Neural Language Models

Language Modeling

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

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

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

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

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

Slide Credit: Abigail See

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 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Slide Credit: Abigail See

N-gram Language Models Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* _____

~~as the proctor started the clock, the~~ — discard

students opened their — condition on this

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Slide Credit: Abigail See

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

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

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

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

(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.

Slide Credit: Abigail See

Storage Problems with n-gram Language Models

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

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

Increasing *n* or increasing corpus increases model size!

Slide Credit: Abigail See

n-gram Language Models in practice

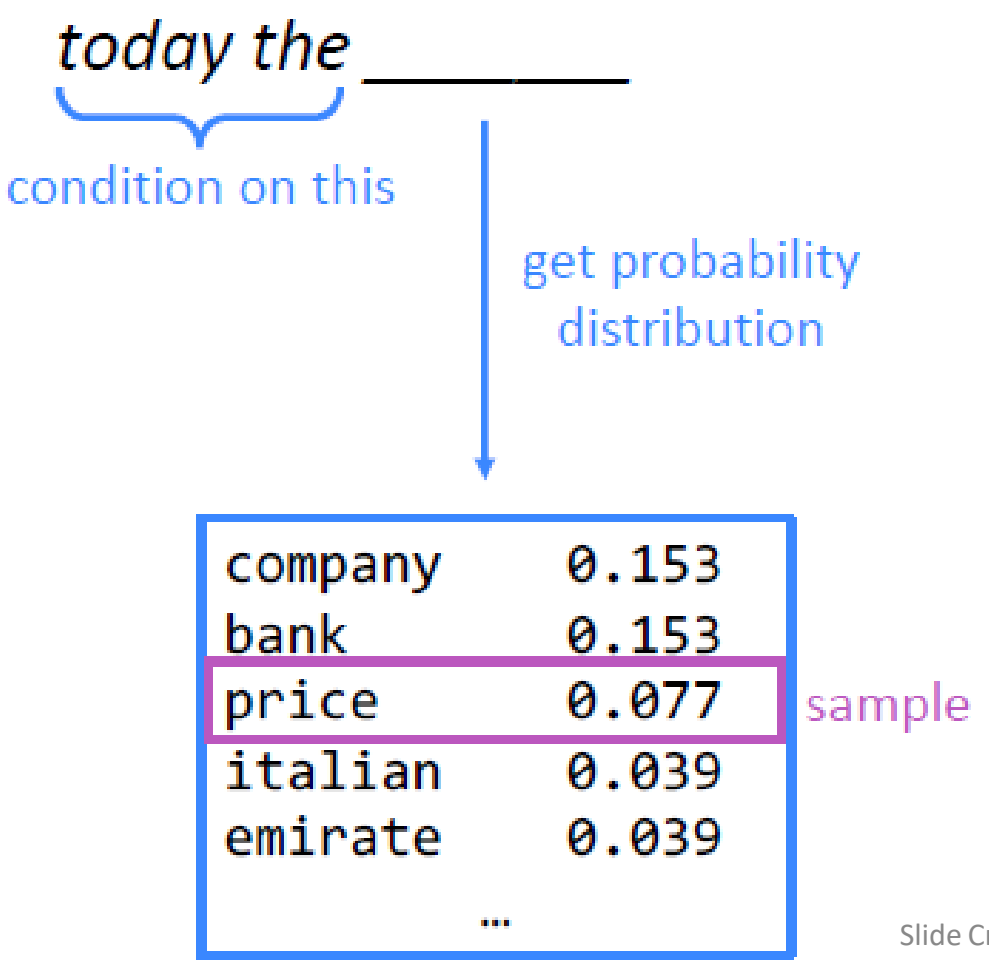
- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

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

Slide Credit: Abigail See

Generating text with a n-gram Language Model

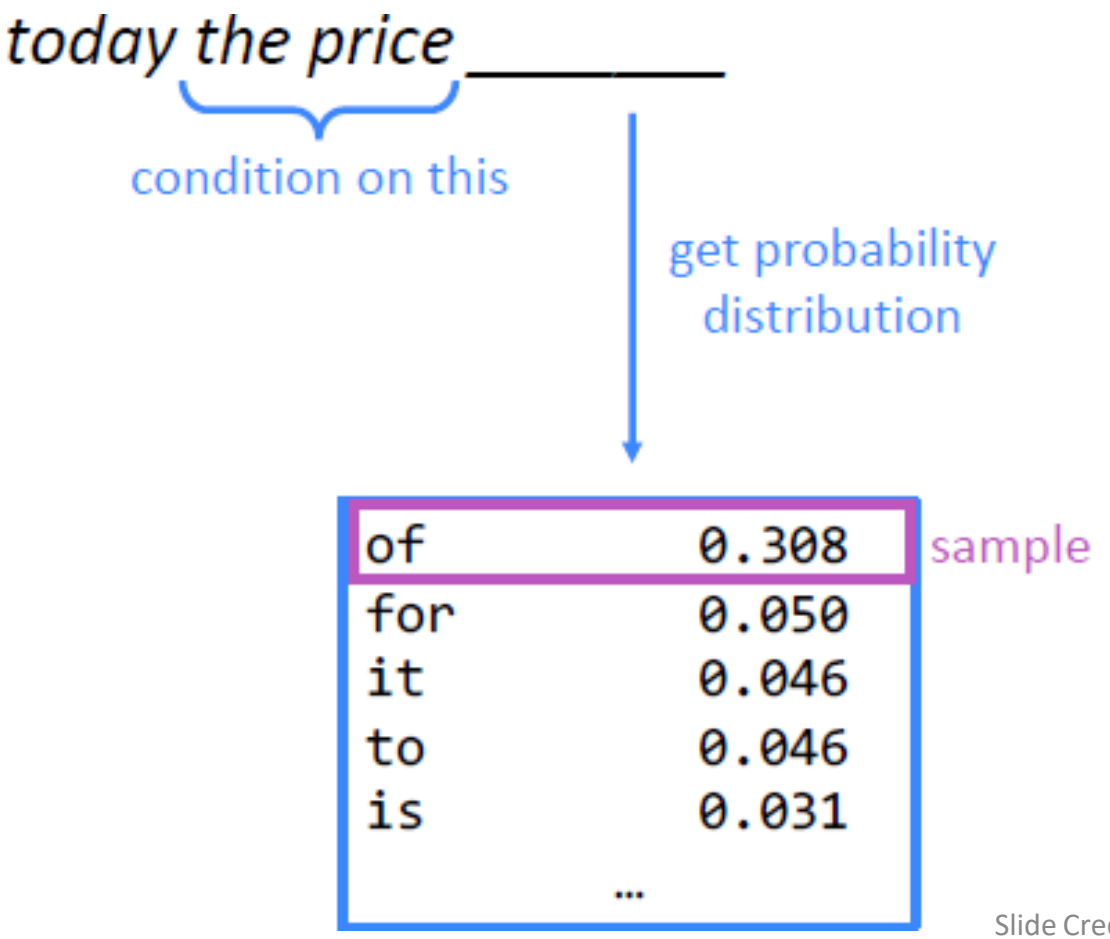
You can also use a Language Model to generate text.



