ITCS 6101/8101: Natural Language Processing

Text Classification and Language Modeling with Neural Networks

Razvan C. Bunescu

Department of Computer Science @ CCI

razvan.bunescu@charlotte.edu

Two Use Cases for Feedforward FCNs

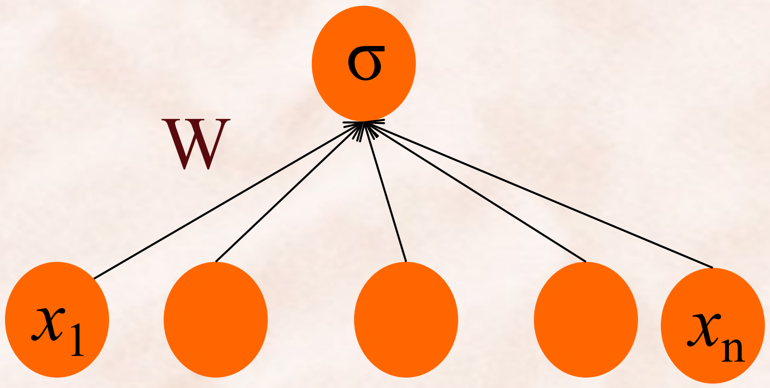
1. **Text Classification**:
   - With manually engineered features.
   - With word embeddings.

2. **Language Modeling**:
   - With word embeddings.

- State of the art systems use neural architectures more powerful than vanilla FCNs:
  - **RNNs**: LSTMs or GRUs.
  - **Transformer**: Masked LMs (BERT) or Causal LMs (GPT).

