INF721

2024/2

UFV

Deep Learning

L15: Word Embeddings

INF721

2024/2

Deep Learning

L15: Word Embeddings

Logistics

Announcements

- PA4 is out and due on Wednesday (13/11), 11:59pm

Last Lecture

- Language Models
- Implementing RNNs
- Vanishing/Exploding Gradients
- LSTM and GRUs

Lecture Outline

- Problems with one-hot encoding
- Word Embeddings
  - Featurized Representation
  - Visualization
  - Properties
  - Applications
- Word2Vec and Negative Sampling
- GloVe

Problem with one-hot encoding

A problem of **one-hot encoding** is that it represents each word as an independent category

$$
\begin{array}{cccccc}
\textbf{Man} & \textbf{Woman} & \textbf{King} & \textbf{Queen} & \textbf{Apple} & \textbf{Orange} \\
(5391) & (9853) & (4914) & (7157) & (456) & (6257) \\
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \\ 0 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \cdots \\ \mathbf{1} \\ \cdots \\ 0 \end{bmatrix} \\
\mathbf{o}_{5391} & \mathbf{o}_{5391} & \mathbf{o}_{5391} & \mathbf{o}_{5391} & \mathbf{o}_{5391} & \mathbf{o}_{5391}
\end{array}
$$

It doesn't let a model easily generalize across words!

- For example, consider a language model that gives high probability to the sentence:

  *"I want a glass of **orange** juice"*

  Now, consider sampling from this model with context:

  *"I want a glass of **apple** ______ "*

  To the model, the relationship between **apple** and **orange** is the same as **apple** and **man**, or queen.

  This is because the distance between any two words is the same!

Vocabulary $V = [a, aaron, \ldots, zebra, zulu], \quad |V| = 10{,}000$

Word Embedding: Featurized Representation

Ideally, we would like to have featurized representation for words, where words with similar meaning to have a similar representation:

apple and **orange** are close in the embedding space

	Man (5391)	**Woman** (9853)	**King** (4914)	**Queen** (7157)	**Apple** (456)	**Orange** (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
…						
	$\mathbf{e}_{5391}$	$\mathbf{e}_{9853}$	$\mathbf{e}_{4914}$	$\mathbf{e}_{7157}$	$\mathbf{e}_{456}$	$\mathbf{e}_{6257}$

Word embeddings are learned featurized representations:

- We can define the number of features d to be learned (e.g., 300), but **not** what they represent (e.g., gender)
- Words with similar meaning to have a similar representation:
- They allow a model to generalize across words more easily:

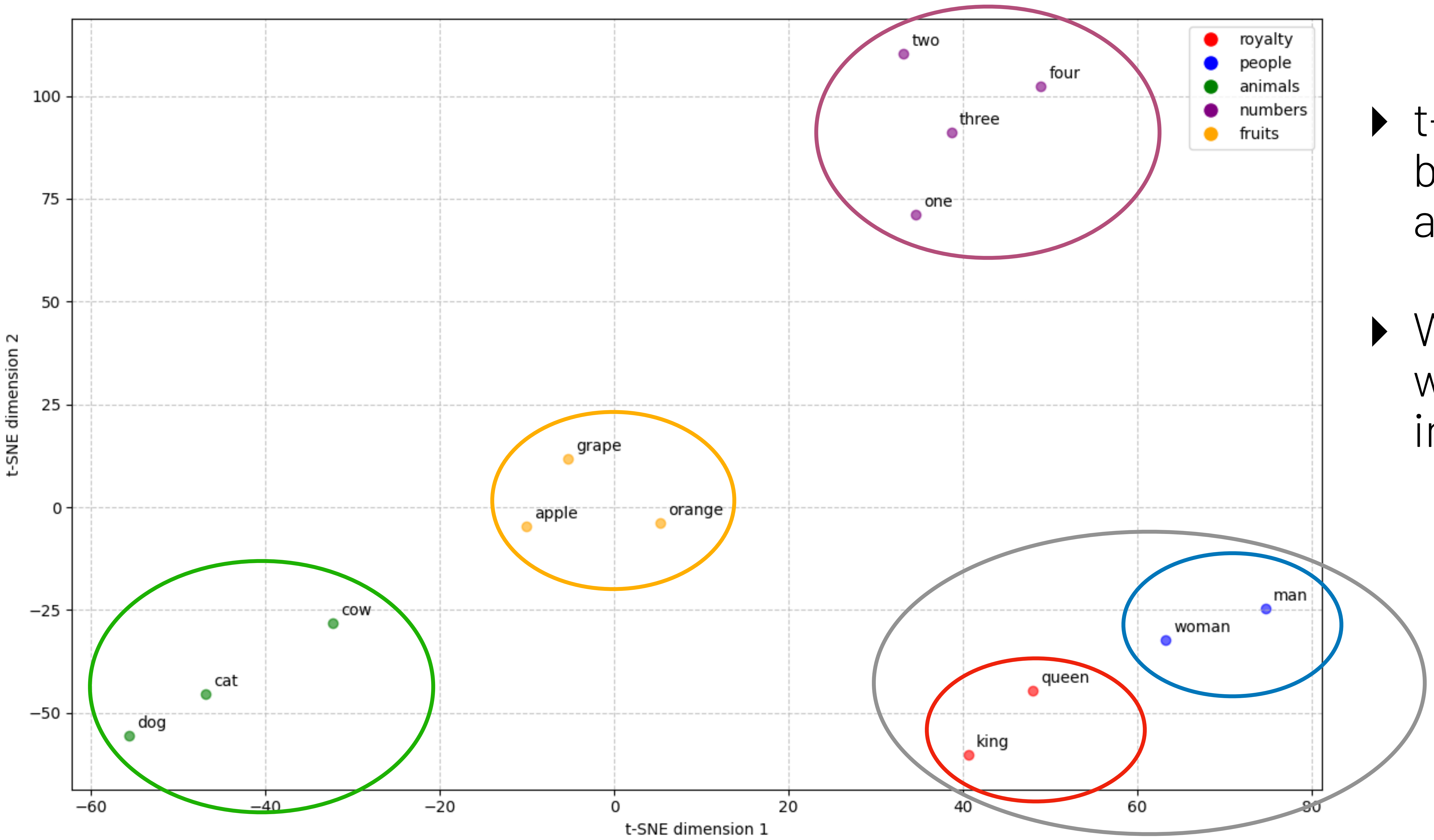
  *"I want a glass of **orange** juice"*