Slide Credit: Abigail See

Generating text with a n-gram Language Model

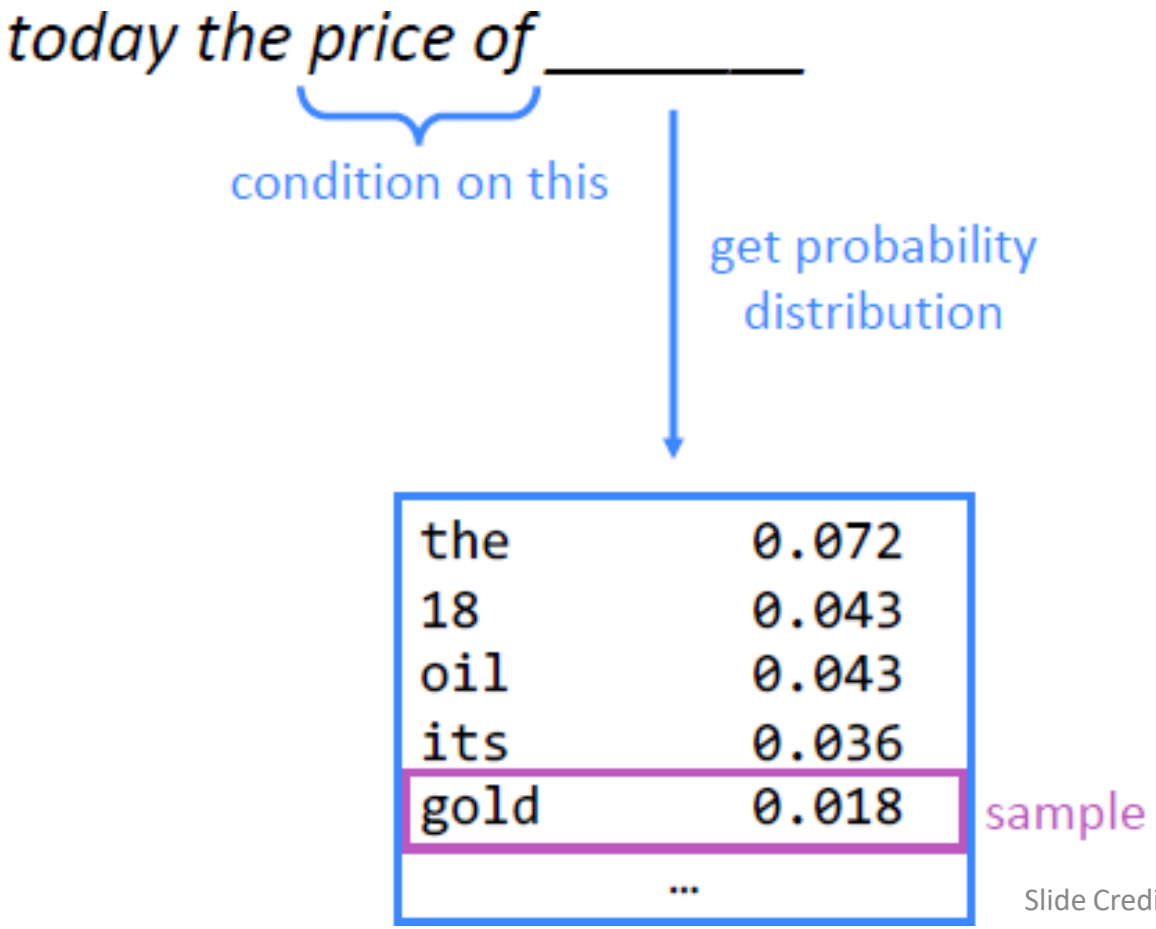
You can also use a Language Model to generate text.



Slide Credit: Abigail See

Generating text with a n-gram Language Model

You can also use a Language Model to generate text.



Slide Credit: Abigail See

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...

Slide Credit: Abigail See

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?

