50.039 Theory and Practice of Deep Learning

W9-S2 The Embedding Problem

Matthieu De Mari, Berrak Sisman

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

About this week (Week 9)

1. Why are **embeddings** an essential component of Neural Networks (NNs)?
2. Why are **good embeddings** difficult to produce?
3. What are the **conventional approaches to embeddings in NLP?** What can we **learn from these approaches?**
4. What are the **typical issues with embeddings** and how do we address them?
5. **State-of-the-art** of current embedding problems, and **open questions** in research.

About this week (Week 9)

6. How do we evaluate the **quality/performance** of an embedding?
7. Can embeddings be **biased**?
8. Can we help the neural networks **identify the important parts of the context** to focus on?
9. What is **attention** in Neural Networks? What are **transformers** in Neural Networks?
10. What are the typical **uses for attention** these days?
11. What are the **limits of attention** and the **current research directions** on this topic?

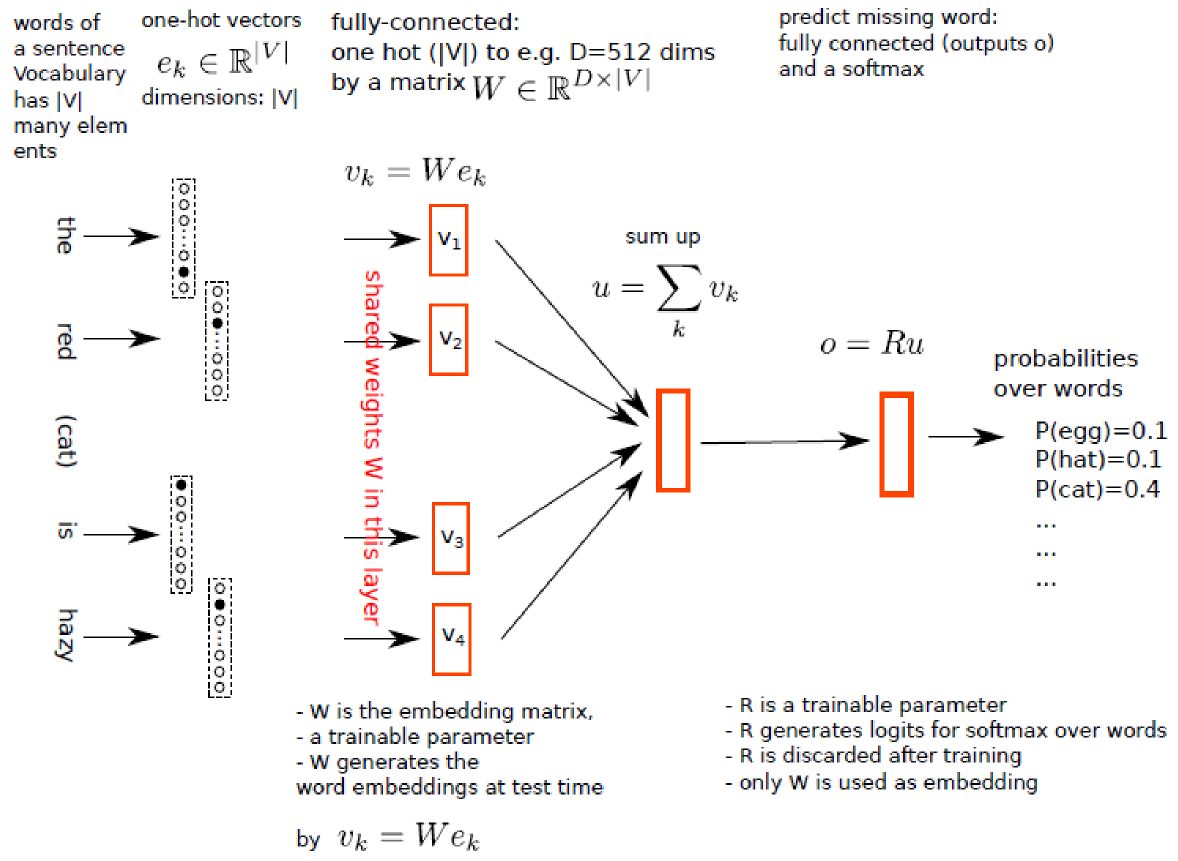
Continuous Bag of Words (CBoW)

Definition (CBoW):

CBoW (introduced in [Mikolov2013]) is a first feature representation model, which can be used for word embedding.

Using a large text corpus for training, it attempts to learn how to predict the word in the middle (with index k) of a sequence of $2k + 1$ words.

Here, k is called the **span** of the language model.