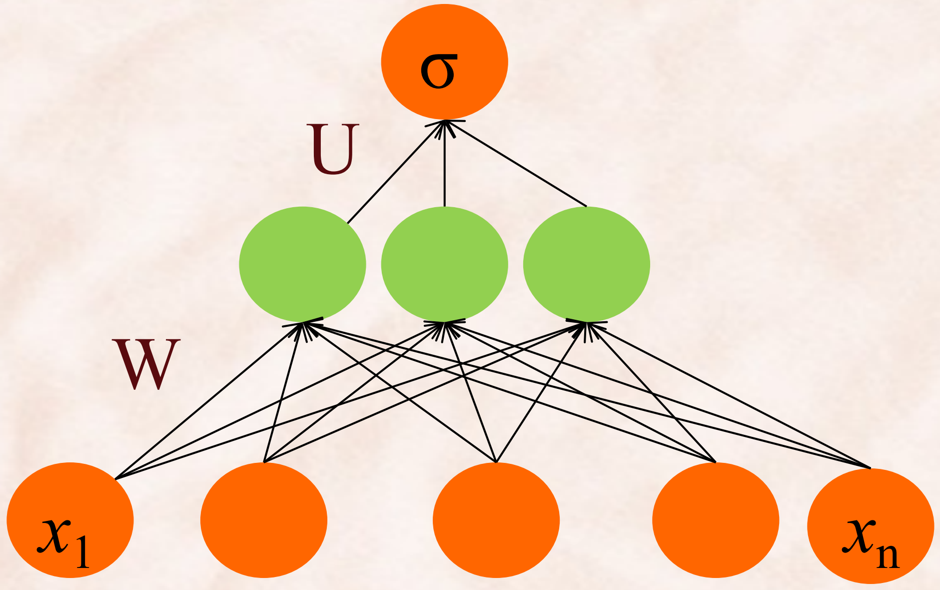
Sentiment Analysis

Logistic Regression



Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Neural Network



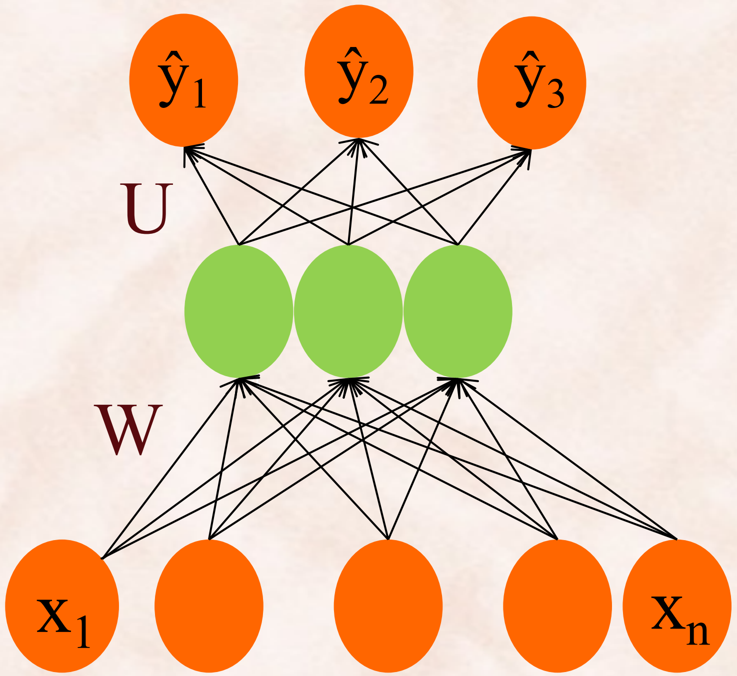
Adding a **hidden layer** to logistic regression allows the network to learn and use non-linear interactions between features:

- may (or may not) improve performance.

Multiclass Classification

- What if you have more than two output classes?
  - Positive, Neutral, Negative sentiment.
  - One output unit for each class + use a **softmax** layer:
    - $\hat{\mathbf{y}} = softmax(\mathbf{z}) = [\hat{y}_1 , \hat{y}_2 , \hat{y}_3]$
      - $\hat{y}_1$ probability of positive sentiment.
      - $\hat{y}_2$ probability of neutral sentiment.
      - $\hat{y}_3$ probability of negative sentiment.

$$
\begin{aligned}
\mathbf{x} &= [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_d] \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \mathrm{softmax}(\mathbf{z})
\end{aligned}
$$