  *"I want a glass of **apple** _juice_ "*

UFV

Visualizing Word Embeddings (with t-SNE)

We can visualize high dimentinoal word embeddings (e.g., 300 features) in 2D using the t-SNE algorithm:



- t-SNE learns a non-linear mapping between the original representation and a 2D space
- Word embeddings tend to group words with similar meaning together in the embedding space.

van der Maaten and Hinton., 2008. Visualizing data using t-SNE

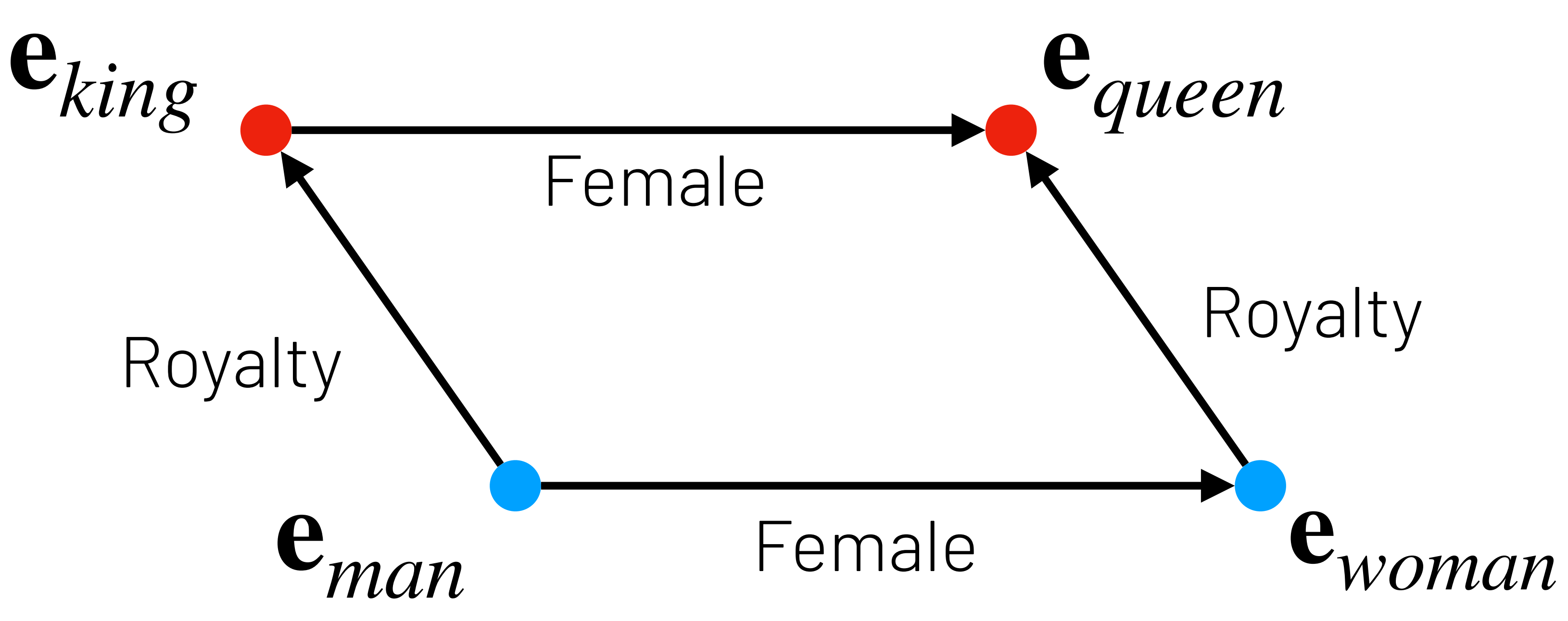
Properties of word embeddings

One interesting property of word embeddings is that they allow **word analogies** to be solved with vector arithmetic.