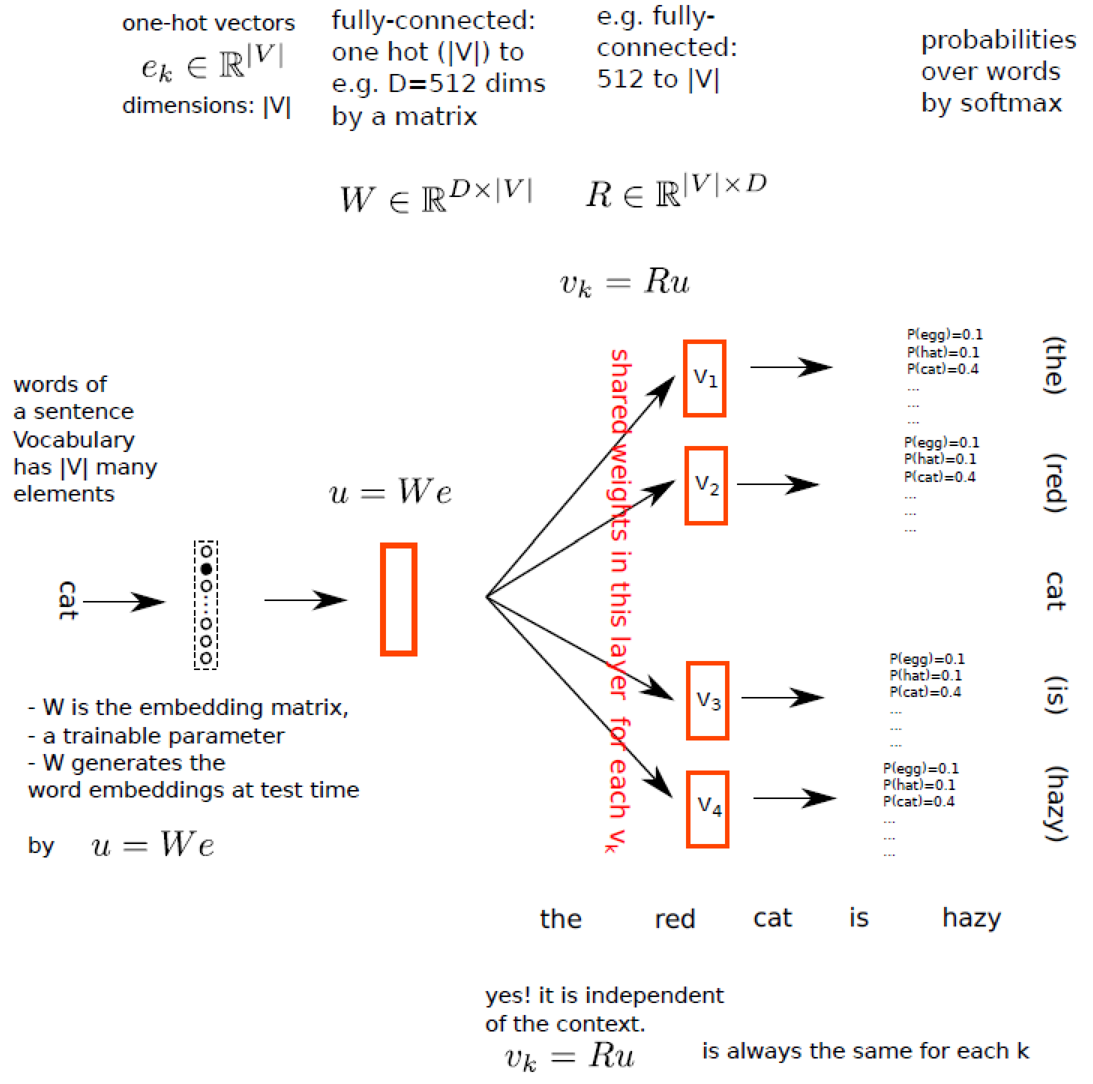

words of a sentence Vocabulary has |V| many elem ents
one-hot vectors $e_k \in \mathbb{R}^{|V|}$ dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix $W \in \mathbb{R}^{D \times |V|}$
predict missing word: fully connected (outputs o) and a softmax
the
red
(cat)
is
hazy
$v_k = We_k$
shared weights W in this layer
v_1
v_2
v_3
v_4
sum up
$u = \sum_k v_k$
$o = Ru$
probabilities over words
P(egg)=0.1
P(hat)=0.1
P(cat)=0.4
...
...
...
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $v_k = We_k$
- R is a trainable parameter
- R generates logits for softmax over words
- R is discarded after training
- only W is used as embedding

SkipGram (SG)

Definition (SkipGram):

Another interesting learning task one could use to train an embedding, would consist of using a single work and attempt to predict some possible surrounding words. In other words, take a middle word k, and try to predict the $2k$ surrounding words.

This is what **SkipGram** attempts to reproduce.



one-hot vectors $e_k \in \mathbb{R}^{|V|}$ dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
e.g. fully-connected: 512 to |V|
probabilities over words by softmax
$W \in \mathbb{R}^{D \times |V|}$
$R \in \mathbb{R}^{|V| \times D}$
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
$u = We$
cat
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the
red
cat
is
hazy
yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

SkipGram (SG)

- Consider the text: “I have a dream that one day this nation will rise up and live out the true meaning of its creed: We hold these truths to be self-evident, that all men are created equal. I have a dream that one day on the red hills of Georgia, the sons of former slaves and the sons of former slave owners will be able to sit down together at the table of brotherhood.”

