Implementing ANNs with TensorFlow

Session 10 - Word Embeddings

Agenda

1. Motivation
2. Statistical Language Model
3. Word Embeddings
4. Continuous Bag of Words
5. Skip Gram
6. Bonus: Seq2Seq, Attention, Transformers

Motivation

- Character-level language models can capture superficial features of the language they are trained on.
- But they fail to capture the **semantic** and **syntactic structure**

Not available due to copyright issues.

Statistical Language Model

Statistical Language Model

- Just as a character-level model a <u>statistical language model</u> is about learning the probabilities of sequences of words.

- These probabilities can be very useful for many purposes.

- E.g. knowing

  P(*'The cat climbed a tree.'*) >> P(*'The cat climbed a brie.'*)

  might be helpful for speech recognition.

Applications

- Speech recognition.
- Machine translation.
- Handwritten text recognition.
- Keyboards on smartphones predicting the next word.

Statistical Language Models

- What is the probability of a sentence?
- Sentence = Sequence of words.

Chain rule

$$
\begin{aligned}
P(s) &= P(w_1, w_2, \ldots, w_n) \\
&= P(w_n \,|\, w_{n-1}, \ldots, w_1) \\
&\cdot P(w_{n-1} \,|\, w_{n-2}, \ldots, w_1) \\
&\cdots \; P(w_2 \,|\, w_1) \cdot P(w_1) \\
&= \prod_{i=1}^{n} P(w_i \,|\, w_{i-1}, \ldots, w_1)
\end{aligned}
$$

Statistical Language Models

- Just as the character-level model a language model is always based on a corpus.

- The probabilities $P(w_i \,|\, w_{i-1}, \ldots, w_1)$ can be computed by simple counting.

- The conditional probability of a word given a sequence of words is given by number of times it occurs after this specific sequence of words.

$$P(w_i \,|\, w_{i-1}, \ldots, w_1) = \frac{count(w_1, \cdots, w_{i-1}, w_i)}{count(w_1, \cdots, w_{i-1})}$$

N-Gram Models

- An n-gram model is an approximation of that model.
- It assumes that a word is only dependent on the the $n-1$ previous words.

$$
\begin{aligned}
P(s) &= P(w_1, w_2, \ldots, w_n) \\
&= \prod_{i=1}^{n} P(w_i \,|\, w_{i-1}, \ldots, w_1) \\
&\approx \prod_{i=1}^{n} P(w_i \,|\, w_{i-1}, \ldots, w_{i-(n-1)})
\end{aligned}
$$

Problem

- There is a problem with these statistical approaches:
- Let’s say we find the following sentence in our text corpus:

Paris is the capital of France.

- Knowing that this sentence is valid would not have any influence on a sentence like:

Rome is the capital of Italy.

- These models are not good in generalizing to unseen sentences/combinations of words.

Problems

- The reason is that these models have no means of understanding whether two words are similar to each other (e.g. *Paris* and *Rome*, *France* and *Italy*)

- In mathematical terms this is analogous to a **one-hot encoding** (= all words are equally dissimilar).

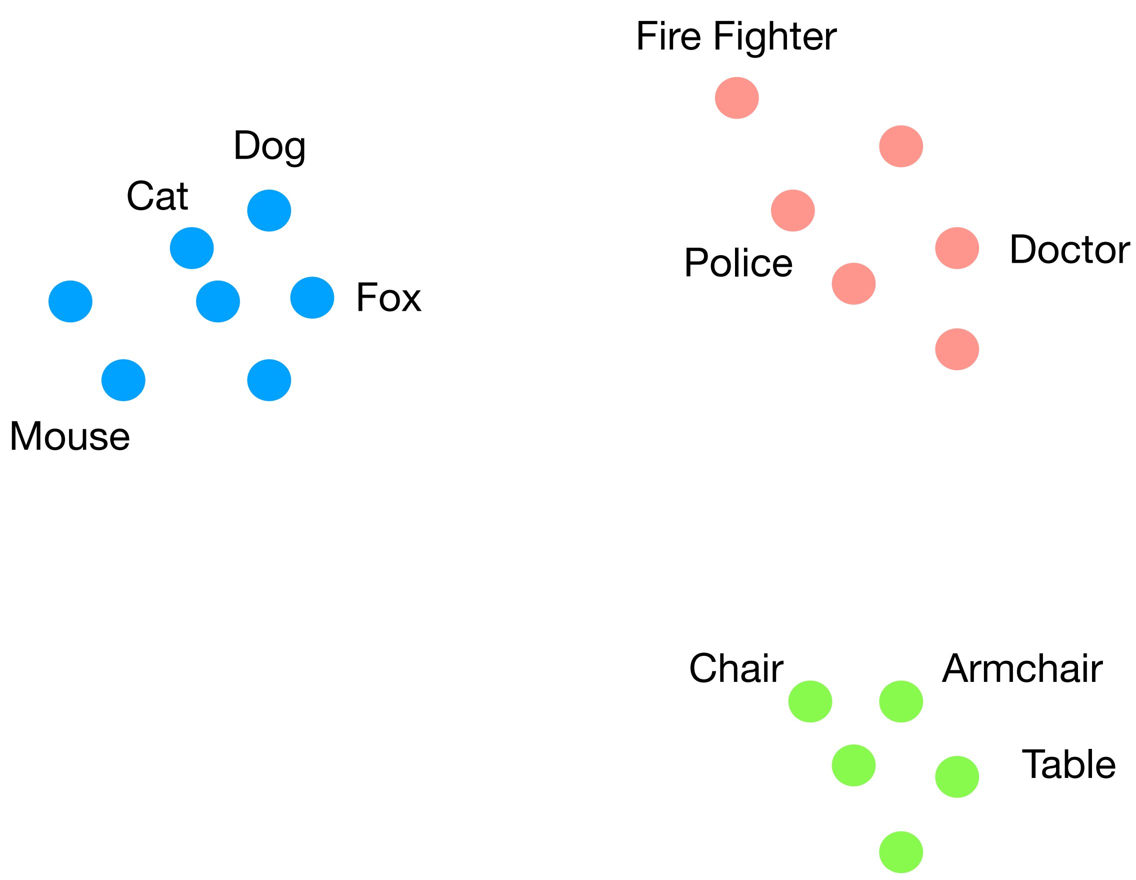
- E.g.