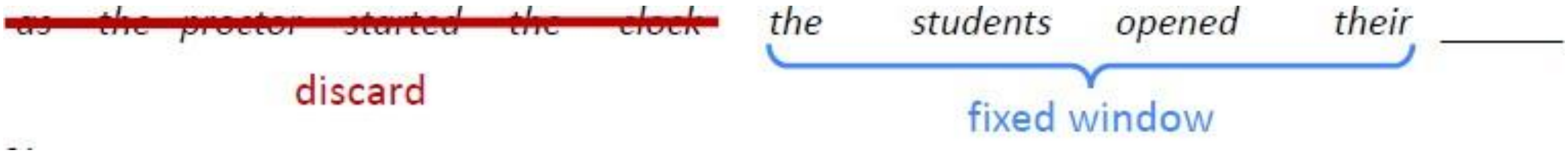
Recall the Language Modeling task:

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

How about a window-based neural model?

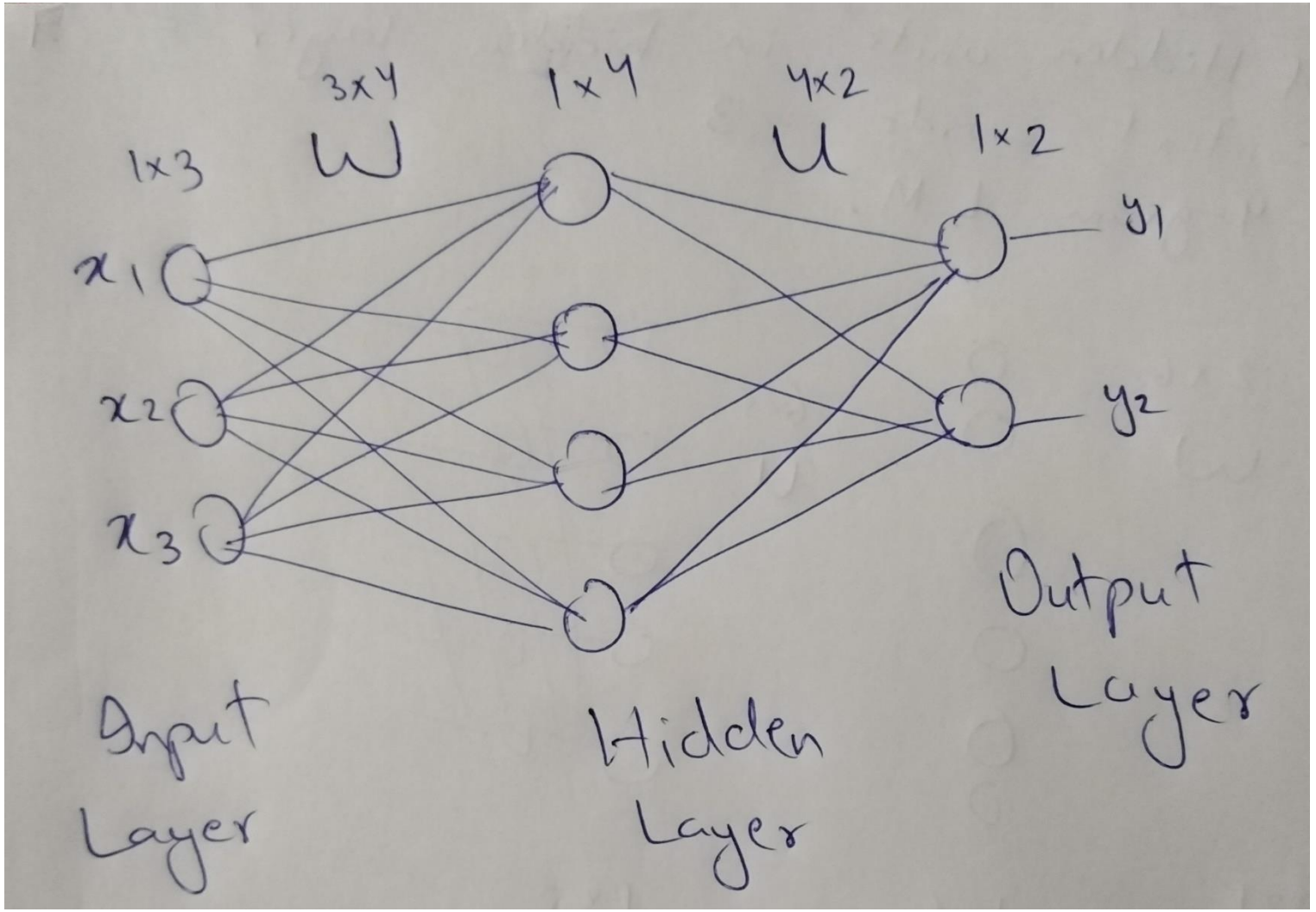
Slide Credit: Abigail See

A fixed-window neural Language Model



Slide Credit: Abigail See

2 Layer Neural Network (How to use it as a classifier for language modeling or how to treat language modeling as a classification problem)