1. We will use a sliding window with e.g. size k=2 to generate pairs of (x, y) values to train our SkipGram on.

$$y_1 = (I, have, dream, that),$$
$$x_1 = a$$

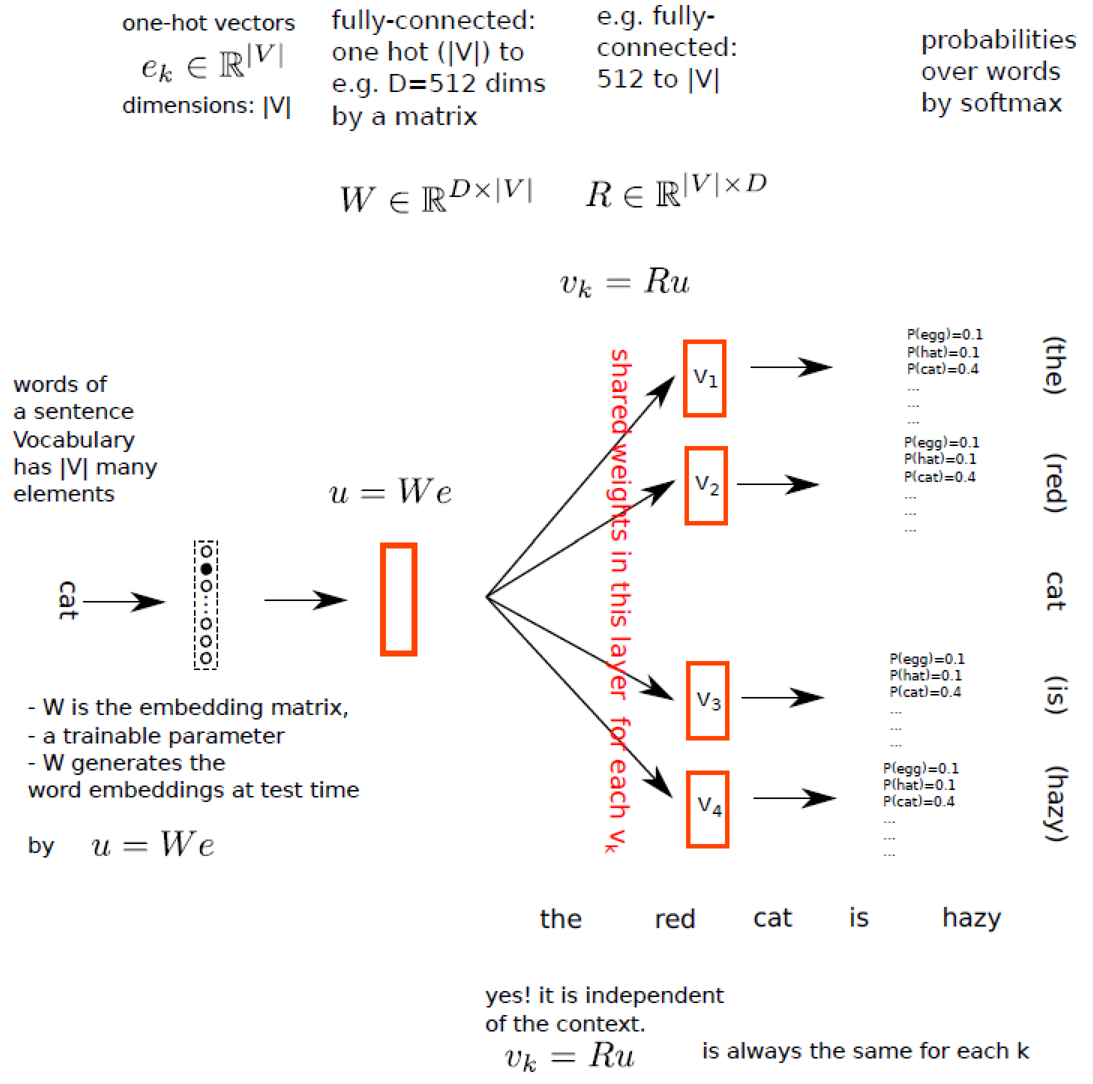
$$y_2 = (have, a, that, one),$$
$$x_2 = dream$$

$$y_3 = (the, sons, former, slave),$$
$$x_3 = of$$

SkipGram (SG)

2. Then, build a simple NN, which takes a middle word, as one-hot embedding $e \in \mathbb{R}^{|V|}$.

   Add one fully-connected layer with matrix $W \in \mathbb{R}^{D \times |V|}$. Here, D denotes the size of the new word embedding and is often chosen such that $D \ll |V|$.