$$\textit{Paris} \;\widehat{=}\; \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \qquad \textit{Italy} \;\widehat{=}\; \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad \textit{France} \;\widehat{=}\; \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Word Embeddings

Embeddings

- The solution is to embed words into a high-dimensional space in which the similarity of words is represented by their distances.



Only 2-dimensional for purpose of visualization!

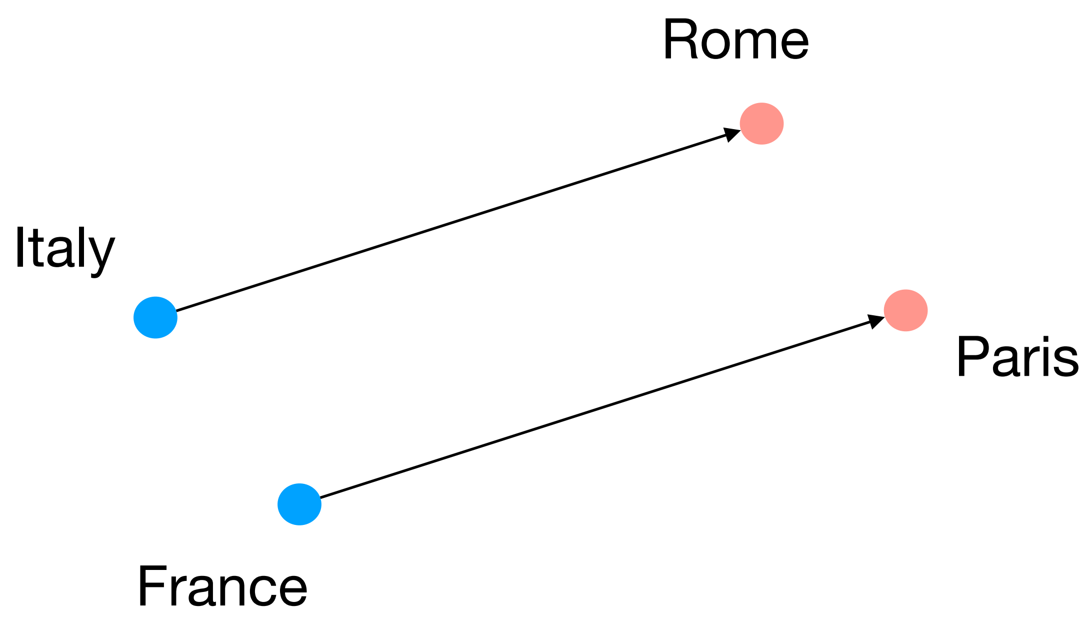
'Paris'

$$\begin{pmatrix} -0.05 & 1.1 & \cdots & 0.22 \end{pmatrix}$$

Embedding

Semantic Information

- In the optimal case such an embedding would not only reflect some form of similarity, but even capture <u>semantic information</u>.



Rome* is the same to *Italy* as *Paris* is to *France.

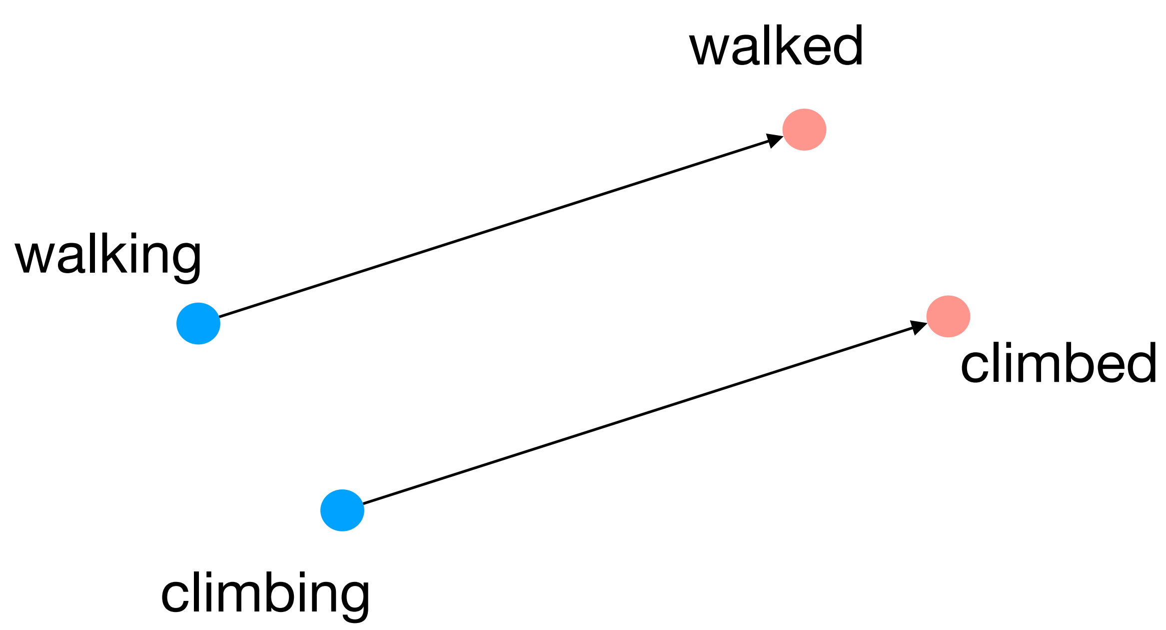
<u>Formalization:</u>

$$\underbrace{\phi(\text{'Rome'}) - \phi(\text{'Italy'})} + \phi(\text{'France'}) = \phi(\text{'Paris'})$$

This vector captures the concept of capital.