A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

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

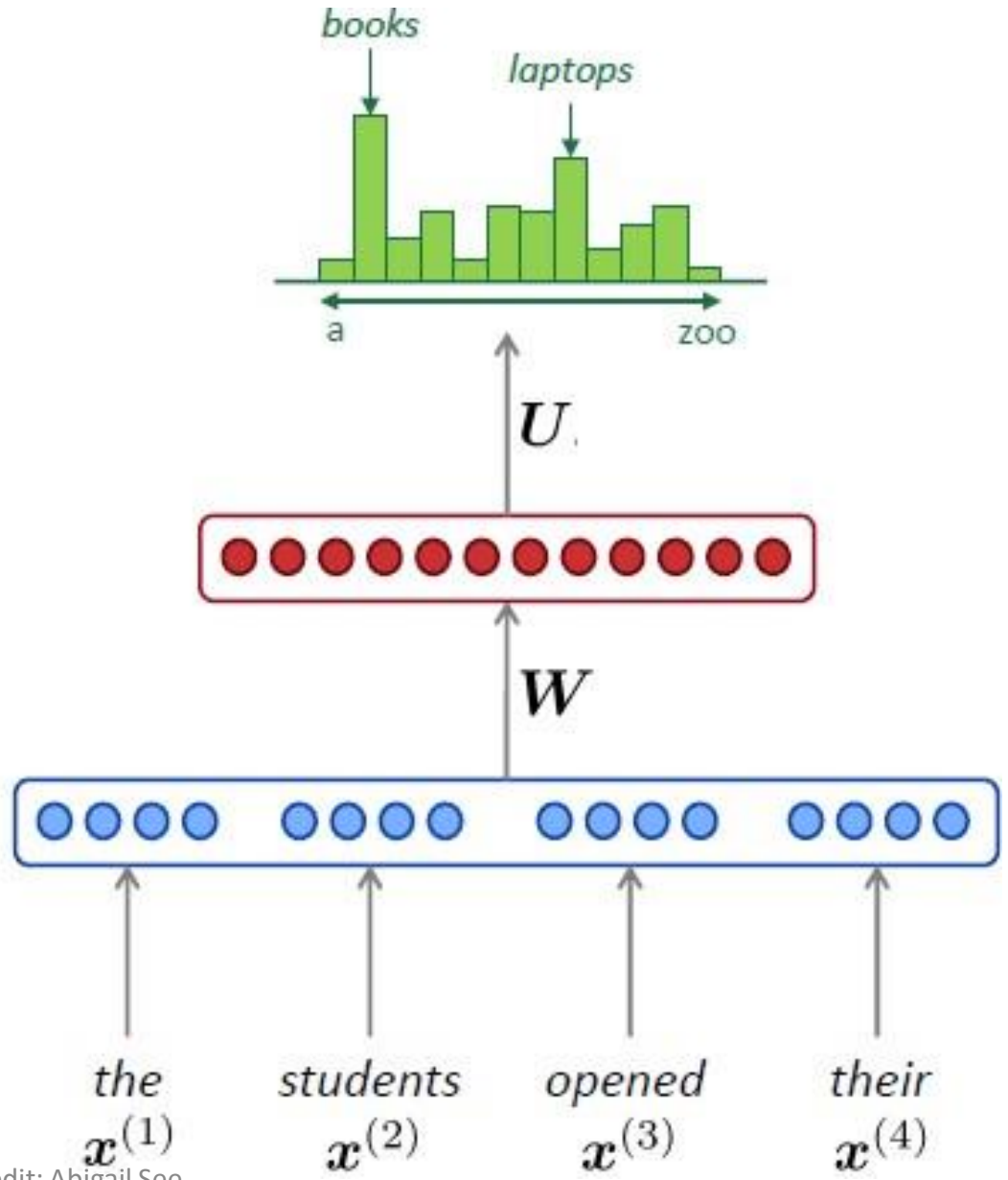
1 x d
e =E.x

E is the embedding matrix: *V x d*

words / one-hot vectors

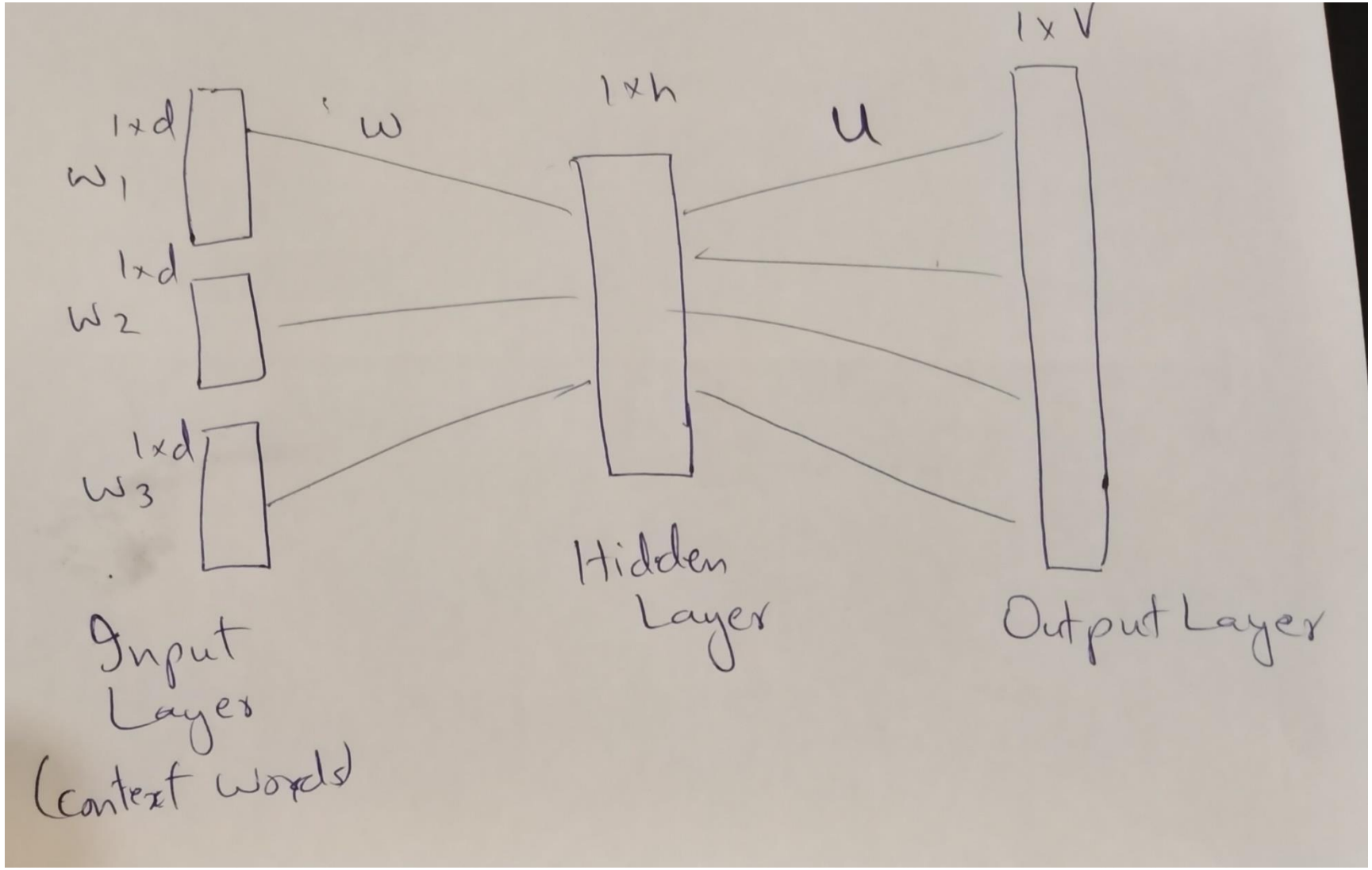
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

