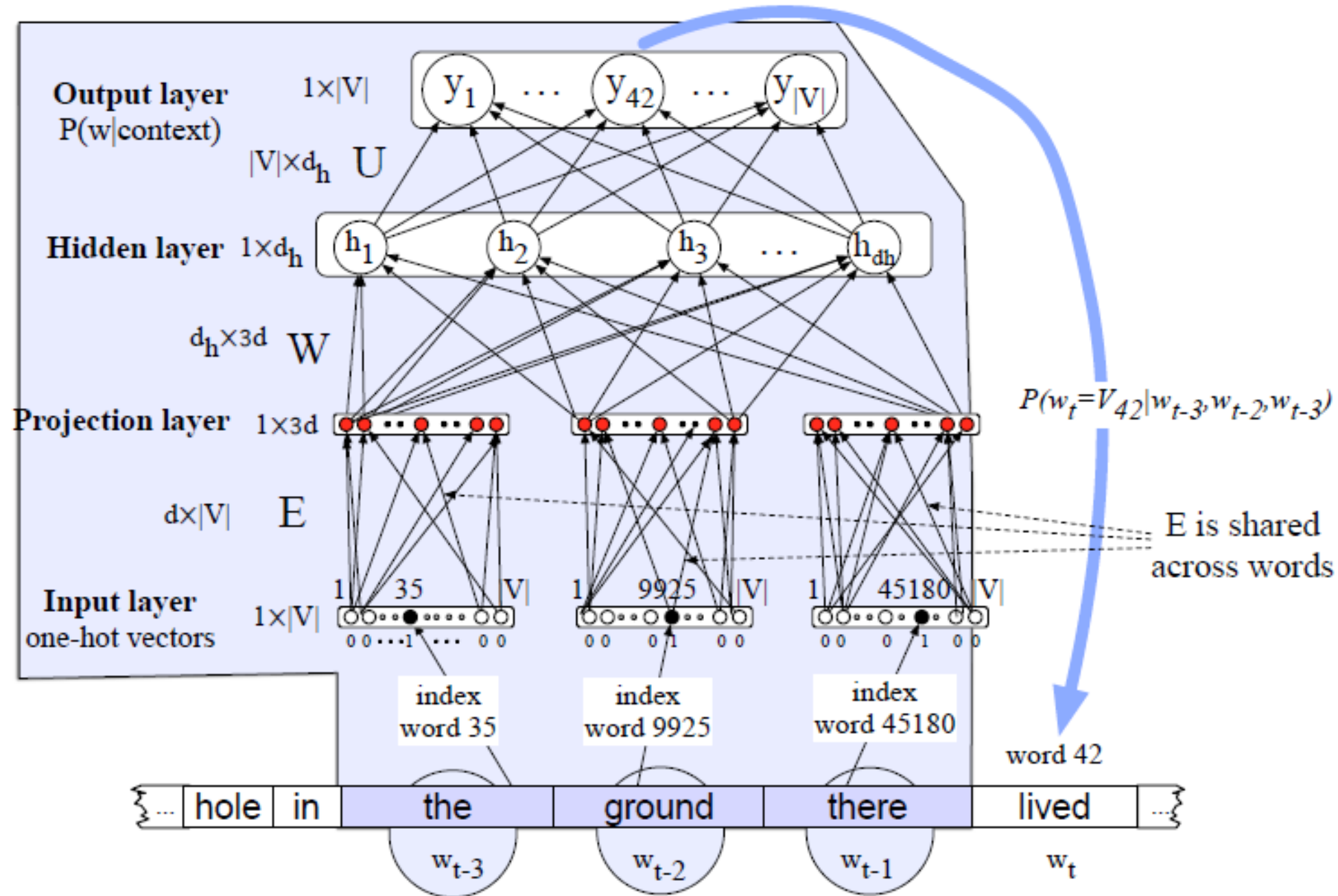


# Feedforward neural language model moving through a text



# Embedding as input for LM

- I have to make sure when I get home to feed the cat.
- I have to make sure when I get home to feed the \_\_\_\_\_
- Word “dog” will also have high probability to appear here even if it is not seen in text with same context.