Syntactic Information

- Similarly <u>syntactic information</u> can be stored this way.



Walked **is the same to** ***walking*** **as** ***climbed*** **is to** ***climbing.***

<u>**Formalization:**</u>

$$\underbrace{\phi(\text{'walked'}) - \phi(\text{'walking'})}_{\textbf{This vector captures the concept of past tense.}} + \phi(\text{'climbing'}) = \phi(\text{'climbed'})$$

Word Embeddings

Can we learn such a representation?

Yes!

Continuous Bag of Words

Distributional Hypothesis

- The distributional hypothesis says, that similar words occur in similar contexts.

- E.g.

The cat climbed the tree.

The dog climbed the tree.

'*Cat*' and '*dog*' are similar words.

- This is quite powerful, because similarity between words can be used to establish new similarity, e.g.:

Johnny has a cat.

Sarah has a dog.

Because we know that '*cat*' and '*dog*' are similar words we can now infer that '*Johnny*' and '*Sarah*' are similar words.

Continuous Bag of Words

- In the <u>continuous bag of words</u> model (CBOW) we try to predict a word from its context.
- The context are the words before and after the word.

Sentence from text corpus: *The cat climbed the tree.*

Context | **Target Word**

The cat … the tree. $\xrightarrow{\textbf{predict}}$ *climbed*

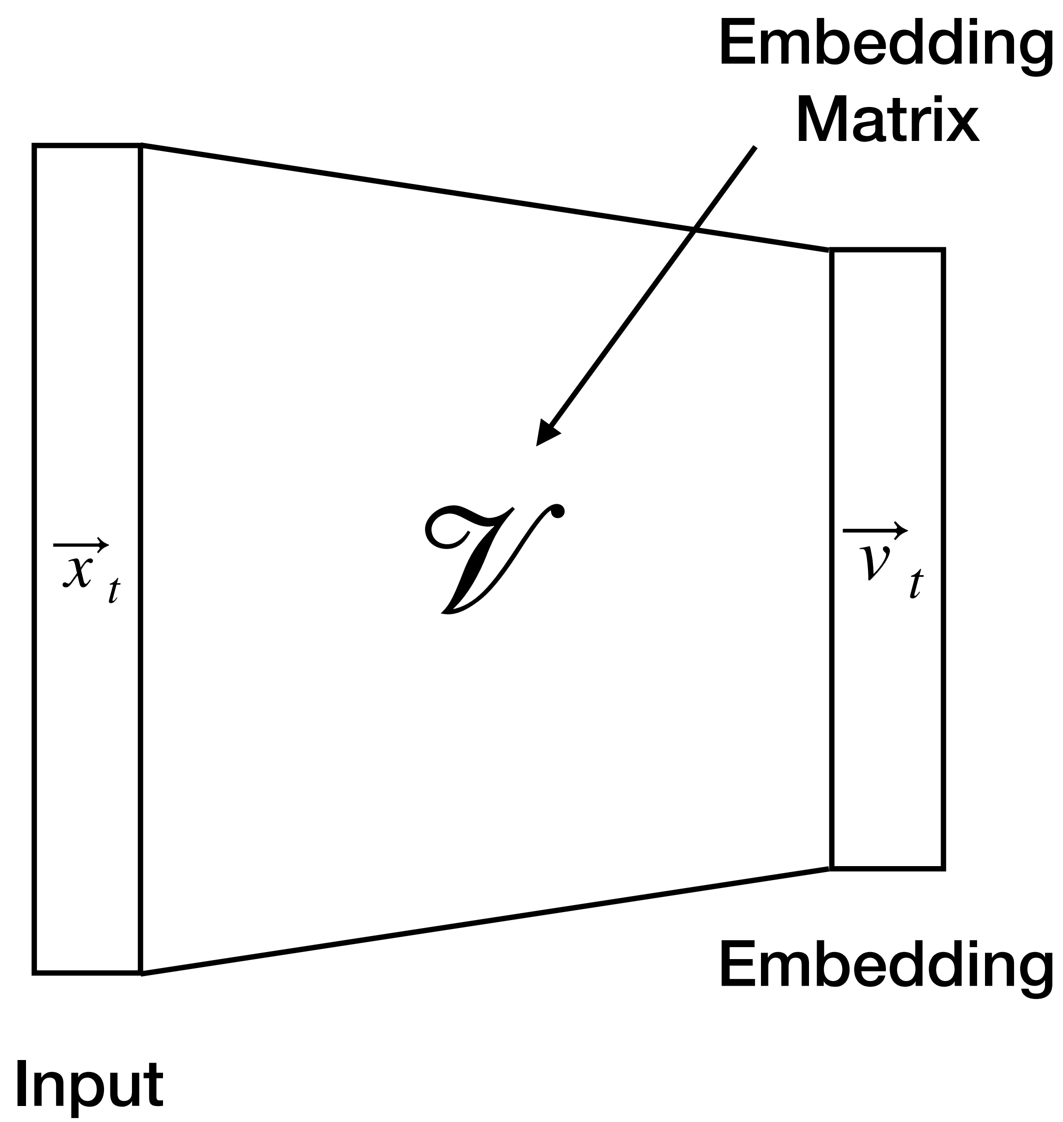
Continuous Bag of Words

- In practice we are given a text corpus with a vocabulary $V = \{w_1, \ldots, w_{|V|}\}$.
- The text is then represented as a sequence $\overrightarrow{x}_1, \overrightarrow{x}_2, \overrightarrow{x}_3, \ldots, \overrightarrow{x}_N$, where each $\overrightarrow{x}_t$ is a one-hot vector representing the corresponding word w_i .
- Based on a defined a context window, e.g. $c = 2$, we generate training pairs

Inputs	**Targets**
$(\overrightarrow{x}_1, \overrightarrow{x}_2, \overrightarrow{x}_4, \overrightarrow{x}_5)$	$\overrightarrow{x}_3$
$(\overrightarrow{x}_2, \overrightarrow{x}_3, \overrightarrow{x}_5, \overrightarrow{x}_6)$	$\overrightarrow{x}_4$
$\vdots$	$\vdots$
$(\overrightarrow{x}_3, \overrightarrow{x}_4, \overrightarrow{x}_7, \overrightarrow{x}_8)$	$\overrightarrow{x}_6$

Embedding

- The <u>embedding</u> is the high dimensional representation of the word.