	Man (5391)	**Woman** (9853)	**King** (4914)	**Queen** (7157)	**Apple** (456)	**Orange** (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
…						
	$\mathbf{e}_{man}$	$\mathbf{e}_{woman}$	$\mathbf{e}_{king}$	$\mathbf{e}_{queen}$	$\mathbf{e}_{apple}$	$\mathbf{e}_{orange}$

Man is to Woman as King is to ______?

$$\mathbf{e}_{king} - \mathbf{e}_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{e}_{man} - \mathbf{e}_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



$$\mathbf{e}_{man} - \mathbf{e}_{woman} \approx \mathbf{e}_{king} - \mathbf{e}_{w}$$

$$\mathbf{e}_{w} = argmax_{w}\ \underbrace{sim(\mathbf{e}_{w}, \mathbf{e}_{king} - \mathbf{e}_{man} + \mathbf{e}_{woman})}_{\text{Similarity}}$$

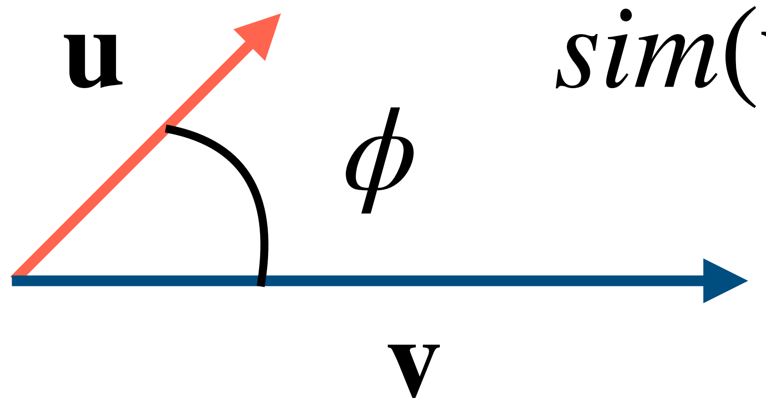
Similarity Between Vectors

Measuring similarity between vectors is very important in deep learning and one of the most populars functions to do that is the **cosine similarity**:

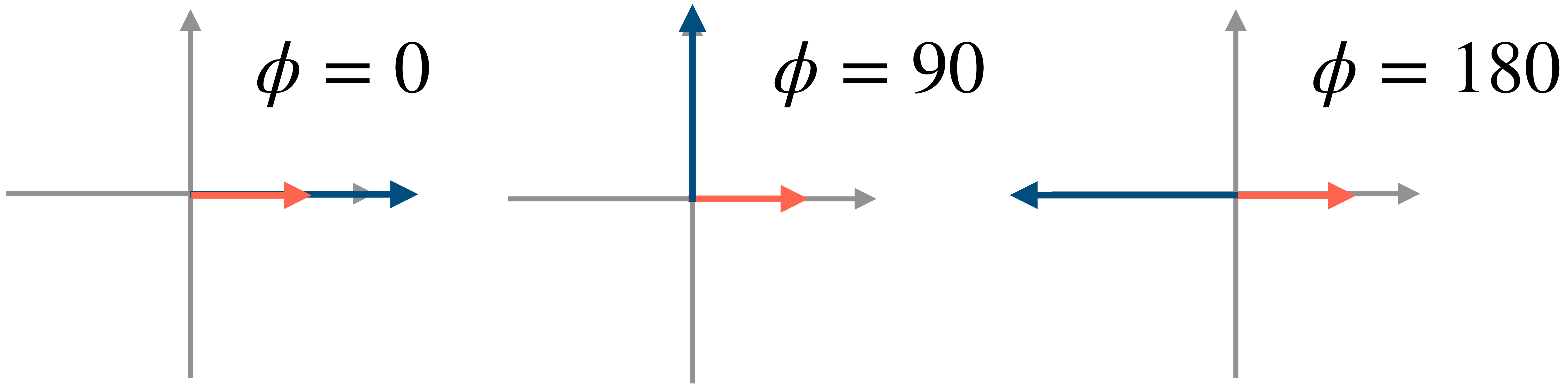
The **cosine similarity** between two vectors u and v is defined as:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||_2 ||\mathbf{v}||_2}$$