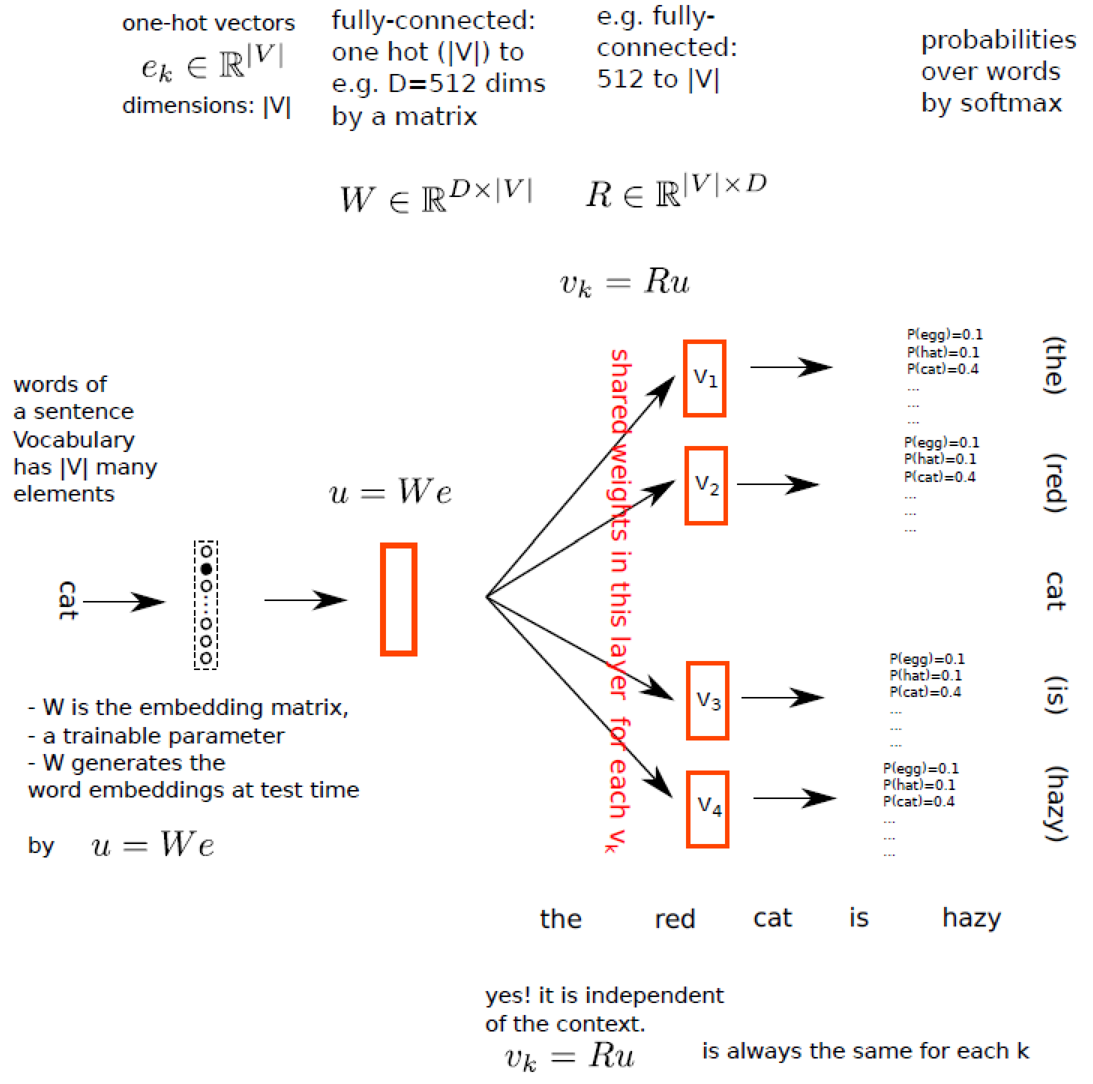

one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
e.g. fully-connected: 512 to |V|
probabilities over words by softmax
$W \in \mathbb{R}^{D \times |V|}$
$R \in \mathbb{R}^{|V| \times D}$
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
$u = We$
cat
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1
P(hat)=0.1
P(cat)=0.4
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the
red
cat
is
hazy
yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

SkipGram (SG)

2. Then, build a simple NN, which takes a middle word, as one-hot embedding $e \in \mathbb{R}^{|V|}$.

   Add one fully-connected layer with matrix W $\in \mathbb{R}^{D \times |V|}$. Here, D denotes the size of the new word embedding and is often chosen such that $D \ll |V|$.

   After training, $\mathrm{u} = We$ will be used as the new word embedding to replace any e!


one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
e.g. fully-connected: 512 to |V|
probabilities over words by softmax
$W \in \mathbb{R}^{D \times |V|}$
$R \in \mathbb{R}^{|V| \times D}$
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
$u = We$
cat
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1
P(hat)=0.1
P(cat)=0.4
...
(the)
(red)
cat
(is)
(hazy)
the
red
cat
is
hazy

- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time

by $u = We$

yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

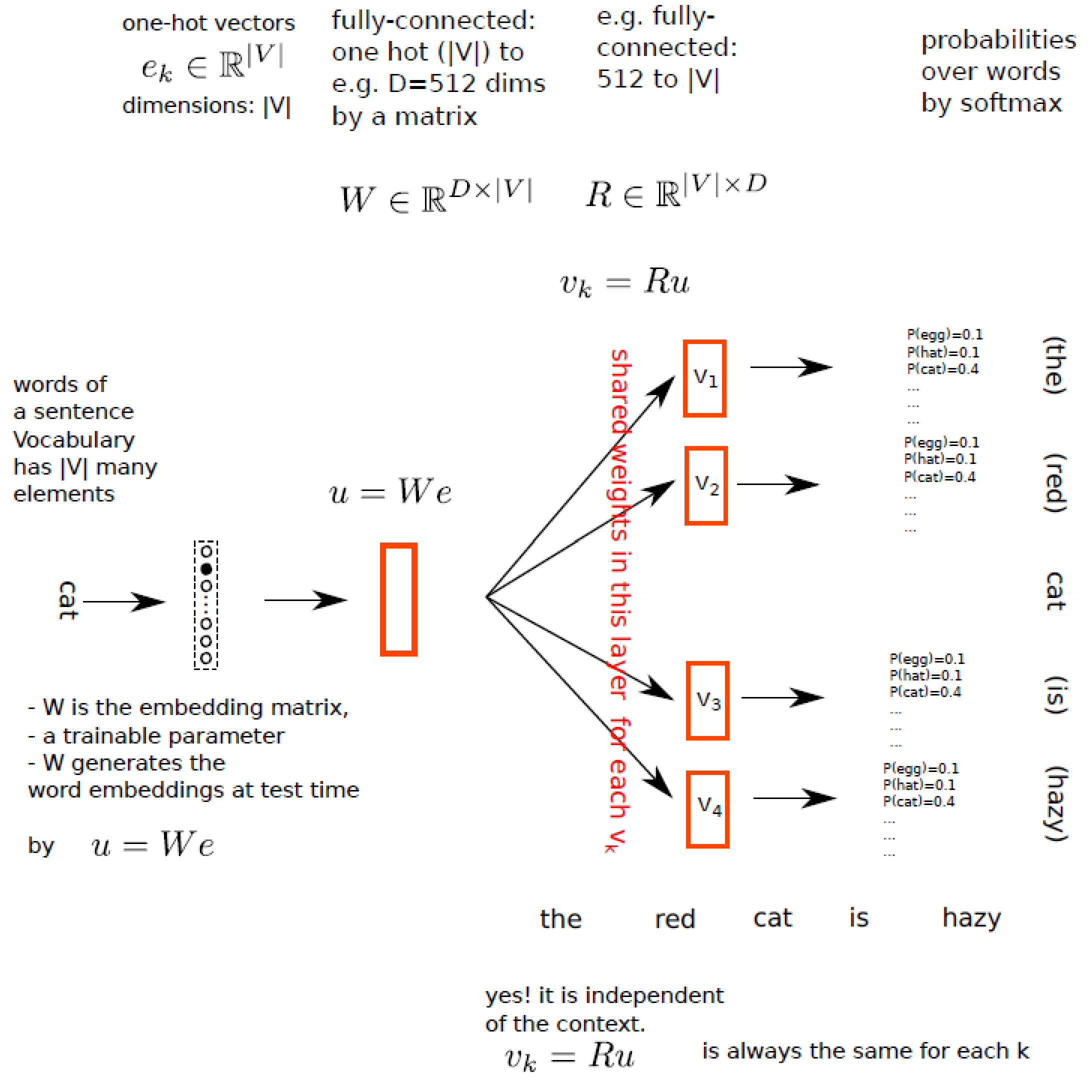
SkipGram (SG)