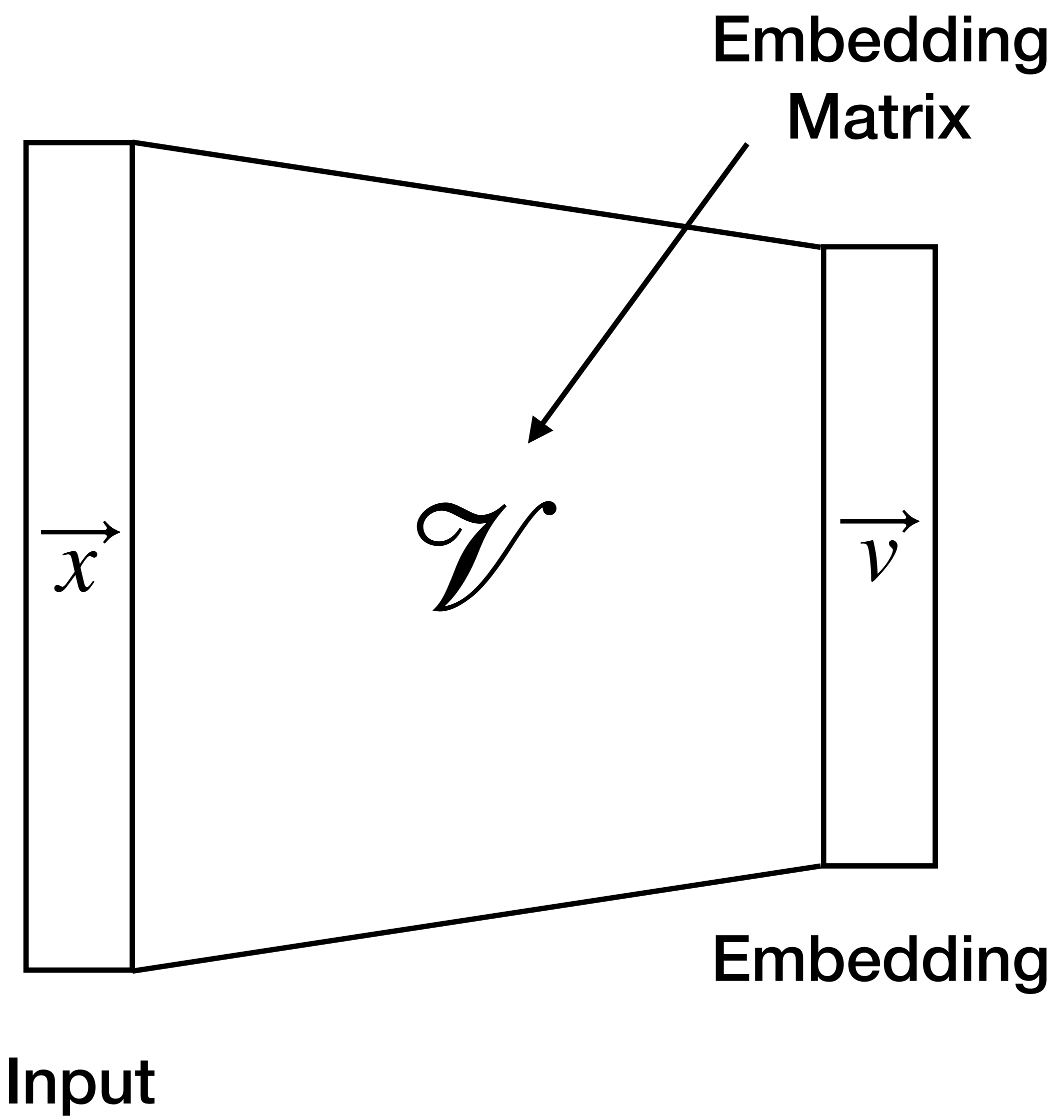


$$\vec{x}_t \in \mathbb{R}^{|V|}, \vec{v}_t \in \mathbb{R}^n$$

$$\mathscr{V} \in \mathbb{R}^{|V| \times n}$$

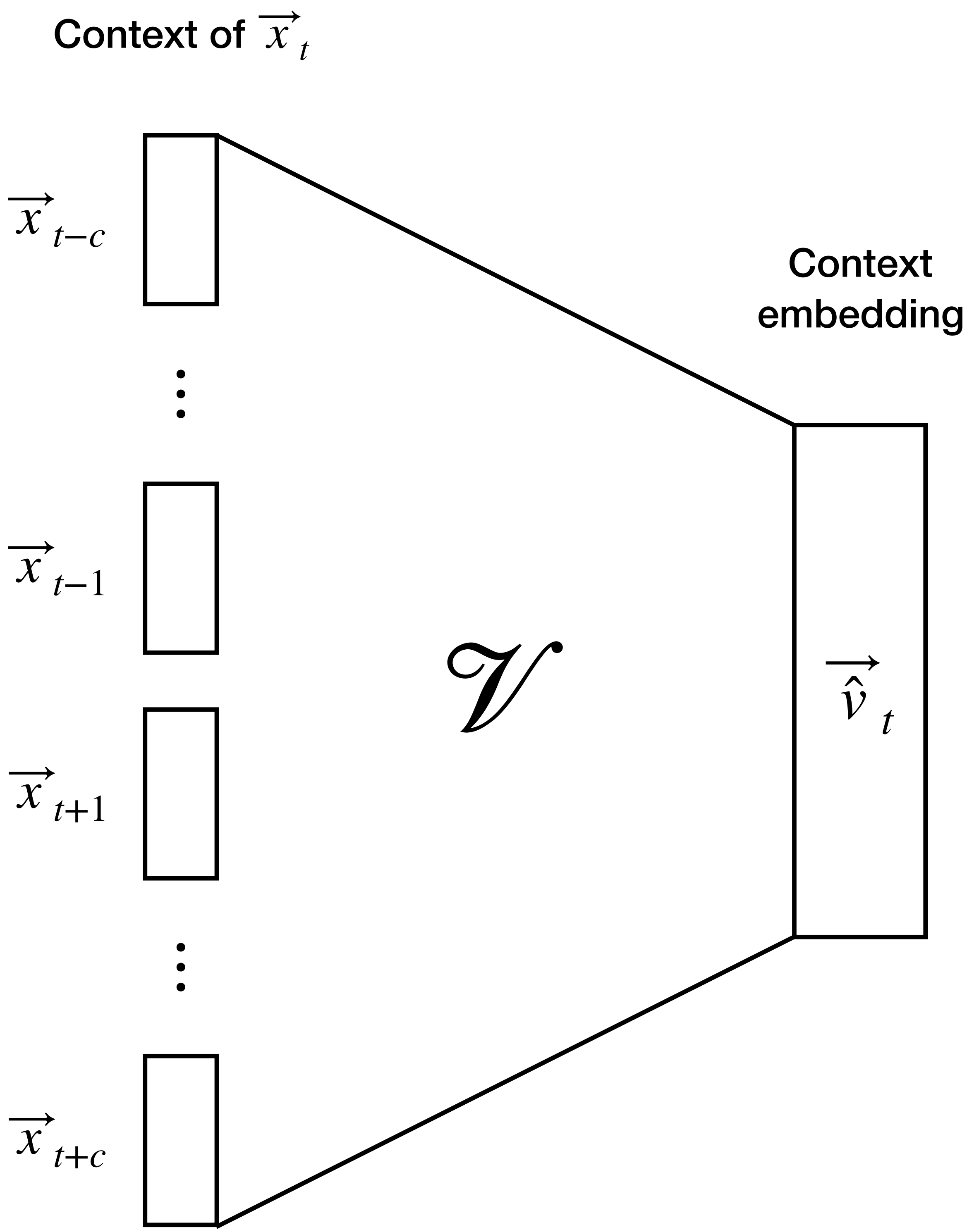
$$\vec{v}_t = \mathscr{V} \vec{x}_t$$

Embedding - LookUp



- Effectively we can substitute the matrix multiplication with a look up.
- Assume $\overrightarrow{x}$ represents the word w_i ; $\overrightarrow{x}$ is thus a one-hot vector with a 1 as its i-th component.
- The result of a matrix multiplication with $\mathscr{V}$ is thus simply the i-th row of $\mathscr{V}$.