Sentiment Analysis: FCN with Manually Engineered Features

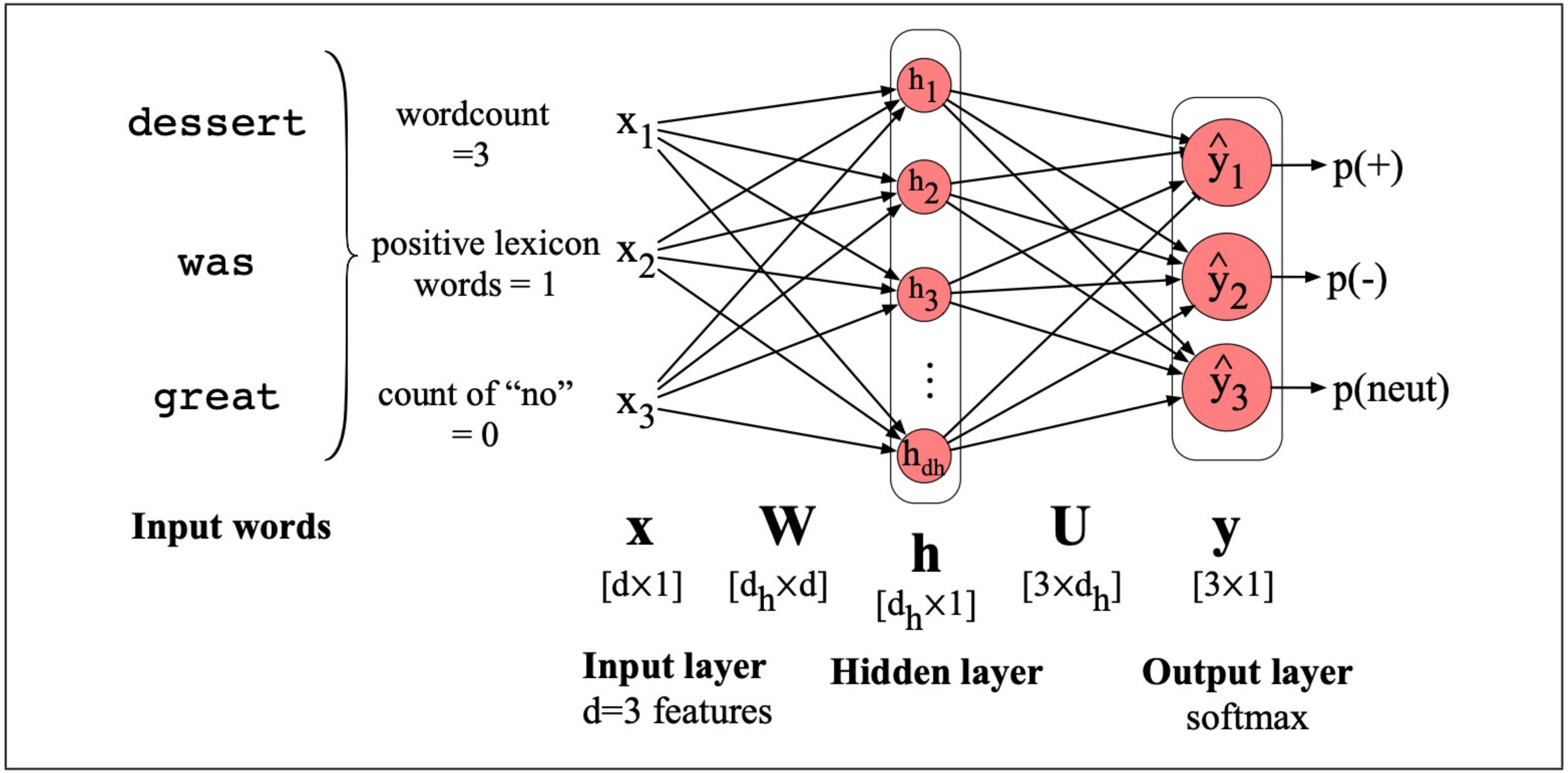
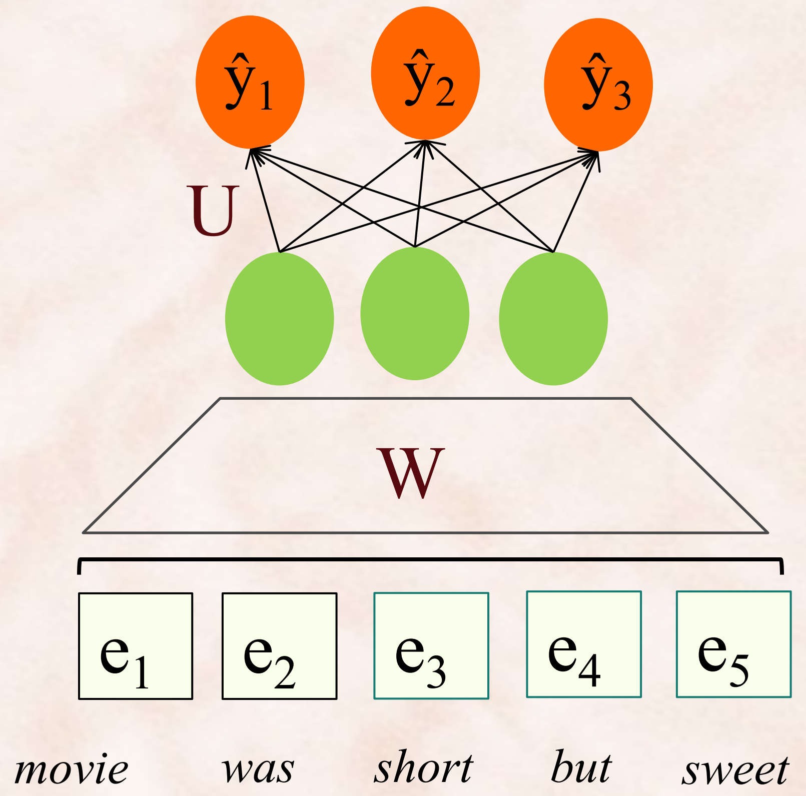


Figure 6.10 Feedforward network sentiment analysis using traditional hand-built features of the input text.

Sentiment Analysis: FCN with Word Embeddings

- The real power of deep learning comes from the ability to **learn** features from the data.
- Instead of:
  - **human-engineered** features.
- Use:
  - **learned representations**, like word embeddings!

How do we use these embeddings as input to an NN?