2. The final layer is a Linear layer (or an Embedding Layer), trainable and produces an outputs v_k as:

$$v_k = Ru$$

With $R \in \mathbb{R}^{|V| \times D}$ and all $v_k \in \mathbb{R}^{|V|}$.

The outputs v_k then give probabilities over which word should be predicted.



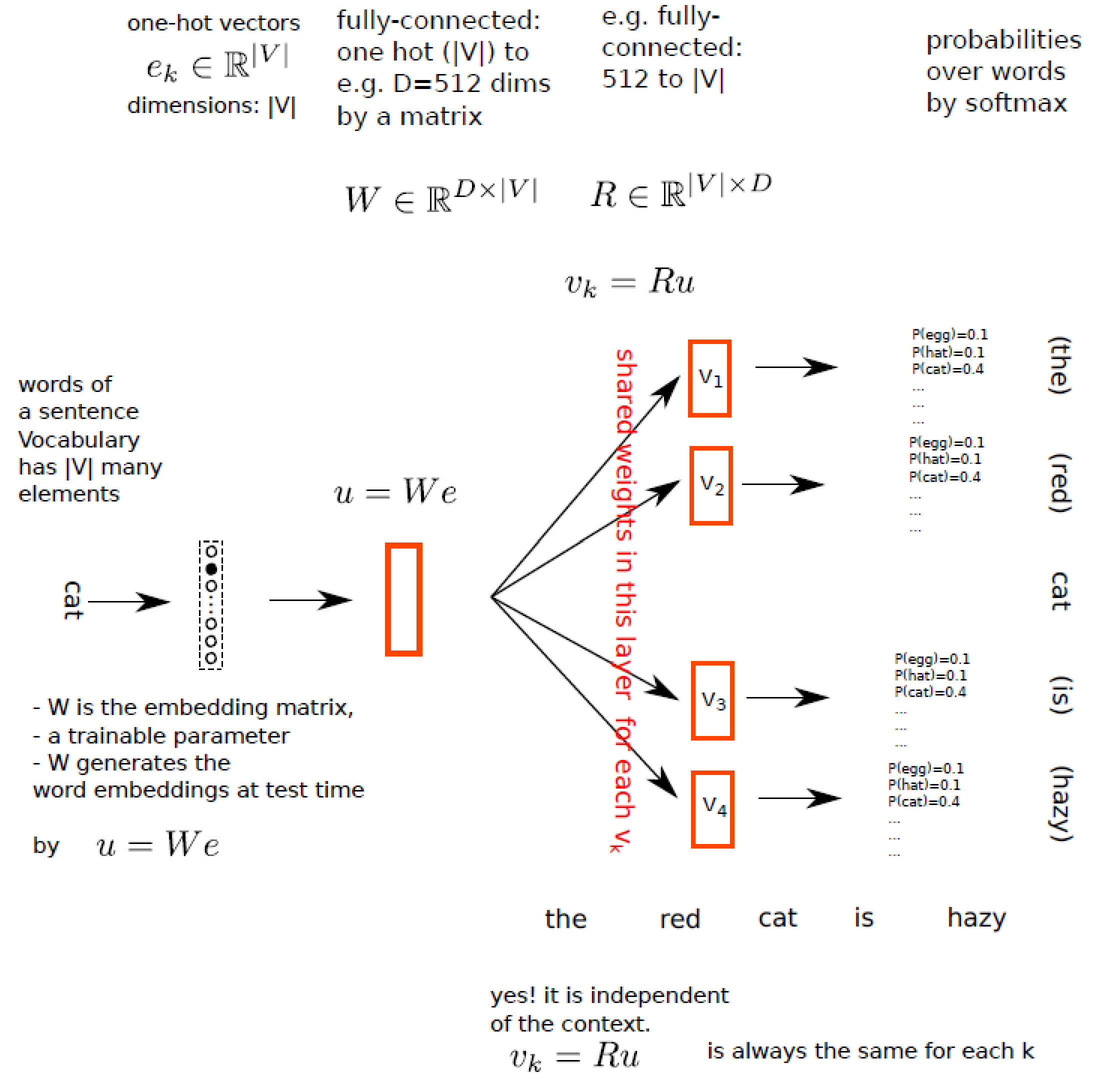
one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
$W \in \mathbb{R}^{D \times |V|}$
e.g. fully-connected: 512 to |V|
$R \in \mathbb{R}^{|V| \times D}$
probabilities over words by softmax
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
cat
$u = We$
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the red cat is hazy
yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

SkipGram (SG)

3. During the training, we browse through all the pairs (x, y) we generated on Step 1.

   Just like before, it is advisable to use some sort of a negative logarithm as the loss function.