This formula is equal to the cossine of the anble between u and v:



$$sim(\mathbf{u}, \mathbf{v}) = cos(\phi)$$

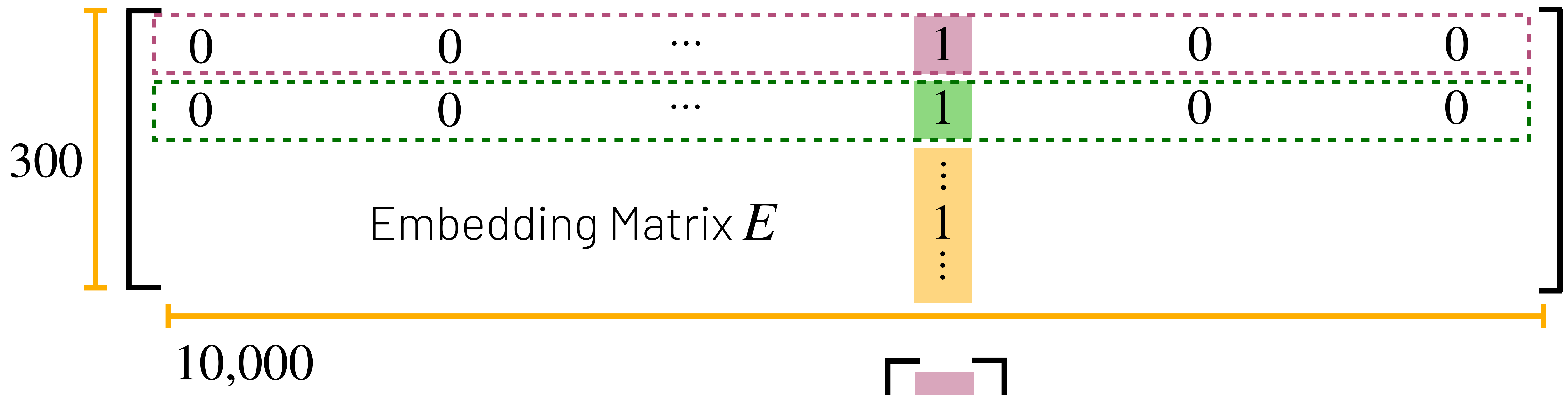


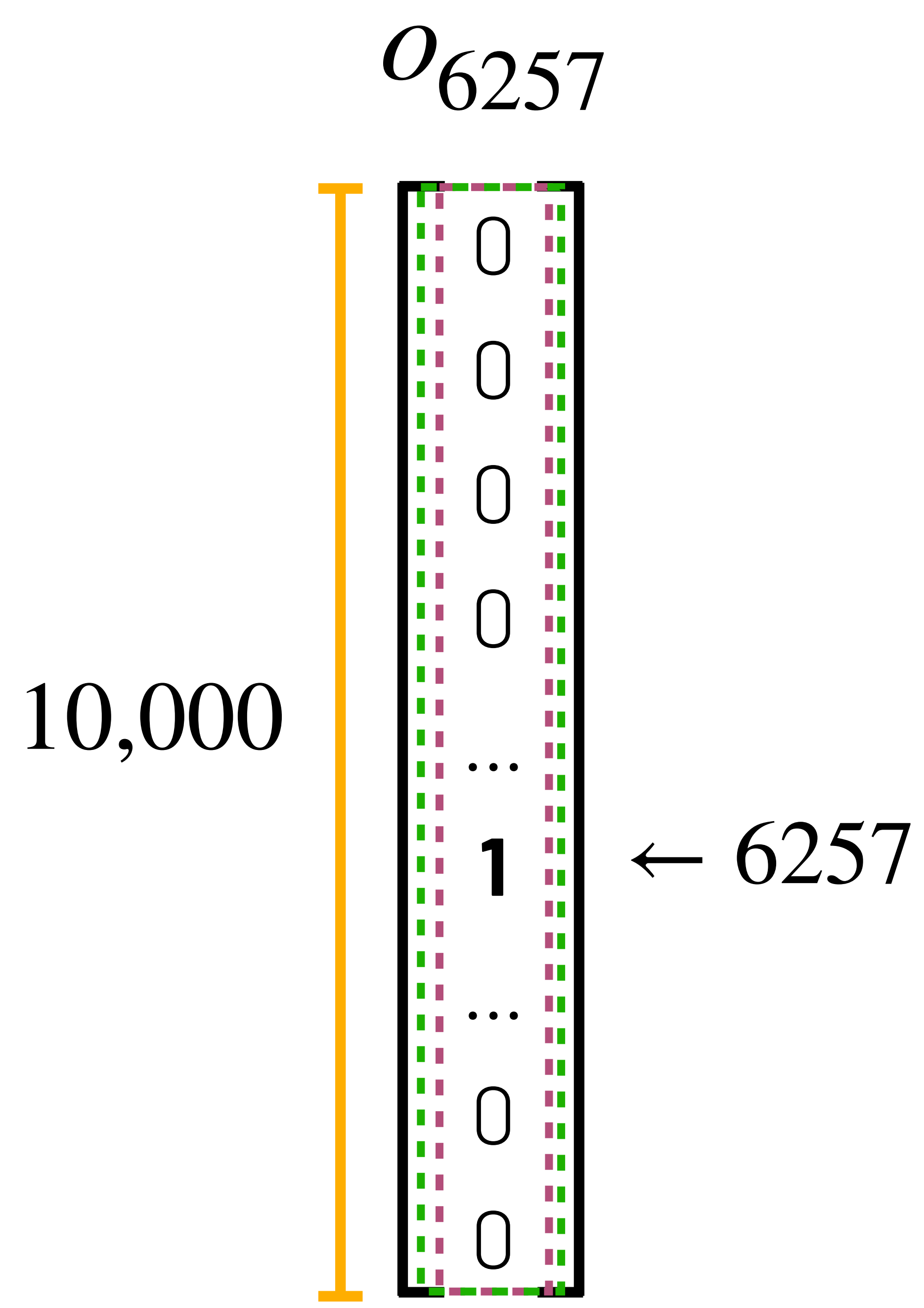
$sim(\mathbf{u}, \mathbf{v}) = 1$ $sim(\mathbf{u}, \mathbf{v}) = 0$ $sim(\mathbf{u}, \mathbf{v}) = -1$