Embedding matrix **E**

- An embedding is a vector of dimension [1 x d] that represents the input token.
- An embedding matrix **E** is a dictionary, one row per token of vocab V.
  - **E** has shape $[|V| \times d]$

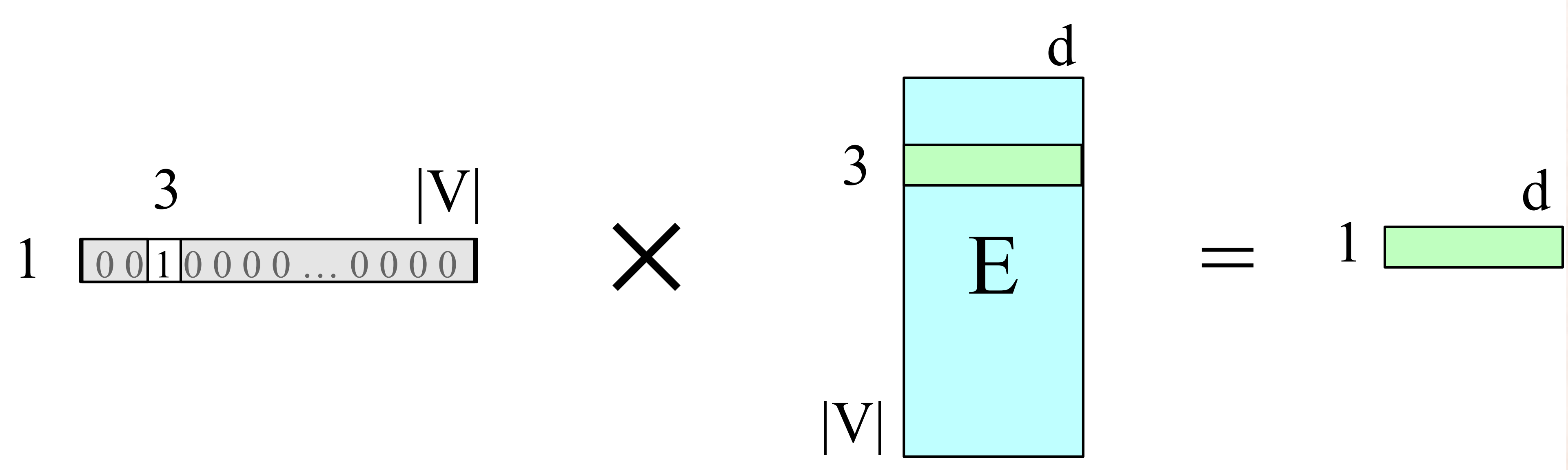
- Given tokenized input "*dessert was great*":
- Select the embedding vectors from **E:**
  - Convert BPE tokens into vocabulary indices.
    - w = [3, 9824, 226]
  - Use indexing to select the corresponding rows from **E**:
    - row 3, row 4000, row 10532

Another way to think of indexing from **E**

- Treat each input word as one-hot vector:
  - If `dessert` is index 3:

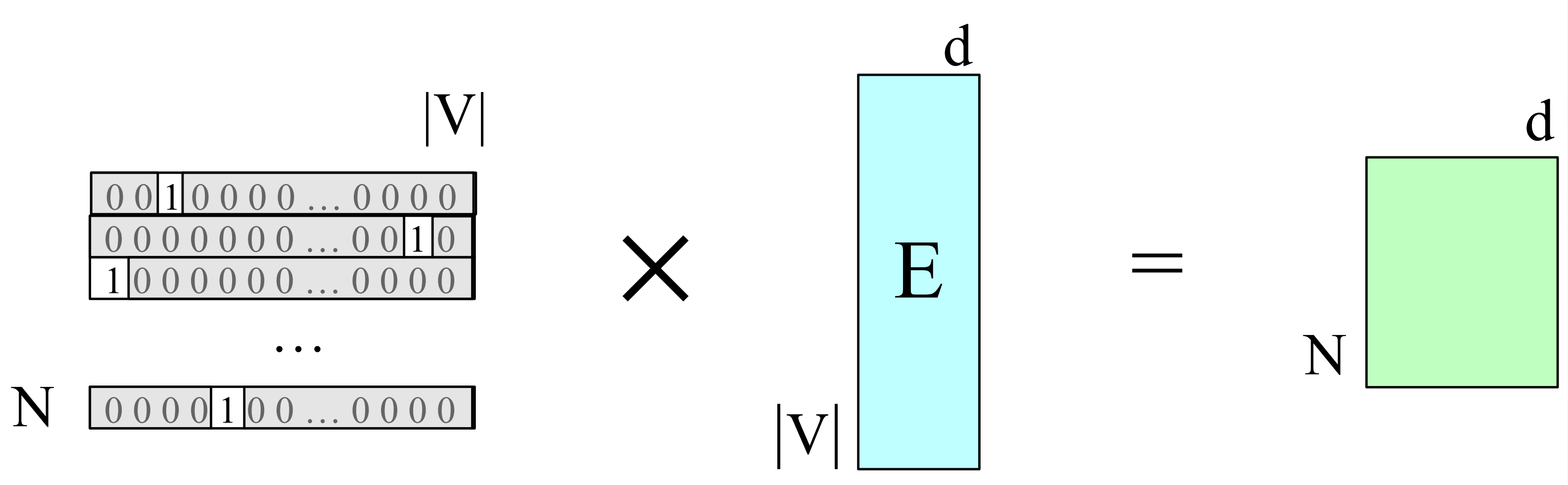
```
[0 0 1 0 0 0 0 ... 0 0 0 0]
 1 2 3 4 5 6 7 ...   ... |V|
```