   And sum over all the pairs (x, y).



one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
e.g. fully-connected: 512 to |V|
probabilities over words by softmax
$W \in \mathbb{R}^{D \times |V|}$
$R \in \mathbb{R}^{|V| \times D}$
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
$u = We$
cat
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1
P(hat)=0.1
P(cat)=0.4
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the
red
cat
is
hazy
yes! it is independent of the context.
$v_k = Ru$
is always the same for each k

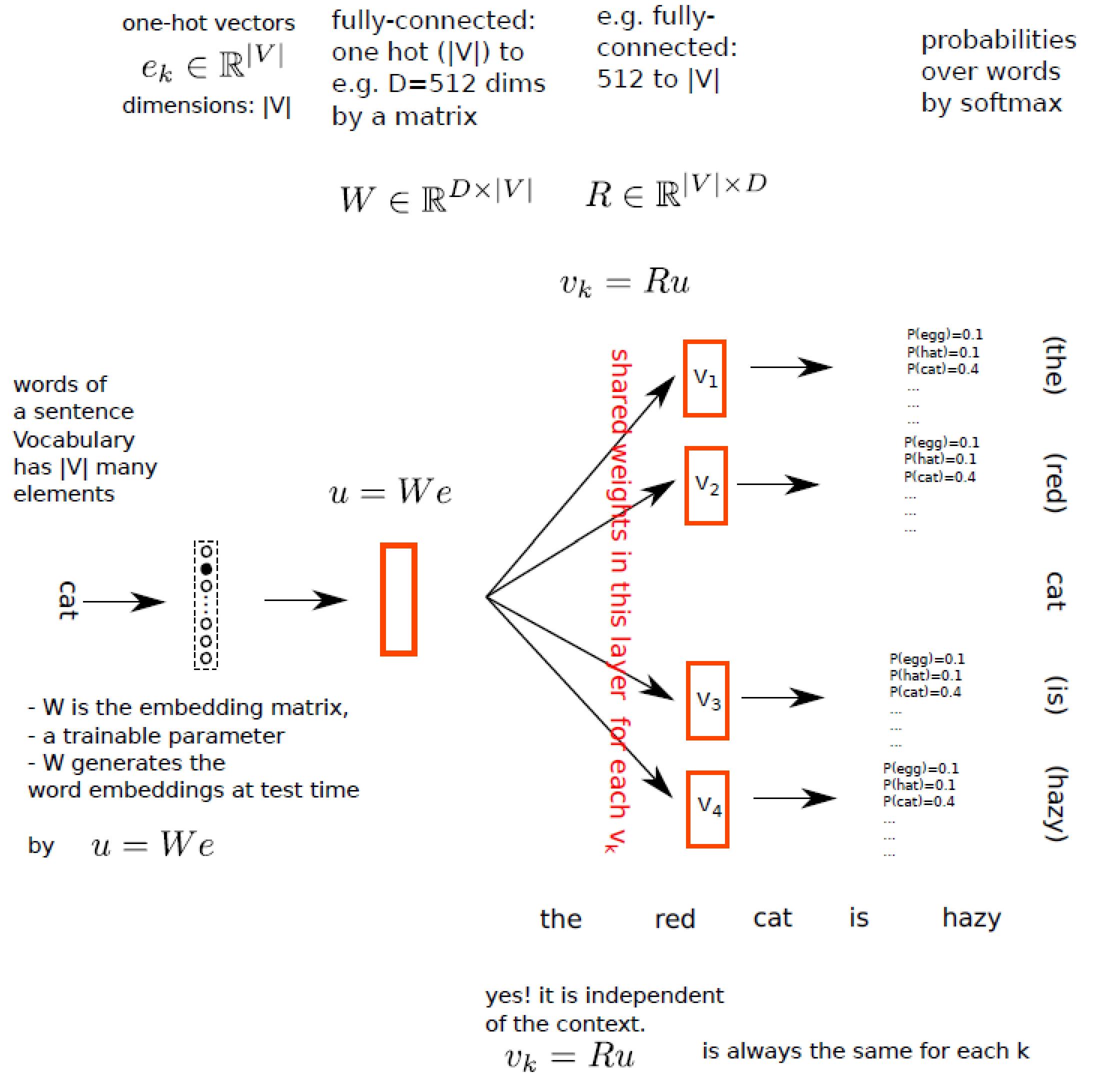
SkipGram (SG)

3. A good loss is often computed by summing all L_t terms, defined as

$$L_t = \sum_{n=-2k}^{n=2k} -\log(P(o_{t+n} = w_{t+n} | w_t))$$

$$P(\dots) = \frac{\exp\left(u.R^T(w_{t+n},:)\right)}{\sum_{w\in V} \exp\left(u.R^T(w,:)\right)}$$