CBOW - Embedding



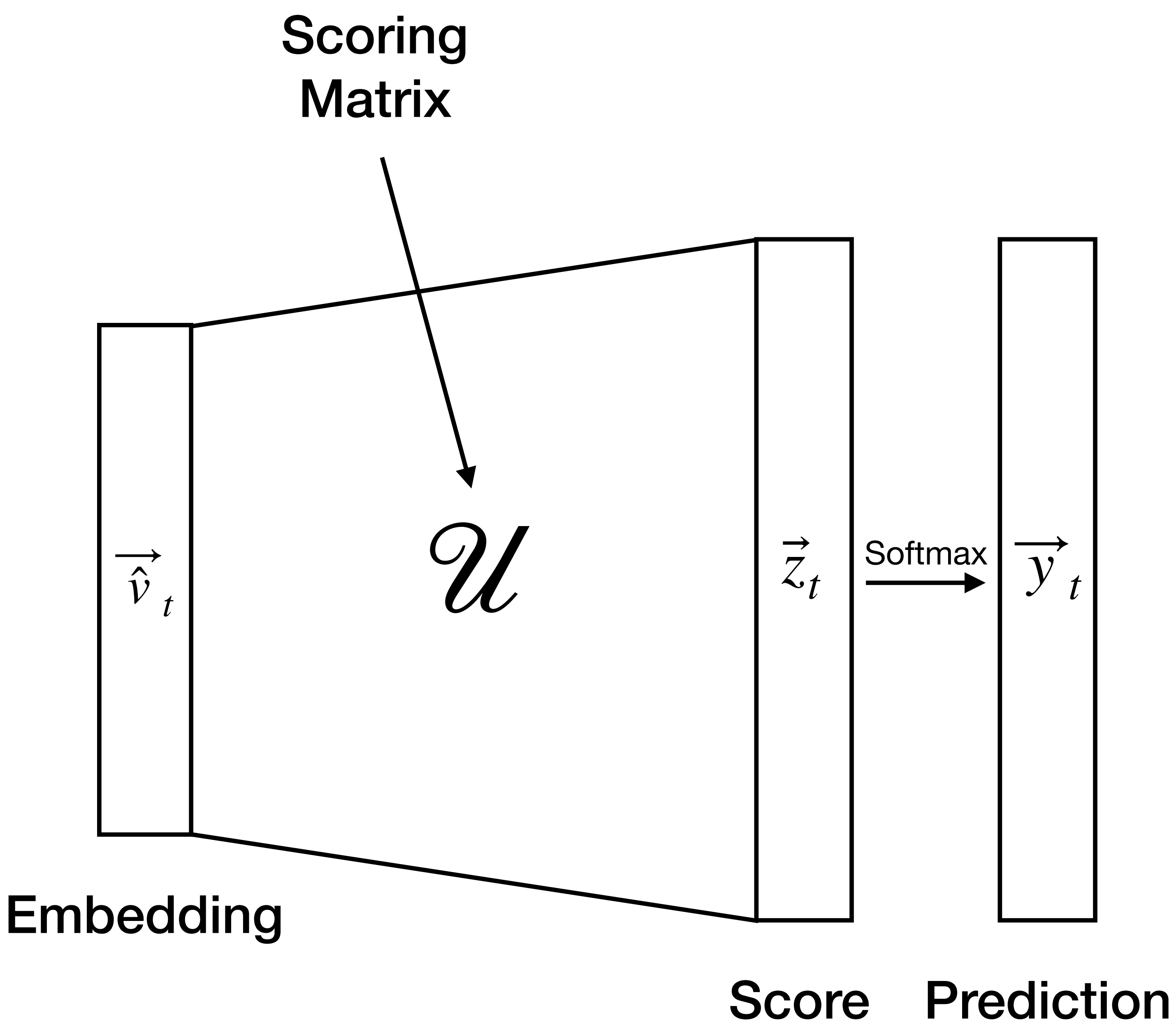
1. Calculate embeddings for all words in the context:

   $$\vec{v}_i = \mathscr{V}\vec{x}_i\,,$$
   $$i = t-c, \ldots, t-1, t+1, \ldots, t+c$$

2. Calculate context embedding by averaging the embeddings:

$$\vec{\hat{v}}_t = \frac{\vec{v}_{t-c} + \cdots + \vec{v}_{t-1} + \vec{v}_{t+1} + \cdots + \vec{v}_{t+c}}{2c}$$

CBOW - Predictions



$$\vec{\hat{v}}_t \in \mathbb{R}^n, \vec{z}_t \in \mathbb{R}^{|V|}$$

$$\mathcal{U} \in \mathbb{R}^{n \times |V|}, \vec{b} \in \mathbb{R}^{|V|}$$

3. Based on the context embedding we calculate a score for each word:

$$\vec{z}_t = \mathcal{U}\vec{\hat{v}}_t + \vec{b}$$

4. By applying softmax we turn the scores into likelihoods of how probable the different words are the actual missing word:

$$\vec{y}_t = softmax(\vec{z}_t)$$

CBOW

Given a text (in one-hot vectors): $\vec{x}_1, \ldots, \vec{x}_N$.

1. Chunk text into context training pairs given a context window size c :

$$(\vec{x}_{t-c}, \ldots, \vec{x}_{t-1}, \vec{x}_{t+1}, \ldots, \vec{x}_{t+c}), \ \vec{x}_t \quad \text{for} \quad t = c+1, \ldots, N-c$$

For all training pairs:

2. Calculate embeddings for all words in the context:

$$\vec{v}_i = \mathscr{V}\vec{x}_i \ ,$$
$$i = t-c, \ldots, t-1, t+1, \ldots, t+c$$

3. Calculate context embedding by averaging the embeddings:

$$\vec{\hat{v}}_t = \frac{\vec{v}_{t-c} + \cdots + \vec{v}_{t-1} + \vec{v}_{t+1} + \cdots + \vec{v}_{t+c}}{2c}$$

4. Based on the context embedding we calculate a score for each word: $\vec{z}_t = \mathscr{U}\vec{\hat{v}}_t + \vec{b}$

5. Apply softmax to turn score into probabilities: $\vec{y}_t = softmax(\vec{z}_t)$

6. Compute cross-entropy loss and minimize it using gradient descent.

Skip Gram

Skip Gram

- The <u>skip gram model</u> works as CBOW turned around.
- Now given a word we try to predict the words that occur in its context.
- This approach gives rise to more meaningful embeddings.

The cat climbed the tree.

Input Word $\xrightarrow{\text{predict}}$ **Context**

climbed — *The cat … the tree.*

Skip Gram

- In practice this means that we try to predict each word in the context from the word in the center.
- A context window of size c therefore gives us $2c$ training points for each word.