- Multiply it by **E** to select out the embedding for `dessert`

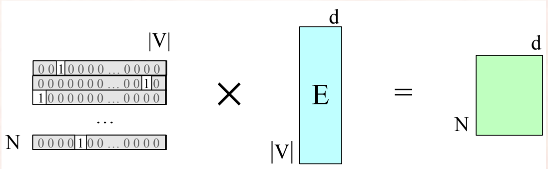


Another way to think of indexing from **E**

- Given window of N tokens, represented by N [$1 \times d$] embeddings, need to return a single class (e.g., *positive* or *negative*).



Text comes in different lengths

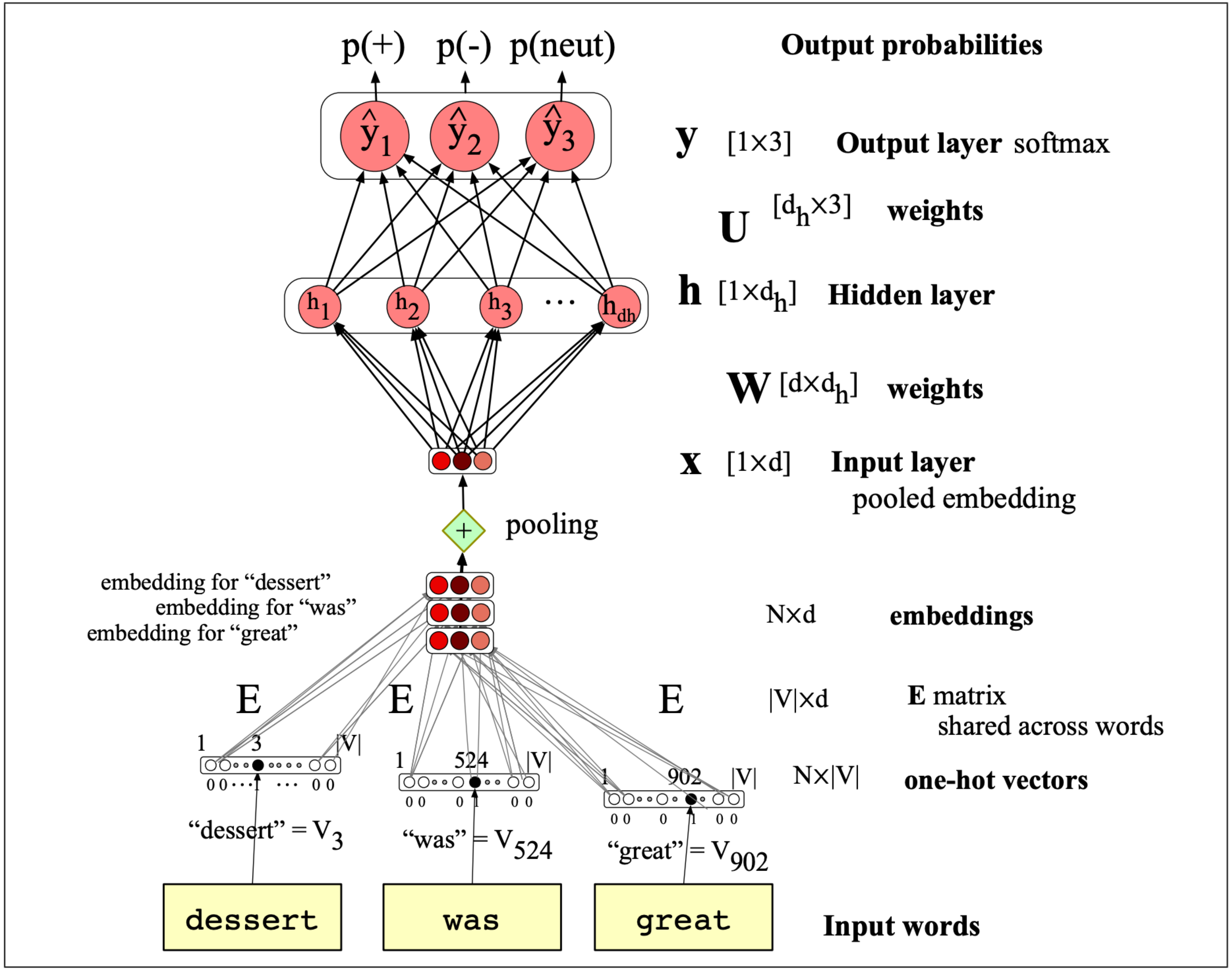


- Two approaches to get a fixed input size:
  1. **Concatenate** all input embeddings into one long vector [1×dN], e.g., N is length of longest review.
     - If shorter than N tokens, then pad with zero embeddings.
     - Truncate if you get longer reviews at test time

$$\mathbf{x} = [\boldsymbol{e}(w_1), \boldsymbol{e}(w_1), \ldots, \boldsymbol{e}(w_n),$$

  2. **Pool** the inputs into a single short [1 × d] vector.
     - A single "sentence embedding" (the same dimensionality as a word) to represent all the words.