UFV

Embedding Matrix

Word embeddings with dimentionality d (e.g. 300) are learned by adding an extra weight matrix , called **embeddings matrix** E, to the model:



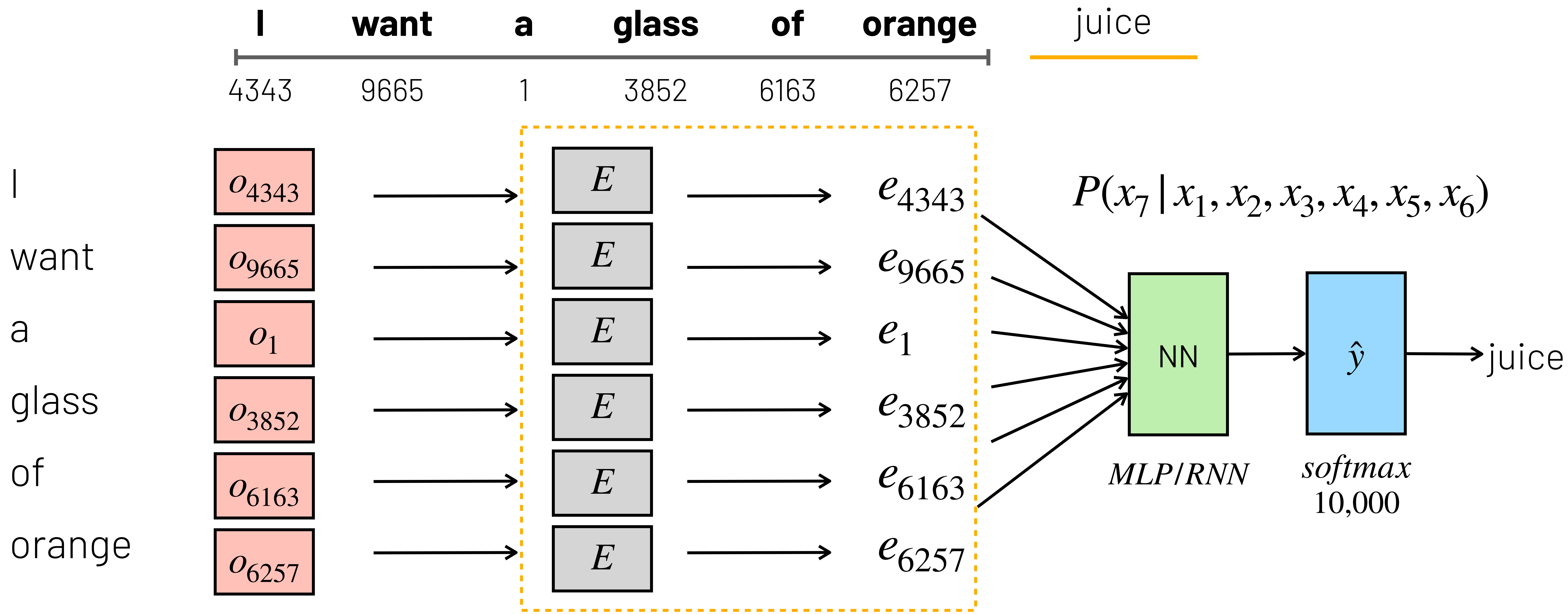


$$E \cdot o_{6257} = e_{6257}$$

To compute the embedding e for a word, we multiply the embedding matrix on the left

Learning Word Embeddings

There are many ways to learn word embeddings, but a simple and popular one is to train a **language model** with an **embedding layer** before the model:



Bengio et. al., 2003, A neural probabilistic language model

Learning Word Embeddings

If your goal is not to learn a language model, but just the embeddings, we don't need to be limited to the previous words as **context**. We can use all the surrounding words:

Last words: (Language Model)	**I**	**want**	**a**	**glass**	**of**	**orange**	**juice**	**to**	**drink**	**with**	**my**	**cereal**
			Context				Target					
Surrounding words: (Word2Vec)	**I**	**want**	**a**	**glass**	**of**	**orange**	**juice**	**to**	**drink**	**with**	**my**	**cereal**
			Context				Target	Context				

Word2Vec

Word2Vec is a model to learn word embeddings using the surronding words as context

- Examples are produced by:
  1. Randomly sampling a **target word** and;
  2. Extracting the m (e.g, 3) words to left and to the right as **context**

I want a glass of orange juice to drink with my cereal

[Mikolov et. al., 2013] evaluated two different types of models:

- **Continuous Bag of Words (CBOW)**: The goal is to predict the **target word** from the **context**
- **Skip-gram**: The goal is to predict the **context words** from the **target word**!

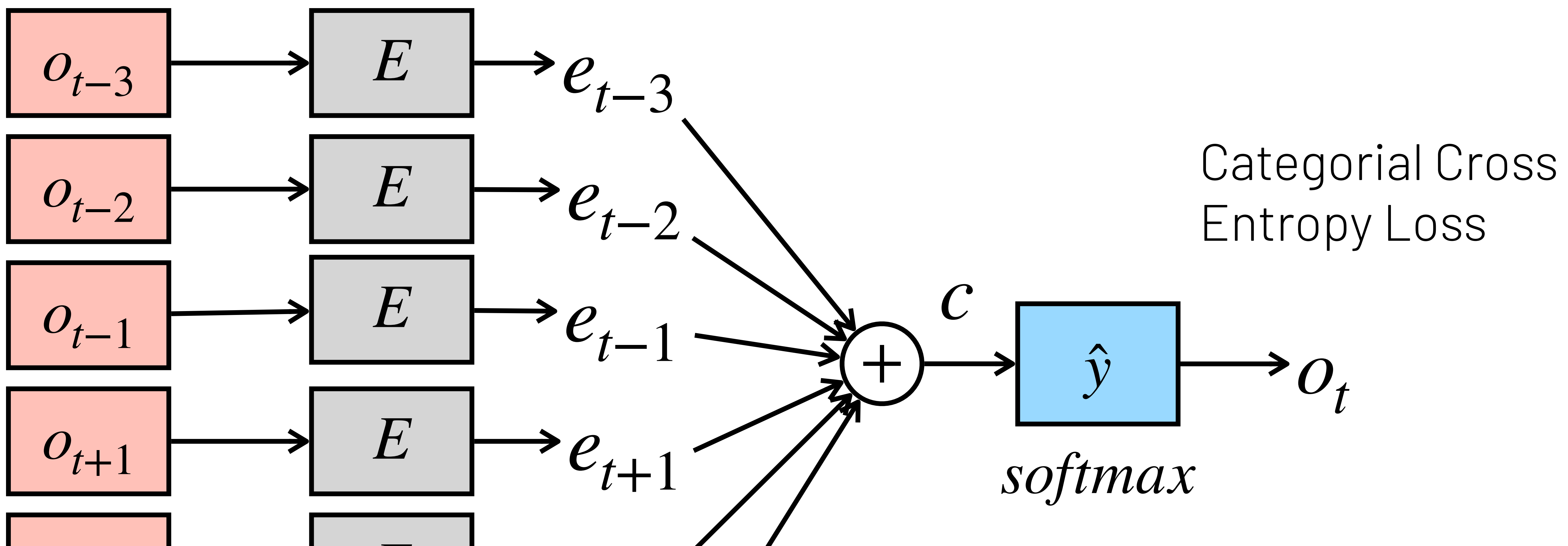
Mikolov et. al., 2013. Efficient estimation of word representations in vector space.

Word2Vec: CBOW

In **CBOW** model, the goal is to predict the **target word** from the **context.**

I want a glass of orange juice to drink with my cereal

Context	Target
...	...
glass of orange _____ to drink with	juice
...	...



$$P(o_t \mid c) = \frac{e^{\mathbf{w}_t \cdot \mathbf{e}_c + b_t}}{\sum_{j=1}^{10{,}000} \mathbf{e}^{\mathbf{w}_t \cdot e_c + b_t}}$$

UFV

Word2Vec: Skip-Gram

In the **Skip-gram** model, the goal is to predict the **context words** from the **target word**!