Input Word

climbed

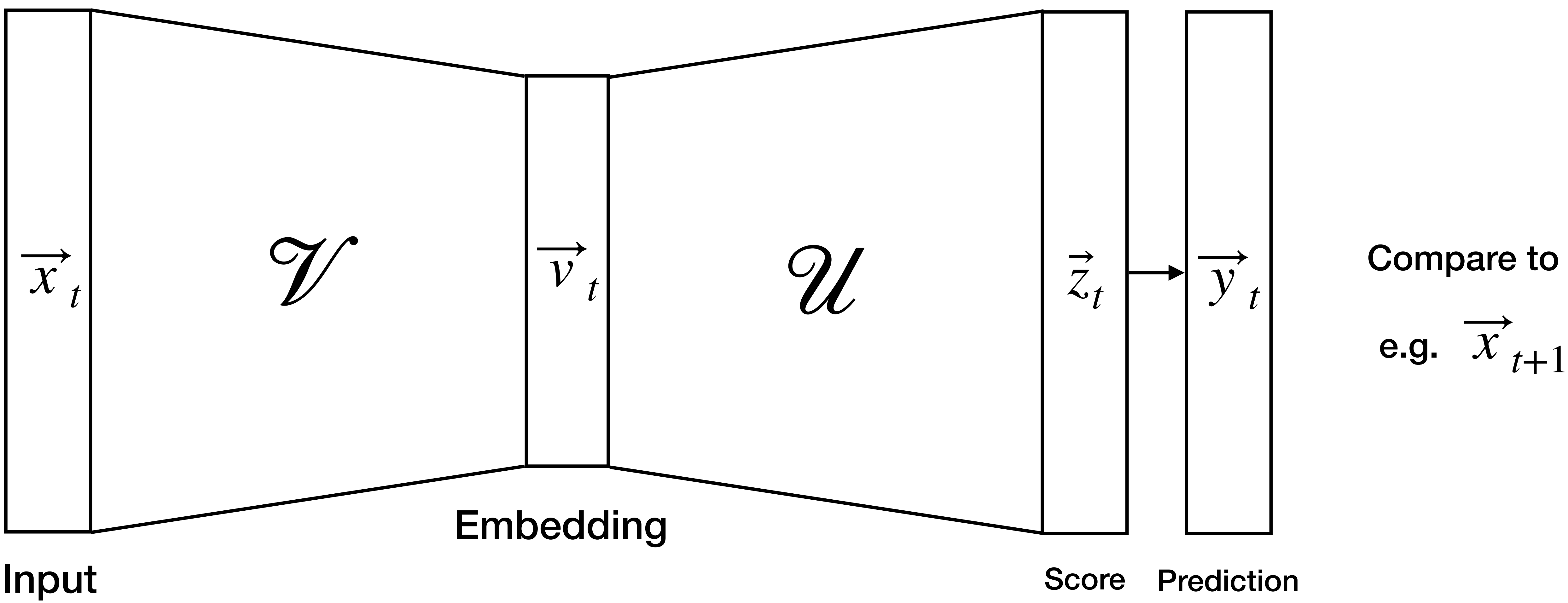
Context

The cat … the tree.

Input	**Target**
climbed	*the*
climbed	*cat*
climbed	*the*
climbed	*tree*

Skip Gram

- It uses the same basic architecture with the embedding and scoring matrix.



Skip Gram

Given a text (in one-hot vectors): $\overrightarrow{x}_1, \ldots, \overrightarrow{x}_N$.

1. Chunk text into context training pairs given a context window size c :

$$\overrightarrow{x}_t, \overrightarrow{x}_{t+i} \quad \text{for} \quad t = c+1, \ldots, N-c \quad \text{and} \quad i = -c, \ldots, c\ ; (i \neq 0)$$

For all training pairs:

2. Calculate embedding for input word: $\overrightarrow{v}_t = \mathscr{V}\overrightarrow{x}_t$
3. Based on the embedding we calculate a score for each word: $\vec{z}_t = \mathscr{U}\overrightarrow{x}_t + \overrightarrow{b}$
4. Apply softmax to turn score into probabilities: $\overrightarrow{y}_t = softmax(\vec{z}_t)$
5. Compute cross-entropy loss and minimize it using gradient descent.

Word2Vec

Word2Vec

- Skip gram was introduced in this paper by Mikolov et al. (2013).
- The model was trained on an internal google dataset (1B words, 692K unique words that occur more often than 5 times).
- The resulting embeddings can be downloaded: link.
- The following are other tricks used to train the model.

Subsampling

- Some words appear very often, e.g. '*the*'.
- These words might contain little/no information about the words around them.

Input	Target
climbed	*the*
climbed	*cat*
climbed	*the*
climbed	*tree*

- The more often a word occurs the more often it is shown to the model.
- To counteract this tendency we can discard a training sample based on how frequent it occurs:

Probability to discard: $$P(w_i) = max(0, 1 - \sqrt{\frac{t}{f(w_i)}}), \ (e.g.\ t = 10^{-5})$$

Negative Sampling

- Both CBOW and skip gram have a huge computational problem.

- They have to compute the softmax over the score vector (has length of size of vocabulary).

$$\overrightarrow{y}_t = softmax(\vec{z}_t) \qquad (\overrightarrow{y}_t)^{(i)} = \frac{exp((\vec{z}_t)^{(i)})}{\sum_j exp((\vec{z}_t)^{(j)})}$$

This sum is the main computational load of the training and therefore really slows it down.

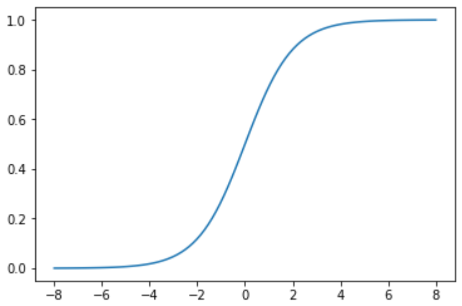
- <u>Negative sampling</u> (a variant of <u>NCE</u>) solves this problem.