     - Less information, but very efficient and fast.
     - Intuition: exact position not so important for sentiment.

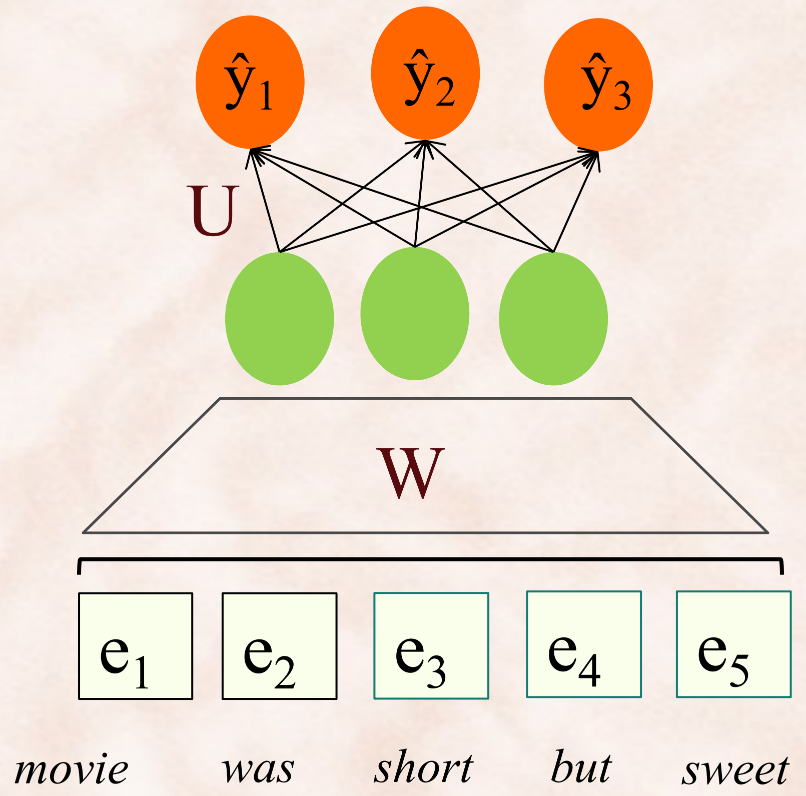
$$\begin{aligned}
\mathbf{x} &= \text{mean}(\mathbf{e}(w_1), \mathbf{e}(w_2), \ldots, \mathbf{e}(w_n)) \\
\mathbf{h} &= \sigma(\mathbf{xW} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{hU} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}$$



p(+) p(-) p(neut)
Output probabilities
y [1×3] Output layer softmax
U [dh×3] weights
h [1×dh] Hidden layer
W [d×dh] weights
x [1×d] Input layer
pooled embedding
pooling
embedding for "dessert"
embedding for "was"
embedding for "great"
N×d embeddings
E E E
|V|×d E matrix
shared across words
N×|V| one-hot vectors
"dessert" = V3
"was" = V524
"great" = V902
dessert
was
great
Input words