$$u = We_{w_t}$$

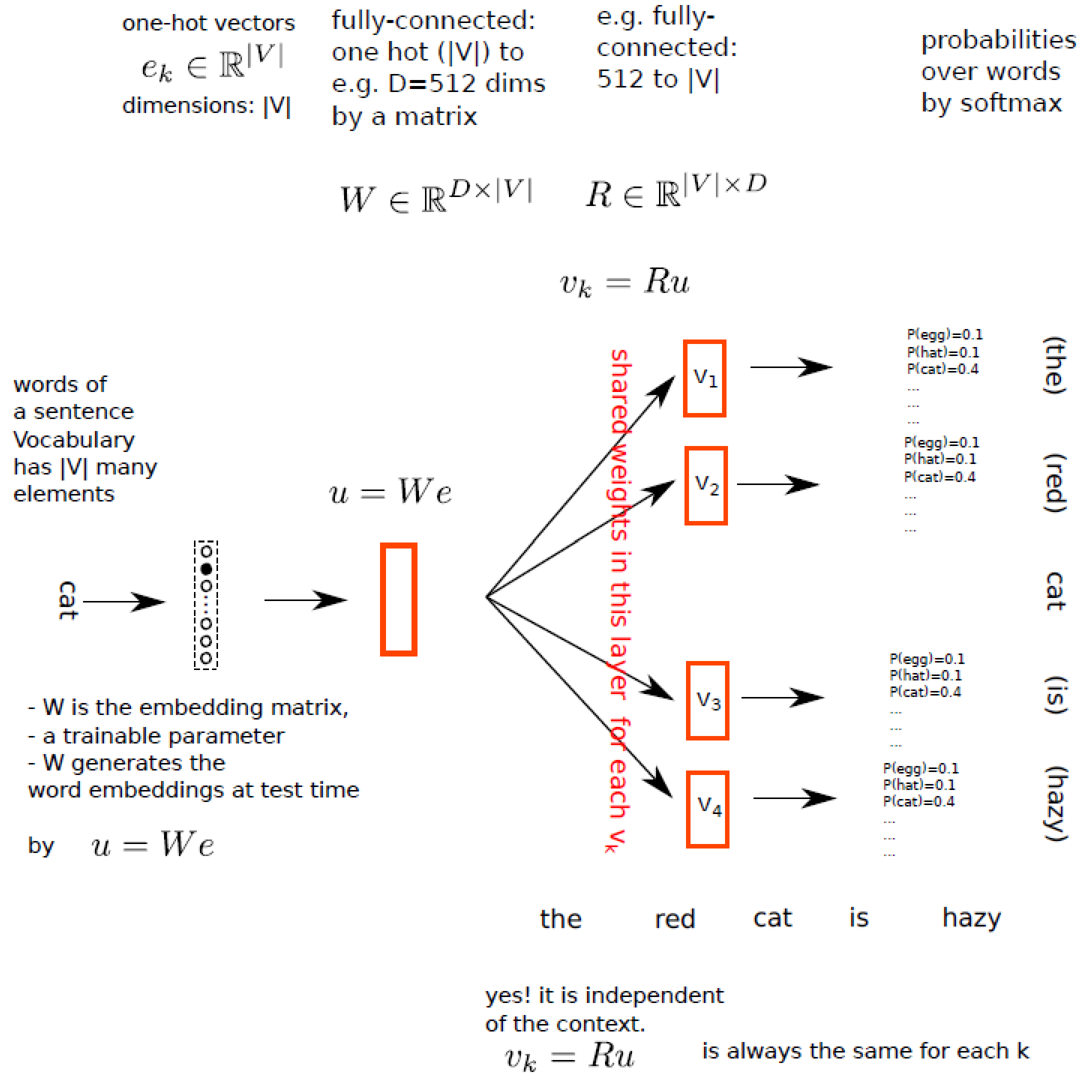


one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
$W \in \mathbb{R}^{D\times|V|}$
e.g. fully-connected: 512 to |V|
$R \in \mathbb{R}^{|V|\times D}$
probabilities over words by softmax
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
cat
$u = We$
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1
P(hat)=0.1
P(cat)=0.4
...
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the
red
cat
is
hazy
yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

SkipGram (SG)

Important Notes #1

- $\boldsymbol{v_k = Ru}$ **can be the same output for each $\boldsymbol{k}$.** This is okay because we do not really want to generate a valid output sentence around word w_t!
- **All we want is to train a good embedding $\boldsymbol{W}$. So the order of output words does not matter.** We only care that the probability is high for the right $2k$ output words which are around w_t.