# Forward Pass

$$e = (Ex_1, Ex_2, \dots, Ex)$$

$$h = \sigma(We + b)$$

$$z = Uh$$

$$y = \text{softmax}(z)$$

# Forward Pass

$$e = (Ex_1, Ex_2, \dots, Ex)$$

$$h = \sigma(We + b)$$

$$z = Uh$$

$$y = \text{softmax}(z)$$

1. **Select three embeddings from E:** the projection layer for input  $w$  will be  $Ex_i = e_i$ , the embedding for word  $i$ .
2. **Multiply by W:** We now multiply by weight matrix  $W$  (and add  $b$ ) and pass through activation function to get the hidden layer  $h$ .
3. **Multiply by U:**  $h$  is now multiplied by  $U$ ,  $U$  is also weight matrix
4. **Apply softmax:** After the softmax, each node  $i$  in the output layer estimates the probability  $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$

# Training of Model

**Loss Function**  $L = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})$

The gradient for this loss is

$$\theta_{t+1} = \theta_t - \eta \frac{\partial -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta}$$



# Training the neural language model

This gradient can be computed in any standard neural network framework which will then backpropagate through  $U, W, b, E$ .

Training the parameters to minimize loss will result both in an algorithm for language modeling (a word predictor) but also a new set of embeddings  $E$  that can be used as word representations for other tasks.

# A fixed-window neural Language Model

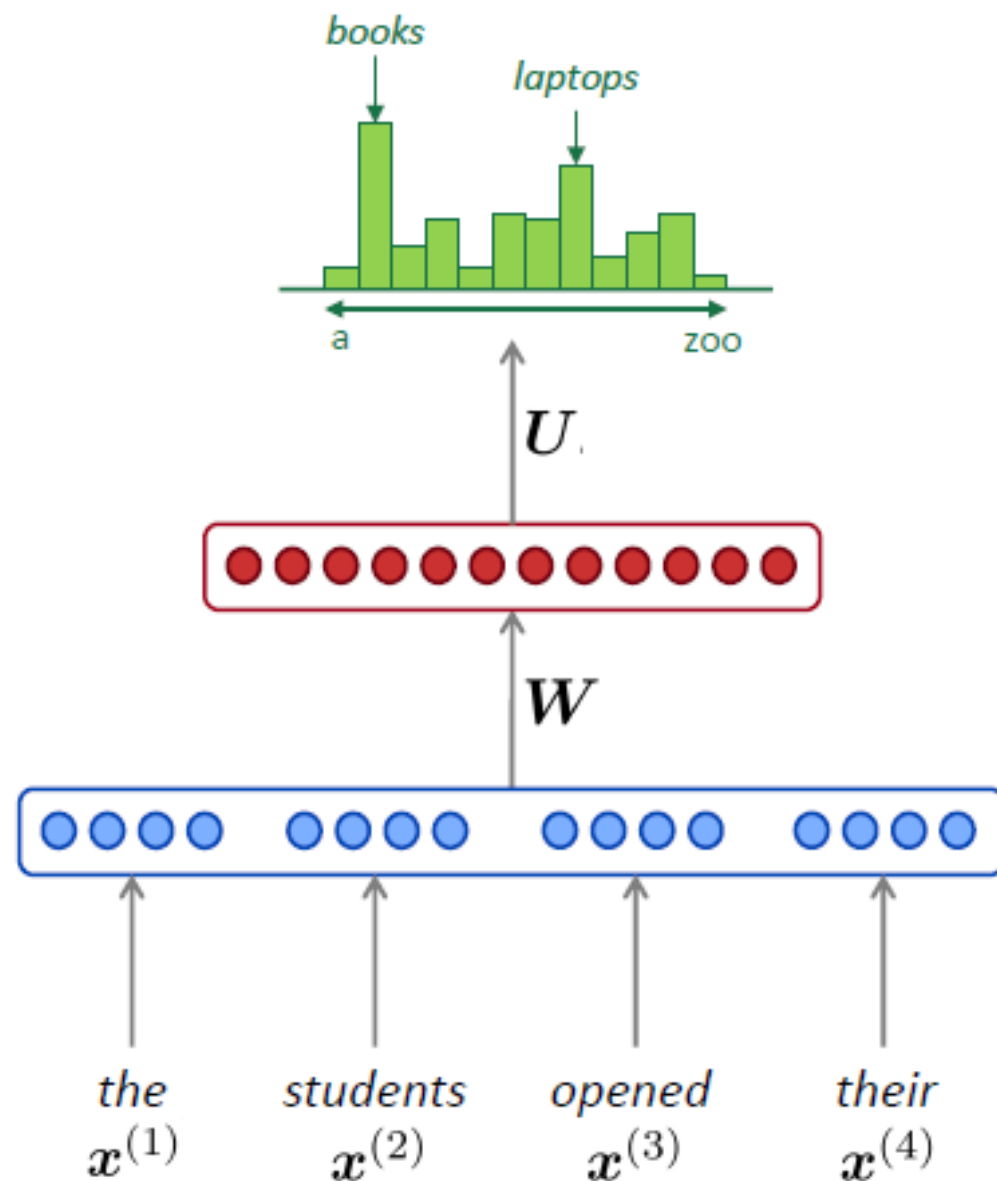
**Improvements** over  $n$ -gram LM:

- No sparsity problem
- Don't need to store all observed  $n$ -grams

Remaining **problems**:

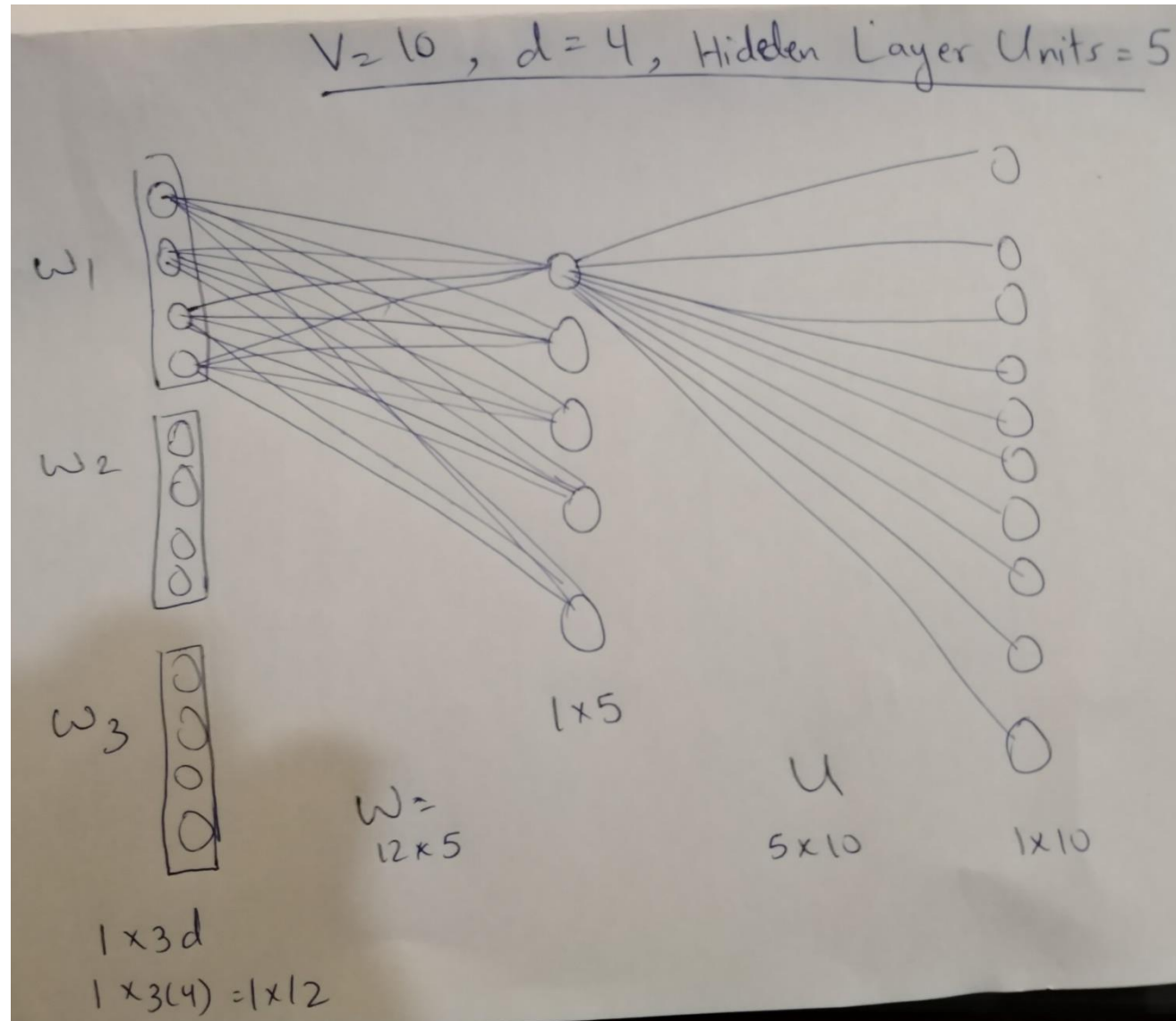
- Fixed window is **too small**
- Enlarging window enlarges  $W$ .
- Window can never be large enough!
- $x^{(1)}$  and  $x^{(2)}$  are multiplied by completely different weights in  $W$ .  
**No symmetry** in how the inputs are processed.

We need a neural architecture that can process *any length input*



# High Level Architecture of Neural LM

(Different weights for each word are used in Matrix  $W$  in first layer)



# Reading

- Chapter 7, Speech and Language Processing. Daniel Jurafsky & James H. Martin. Third edition  
<https://web.stanford.edu/~jurafsky/slp3/7.pdf>