Sentiment Analysis: FCN with Word Embeddings

- Use **word embeddings** as input.
- Problem: text has *variable length*, whereas NN's need *fixed-size* input.
- Solutions:
  1. **Concatenate** embeddings over N positions:
     - Truncate to last N tokens, if text length > N.
     - Pad with dummy tokens, if text length < N.
  2. **Pool** embeddings across all positions:
     - Max-pooling.
     - Mean-pooling



Any issues with these two FCN approaches? Can we do better?

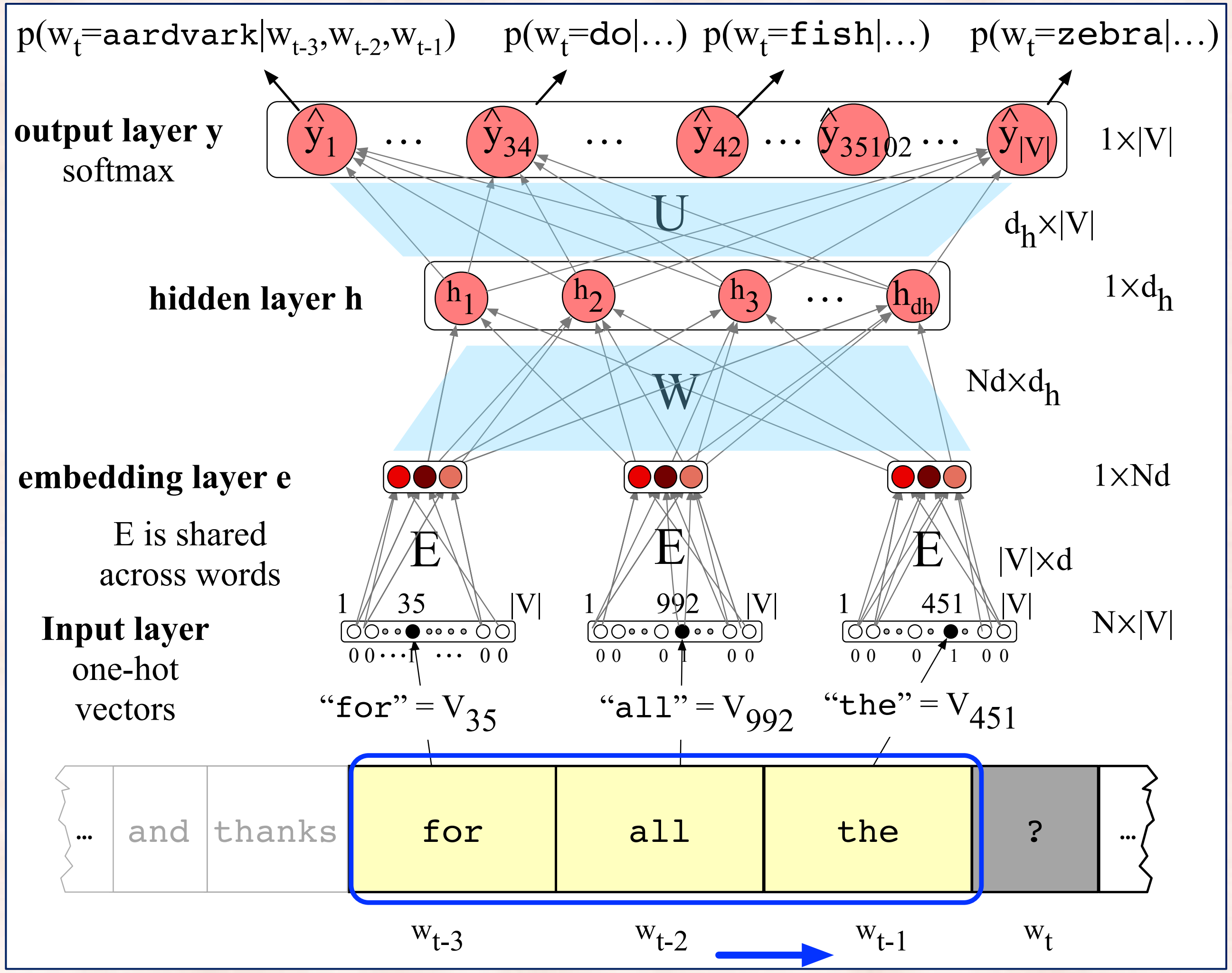
FCNs vs. RNNs vs. Transformer

- Limitations:
  - Concatenation => large number of parameters (linear in N)
  - Pooling => information loss (removes ordering).
- Better approaches:
  - **Recurrent Neural Networks** (RNNs).
  - **Transformer**.
- Both RNNs and Transformer:
  - The number of parameters is constant w.r.t. the length of the text.
    - The same network is used at every position in the text.