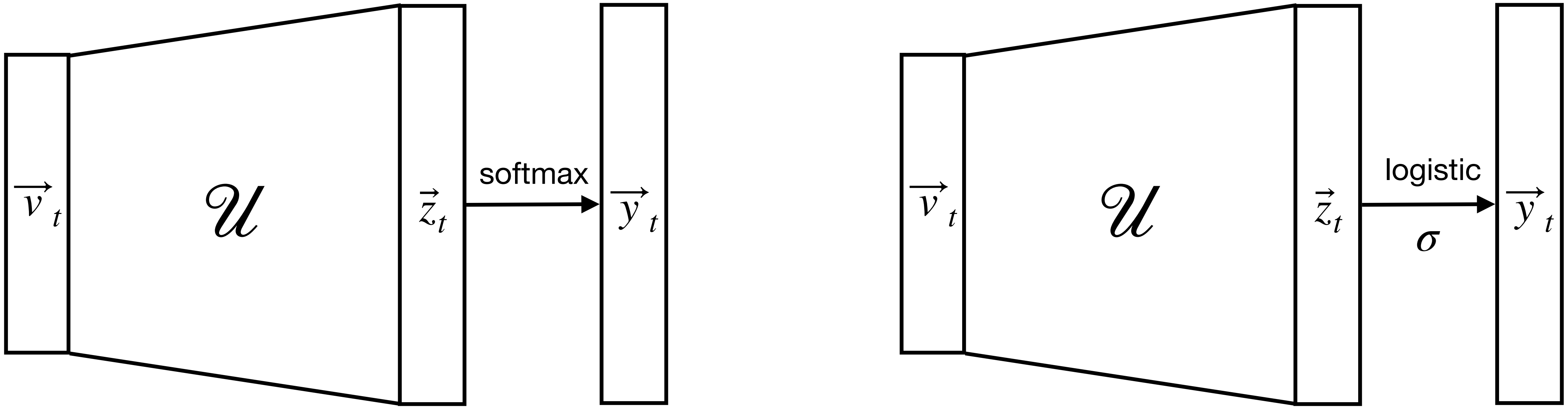
Negative Sampling

- To understand negative sampling we have to understand what applying softmax and cross entropy loss does.
- The softmax returns the probabilities of all words to be the target word.
- Minimizing the cross entropy loss
  - maximizes the probability for the actual target word,
  - minimizes the probability for all other words.
- **Idea: Don’t minimize probabilities of all other words but only of a few.**

Negative Sampling

- The problem is that by using softmax the probabilities are dependent on each other (they sum up to 1).
- So we have to find an alternative to softmax.
- Instead of applying the softmax we can apply the logistic activation function.





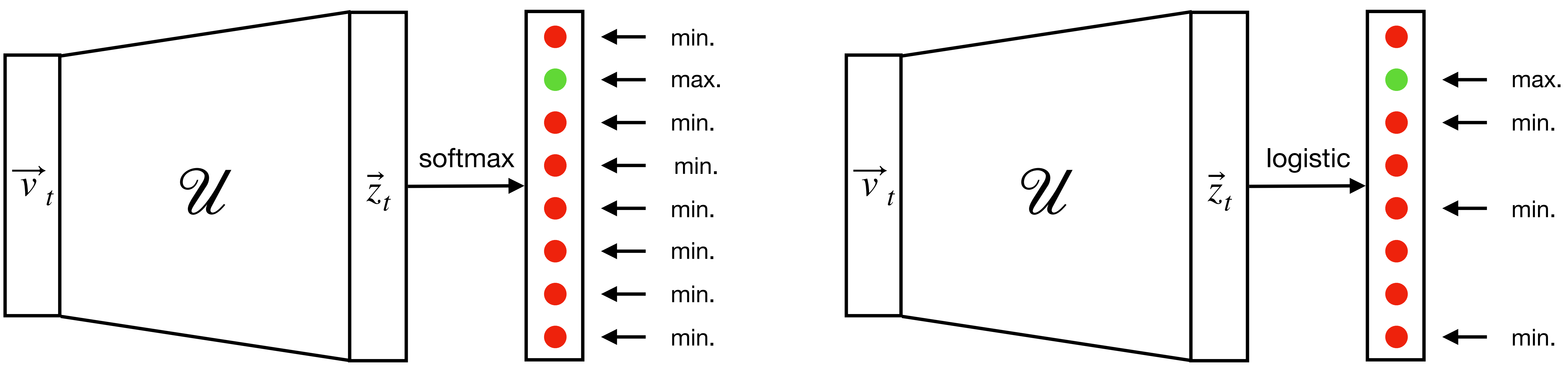
The values of $\vec{y}_t$ are between 0 and 1 and sum up to 1.

The values of $\vec{y}_t$ are just between 0 and 1.

Negative Sampling

- Now we can simply sample a few (e.g. $n = 15$) negative samples (= wrong words) and minimize their probability.

Previously: **Now:**



- How to sample the negative words? Paper empirically found that $\mathcal{U}^{3/4}$ (unigram distribution (frequency) to the power of $3/4$).

Word2Vec - Example Embeddings

Not available due to copyright issues.

Bonus

Bonus

- Seq2Seq: [Blog](), [Paper]()
- Attention: [Blog]()
- Transformers: [Blog](), [Paper]()

Any questions?

See you next week!