I want a glass of orange juice to drink with my cereal

Target	Context
juice	glass
juice	of
juice	orange
juice	to
juice	drink
juice	with

$$o_c \rightarrow \boxed{E} \rightarrow e_c \rightarrow \underbrace{\boxed{\hat{y}}}_{softmax} \rightarrow o_t$$

Categorial Cross Entropy Loss

$$P(o_t | c) = \frac{e^{\mathbf{w}_t \cdot \mathbf{e}_c + b_t}}{\sum_{j=1}^{10,000} \mathbf{e}^{\mathbf{w}_t \cdot e_c + b_t}}$$

Negative Sampling

$$P(t \mid c) = \frac{e^{\mathbf{w}_t \cdot \mathbf{e}_c + b_t}}{\sum_{j=1}^{10{,}000} \mathbf{e}^{\mathbf{w}_t \cdot e_c + b_t}}$$

(we have to sum 10,000 terms)

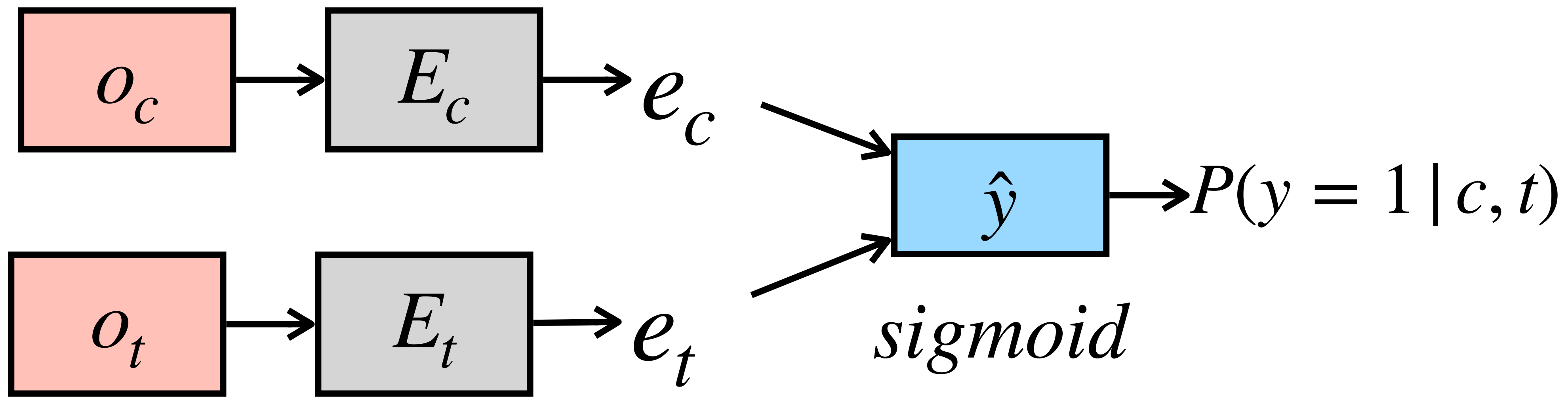
Since the vocabulary size is typically large (e.g., 10,000), computing a probability with the softmax layer is expensive!

Negative sampling changes the classification text to a binary classification problem by generating a dataset with positive and negative examples:

Context	**Word**	**Target?**
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

Sample with skip-gram

Sample k random words from the dictionary



$$P(y = 1 \mid c, t) = \sigma(\mathbf{e}_t \cdot \mathbf{e}_c)$$

GloVe (Global Vectors for Word Representation)

In Word2Vec, the global information is not preserved. To address that, GloVe predicts the number of times a target word t appears in the context of the context word c

1. Build a word co-occurence matrix X

I want a glass of orange juice

2. Assign a weight vector per context $\mathbf{w}_i$ and target word $\tilde{\mathbf{w}}_j$

3. Predict log co-occurance from $\mathbf{w}_i$ and $\tilde{\mathbf{w}}_j$ using weighted MSE

		$\tilde{\mathbf{w}}_1$ I	$\tilde{\mathbf{w}}_2$ want	$\tilde{\mathbf{w}}_3$ a	$\tilde{\mathbf{w}}_4$ glass	$\tilde{\mathbf{w}}_5$ of	$\tilde{\mathbf{w}}_6$ orange	$\tilde{\mathbf{w}}_7$ juice
$\mathbf{w}_1$	I	0	1.0	0.5	0	0	0	0
$\mathbf{w}_2$	want	1.0	0	1.0	0.5	0	0	0
$\mathbf{w}_3$	a	0.5	1.0	0	1.0	0.5	0	0
$\mathbf{w}_4$	glass	0	0.5	1.0	0	1.0	0.5	0
$\mathbf{w}_5$	of	0	0	0.5	1.0	0	1.0	0.5
$\mathbf{w}_6$	orange	0	0	0	0.5	1.0	0	1.0
$\mathbf{w}_7$	juice	0	0	0	0	0.5	1.0	0

$$X_{ij} = \sum_{j}^{C} \frac{1}{dist(i,j)}$$

where C is context window size (e.g, 2)

$$L = \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) \, (\underbrace{\mathbf{w}_i \tilde{\mathbf{w}}_j + b_i + b_j}_{\text{Prediction}} - \underbrace{log(X_{ij})}_{\text{Target}})^2$$