1x V for each word

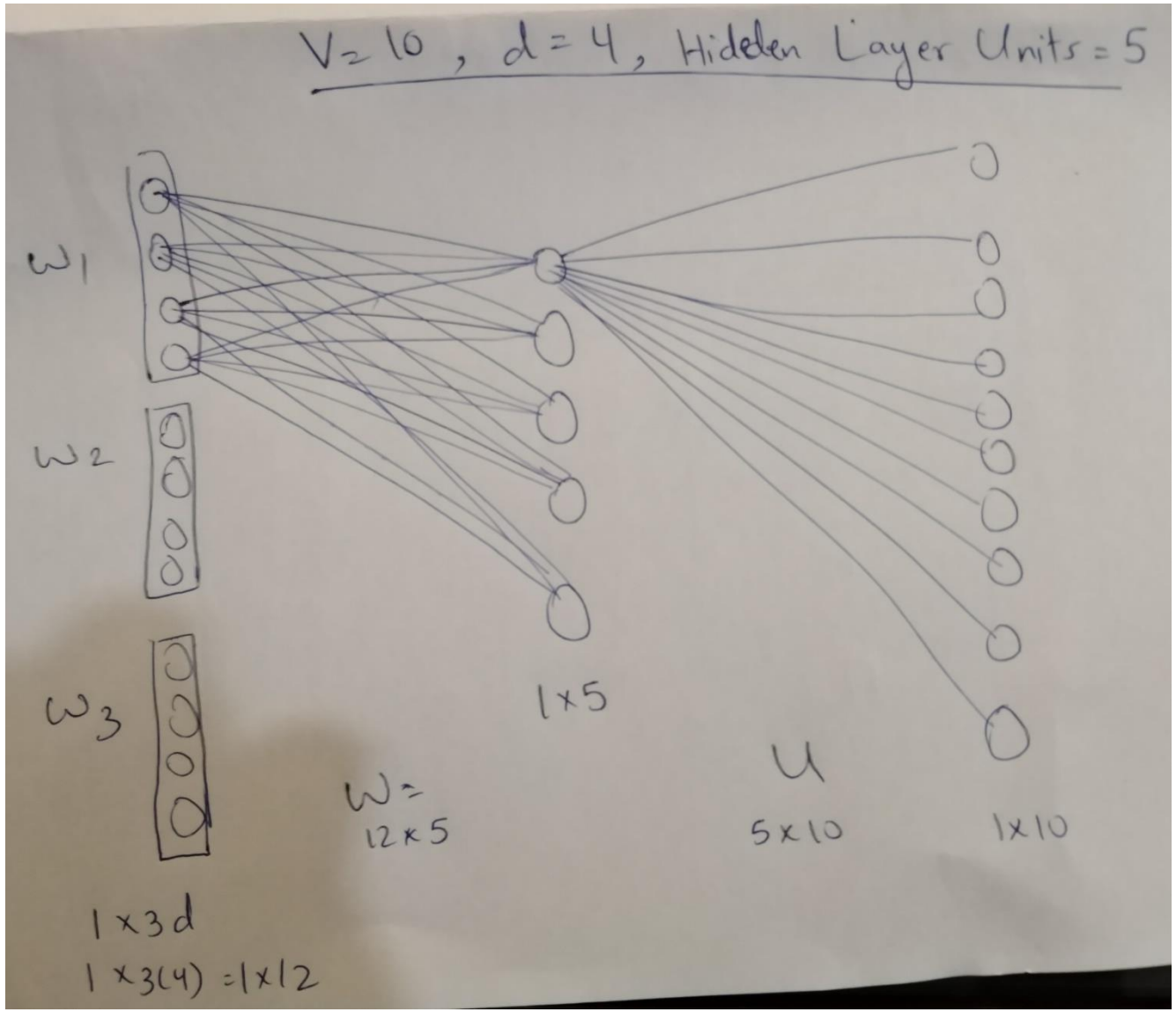


Slide Credit: Abigail See

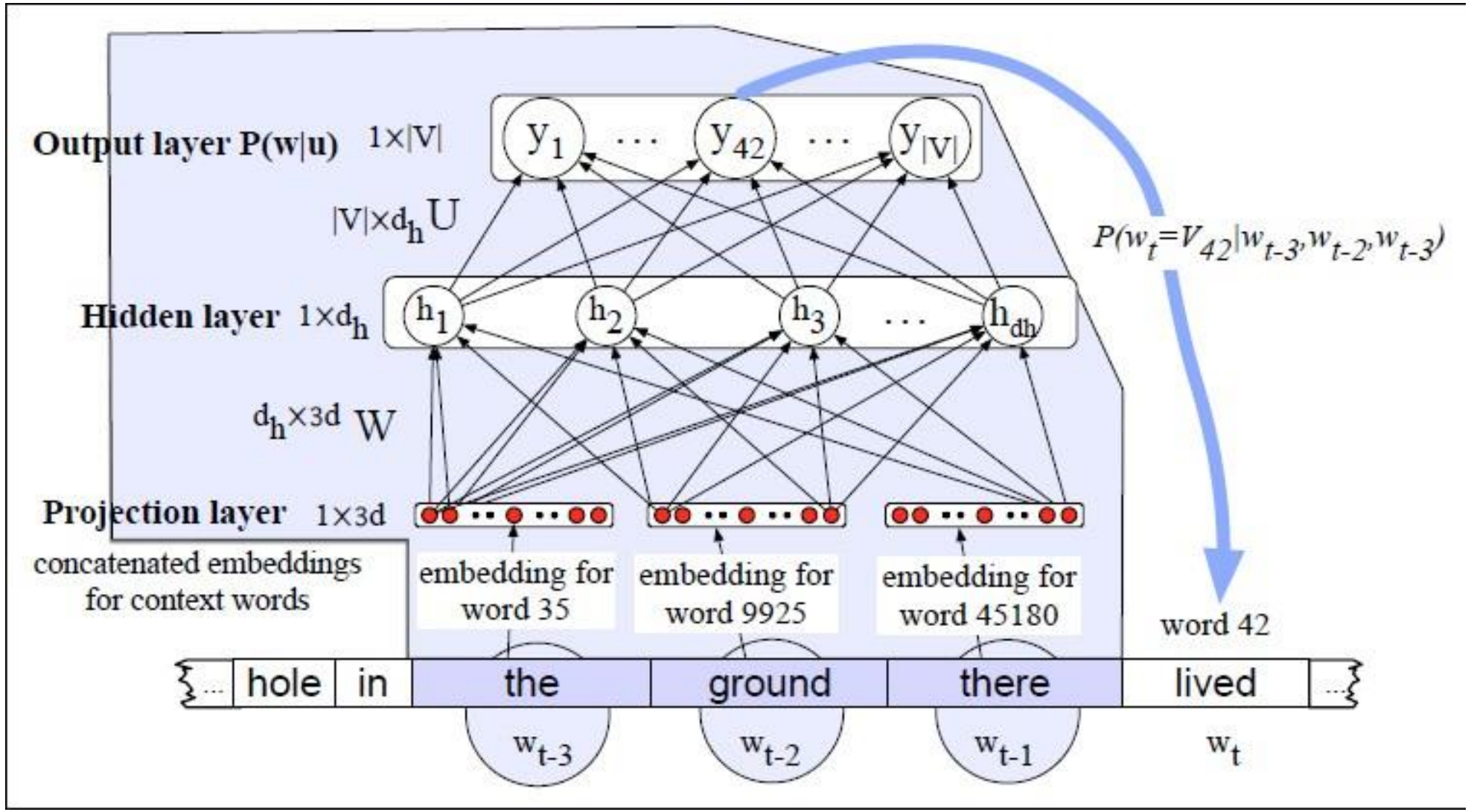
High Level Architecture of Neural LM



High Level Architecture of Neural LM

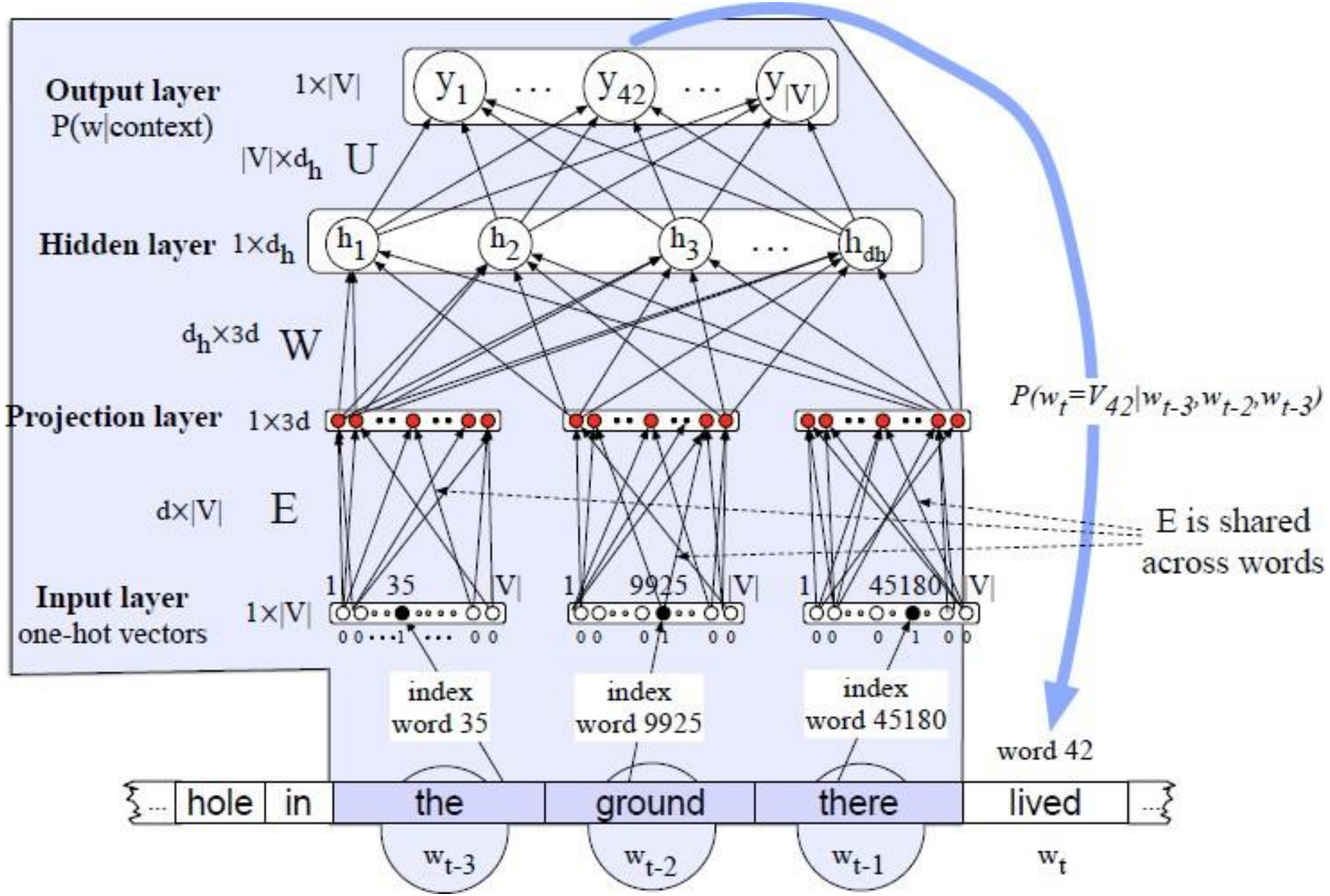


Feedforward neural language model moving through a text



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.



Fixed window of size 3

Forward Pass

$$