      - RNNs: **sequential** application.
      - Transformer: **parallel** application.
  - Theoretically, no information loss (order is important).

Neural Language Modeling with FCNs

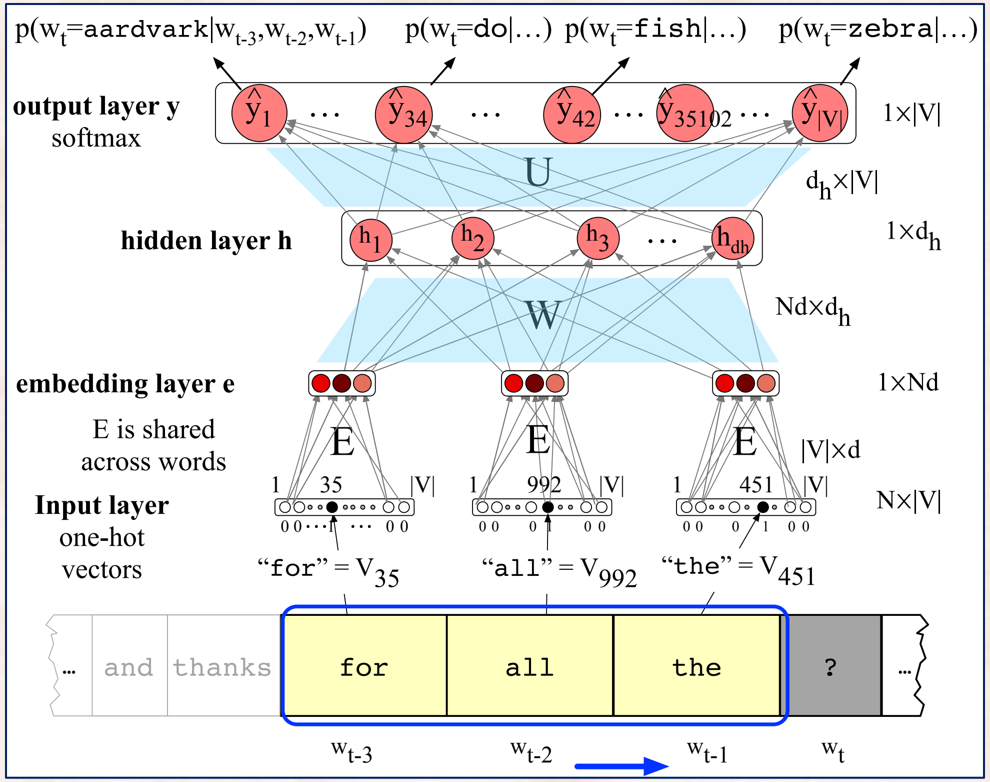
- **Task**: predict next word w_t given prior words w_{t-1}, w_{t-2}, w_{t-3}, … w_1
  - **Problem**: sequences of arbitrary length.
  - **Solution**: sliding window of fixed length.
    - Truncated history.
    - Low-order Markov model.

$$P(w_t|w_1,\ldots,w_{t-1}) \approx P(w_t|w_{t-N+1},\ldots,w_{t-1})$$



Inference in a Feedforward Language Model

$$
\begin{aligned}
\mathbf{e} &= [\mathbf{E}\mathbf{x}_{t-3};\mathbf{E}\mathbf{x}_{t-2};\mathbf{E}\mathbf{x}_{t-1}] \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{e}+\mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$



- Here, $\hat{y}_{42}$ is the probability of the next word w_t being V_{42} = `fish`

Supplementary Readings

- Chapter 6 in the textbook.