Using log helps compress the range of values

weighting function that helps balance the importance of different co-occurrences

4. Embedding:

$$e_i = \frac{w_i + \tilde{w}_i}{2}$$

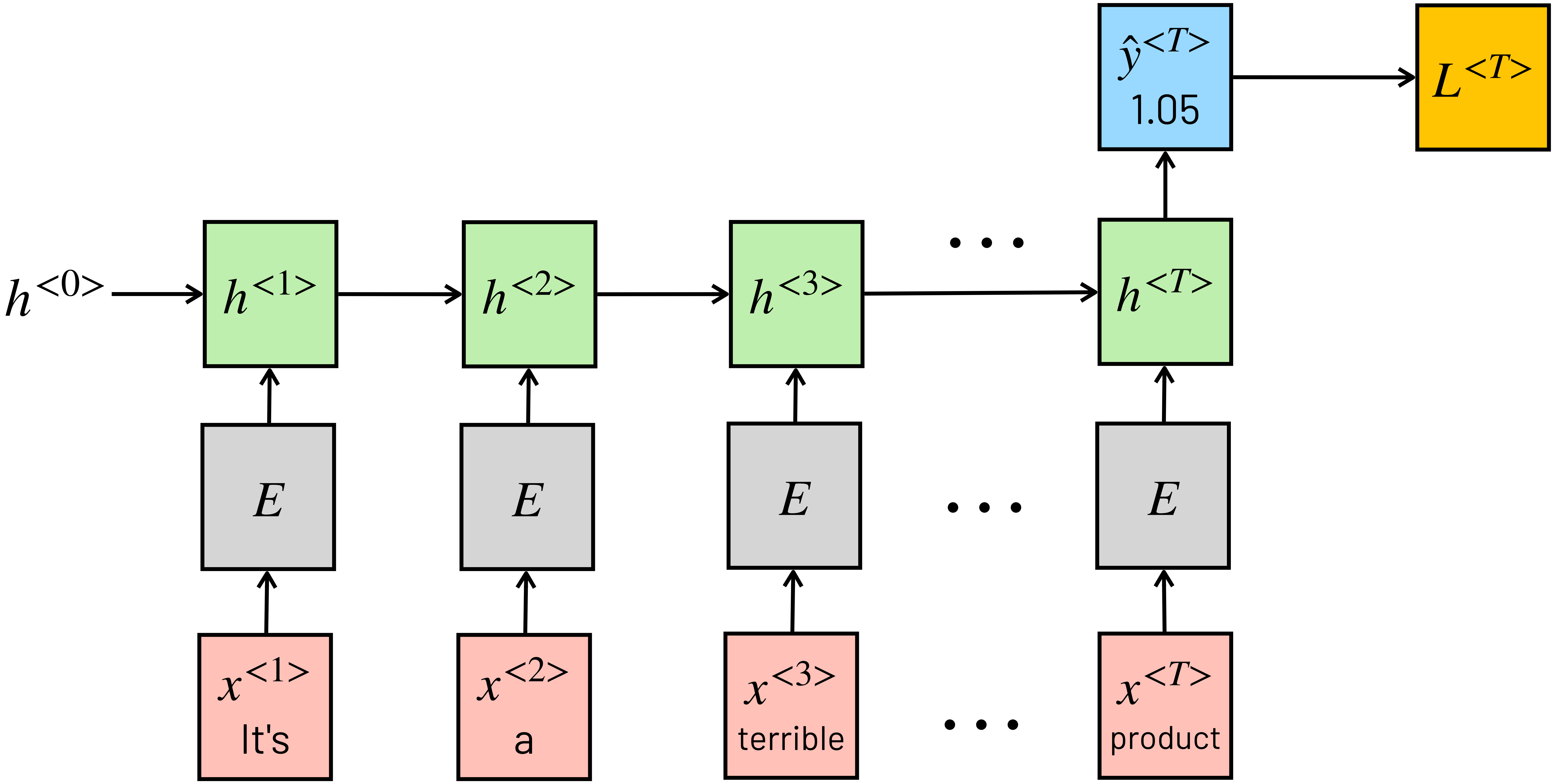
UFV

Pennington et. al., 2014. GloVe: Global vectors for word representation

Using Word Embeddings

Sentiment Analysis "It's a terrible product." ★☆☆☆☆

When working with Natural Language Processing problems, you can learn word embeddings by just adding an embedding layer to your network:



UFV

Using Word Embeddings

Sentiment Analysis "It's a terrible product." ★☆☆☆☆

In PyTorch, you can use the Embedding layer to add an Embedding Matrix E to your model:

```python
class RNNSentiment(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, padding_idx=None):
        super().__init__()

        # Embedding layer
        self.embedding = nn.Embedding(
            num_embeddings=vocab_size, embedding_dim=embedding_dim, padding_idx=padding_idx
        )

        # RNN layer
        self.rnn = nn.RNN(input_size=embedding_dim, hidden_size=hidden_dim, num_layers=1,
            batch_first=True
        )

        # Output layer
        self.fc = nn.Linear(hidden_dim, output_dim)
```

Next Lecture

L15: Attention Mechanisms

Machine Translation, Decoding Strategies, Attention Mechanisms in RNNs