\begin{aligned}
e &= (Ex_1, Ex_2, \ldots, Ex) \\
h &= \sigma(We + b) \\
z &= Uh \\
y &= \text{softmax}(z)
\end{aligned}
$$

Forward Pass

$$
\begin{aligned}
e &= (Ex_1, Ex_2, \ldots, Ex) \\
h &= \sigma(We + b) \\
z &= Uh \\
y &= \text{softmax}(z)
\end{aligned}
$$

1. **Select three embeddings from E:** the projection layer for input w will be $Ex_i = e_i$, the embedding for word i.

   The embedding matrix (E) is a lookup table that maps words to their corresponding numerical representations (embeddings).

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 $$L = -\log p(w_t | w_{t-1}, \ldots, w_{t-n+1})$$

The gradient for this loss is

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

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

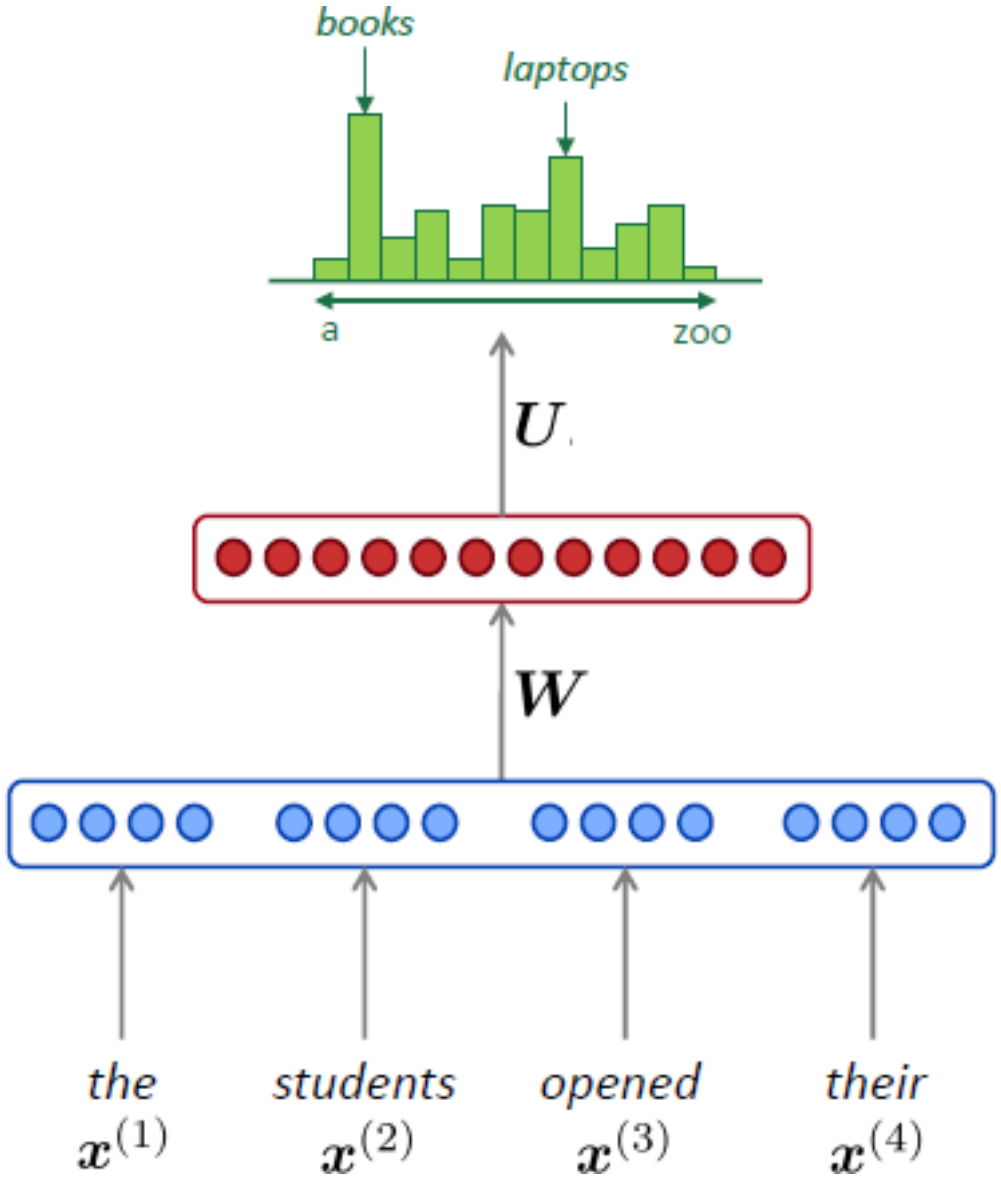
Improvements over *n*-gram LM:

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

Remaining **problems**:

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

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



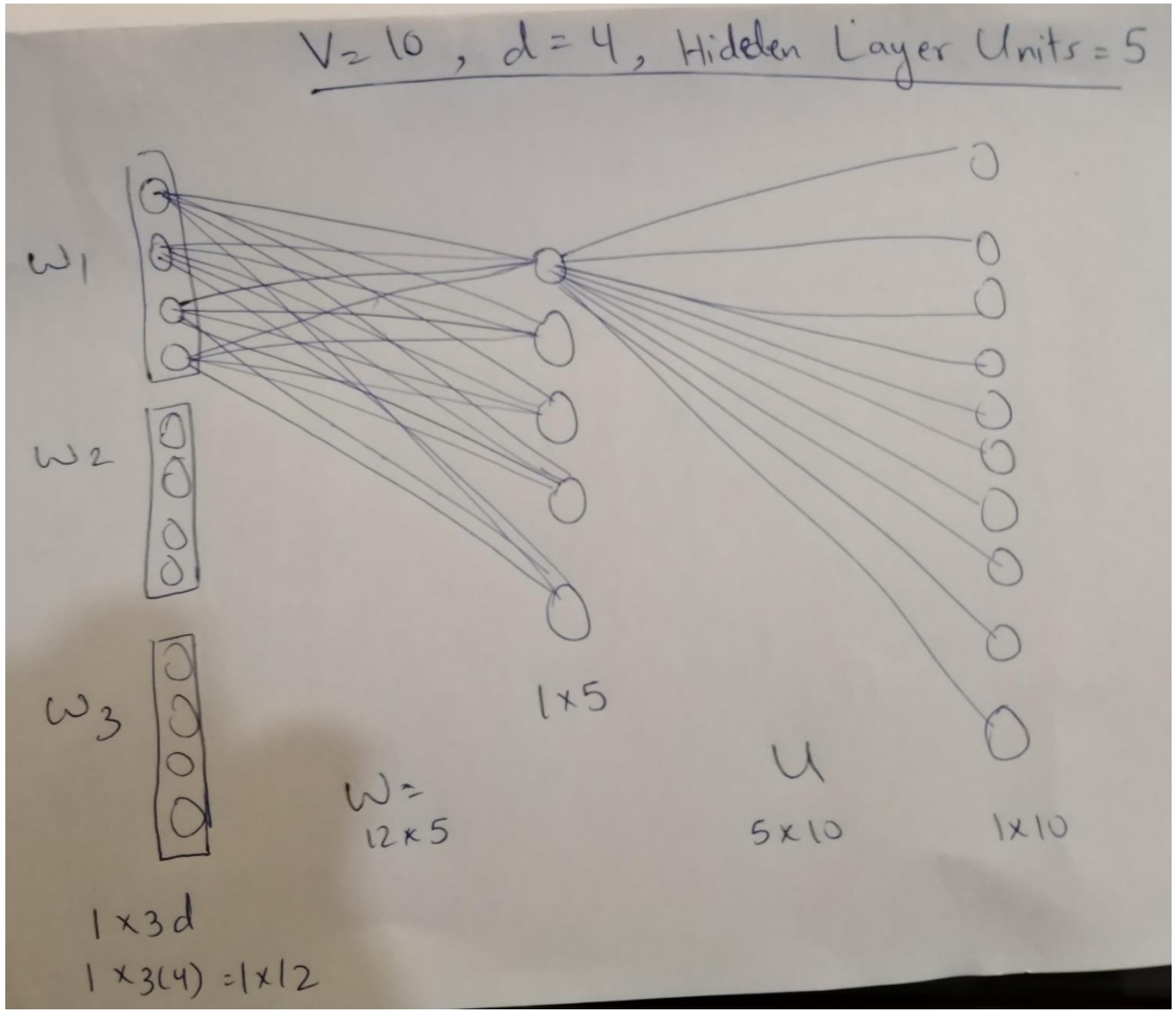
Using pre trained or custom embeddings

1. Using Pre-trained Embeddings:
Benefits: Pre-trained embeddings, such as Word2Vec, GloVe, or FastText, are trained on large datasets and **capture semantic relationships between words**. This can save training time and improve model performance, especially on smaller datasets.
Considerations: Pre-trained embeddings might not be perfectly tailored to your specific task or domain. You might need to fine-tune them on your own data.

2. Training Custom Embeddings:
Benefits: Custom embeddings can be tailored to your specific task or domain, potentially improving performance.
Challenges: Training custom embeddings requires a large amount of data and can be computationally expensive.

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

Q 1) (b) Suppose we have a collection with the following statistics: [3 Marks]

Total words: 300,000

Vocabulary: 2000

Hidden layers unit: 10

Context size = 3 words

We want to train a NLM (Neural Language Model) on this collection. The dimension of each word vector should be 150. Draw the NLM architecture diagram with dimensions of input, hidden, and output layers, and dimensions of weight matrices. Write equations for each layer. Draw the embedding layer.