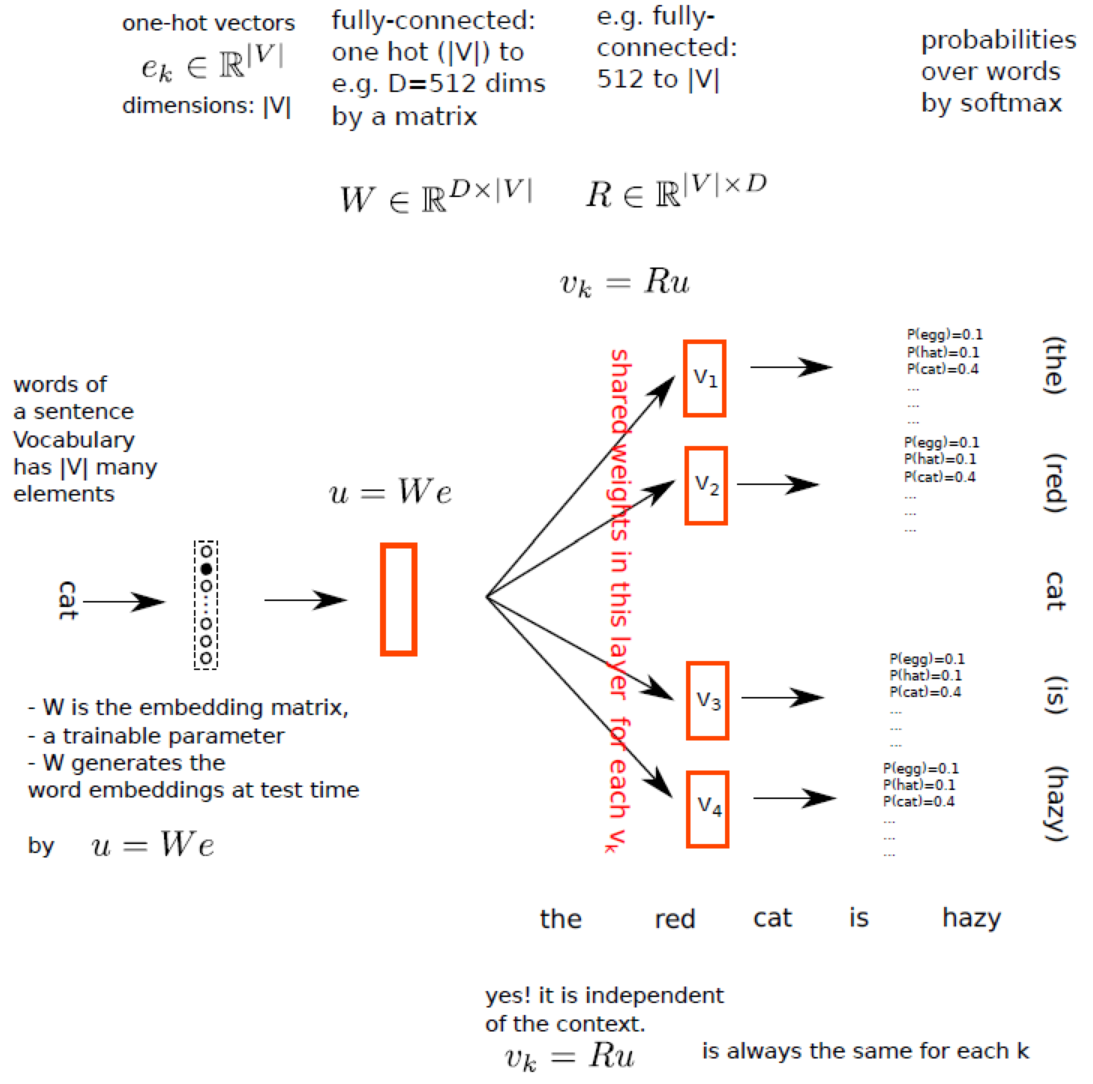


one-hot vectors
$e_k \in \mathbb{R}^{|V|}$
dimensions: |V|
fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix
e.g. fully-connected: 512 to |V|
probabilities over words by softmax
$W \in \mathbb{R}^{D\times|V|}$
$R \in \mathbb{R}^{|V|\times D}$
$v_k = Ru$
words of a sentence Vocabulary has |V| many elements
$u = We$
cat
shared weights in this layer for each v_k
v_1
v_2
v_3
v_4
P(egg)=0.1 P(hat)=0.1 P(cat)=0.4 ... ...
(the)
(red)
cat
(is)
(hazy)
- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time
by $u = We$
the
red
cat
is
hazy
yes! it is independent of the context.
$v_k = Ru$ is always the same for each k

SkipGram (SG)

Important Notes #2

- This also means that training a SkipGram is far more difficult than training a CBoW! It is however producing better encodings, as it produces embedding that are able to describe any word by (somewhat) predicting the surrounding words around it.


one-hot vectors $e_k \in \mathbb{R}^{|V|}$ dimensions: |V|

fully-connected: one hot (|V|) to e.g. D=512 dims by a matrix $W \in \mathbb{R}^{D\times|V|}$

e.g. fully-connected: 512 to |V| $R \in \mathbb{R}^{|V|\times D}$

$v_k = Ru$

probabilities over words by softmax

words of a sentence Vocabulary has |V| many elements

$u = We$

- W is the embedding matrix,
- a trainable parameter
- W generates the word embeddings at test time by $u = We$

shared weights in this layer for each v_k

yes! it is independent of the context. $v_k = Ru$ is always the same for each k

Homework this week will require to train a SkipGram (correctly!)

HW9!

Word2Vec and Glove

- The two approaches (CBoW and SkipGram) are commonly referred to as **Word2Vec** approaches.
- Another option, often referred to in literature is **GloVe** (Global Vectors for Word Representation).
- This is a “simple” count-based embedding [GloVe2014] and was considered a good challenger to Word2Vec.

Choose a Pre-Trained Embedding If

- Your dataset is composed of more “general” language and you don’t have that big of a dataset, to begin with.
- Since these embeddings have been trained on a lot of words from different sources, pre-trained models might do well if your data is generalized as well.
- Also, you will save on time and computing resources with pre-trained embeddings.

Word2Vec and Glove

Choose to Train Your Own Embedding If

- Your data (and project) is based on a niche industry, such as medicine, finance or any other non-generic and highly specific domains.
- In such cases, a general word embedding representation might not work out for you and some words might be altogether missing from the pre-trained embeddings.

Word2Vec and Glove

Choose to Train Your Own Embedding If

- Your data (and project) is based on a niche industry, such as medicine, finance or any other non-generic and highly specific domains.
- In such cases, a general word embedding representation might not work out for you and some words might be altogether missing from the pre-trained embeddings.

- On the downside, a lot of data is needed to ensure that the word embeddings being learned do a proper job of representing the various words and the semantic relationships between them, unique to your domain.
- Also, it takes a lot of computing resources to go through your corpus and build word embeddings.

Word2Vec and Glove

- Both Word2Vec and GloVe approaches are commonly referred to as **unsupervised (or semi-supervised) approaches to embedding,** both relying on the **distributional hypothesis** we mentioned on the previous lecture.
- Unsupervised representation learning of sentences had been the norm for quite some time.
- **More advanced versions of unsupervised approaches exist!**

- For instance: **FastText** and **ELMo**.

FastText

- **FastText** was developed by the team of Mikolov (him again!), triggering the explosion of research on **universal word embeddings** [Mikolov2018].

Universal embeddings?

- A huge trend in DL/NLP is the quest for **Universal Embeddings.**

Definition (Universal Embedding):

Universal Embeddings are **embeddings that are pre-trained** on a large corpus and can be plugged in a variety of downstream task models to automatically improve their performance, by incorporating some general word/sentence representations learned on larger datasets.

In a sense, it is the ultimate form of **transfer learning** for language embeddings, the holy grail, i.e. a unique **universal embedding** that everyone can use on any task related to language.

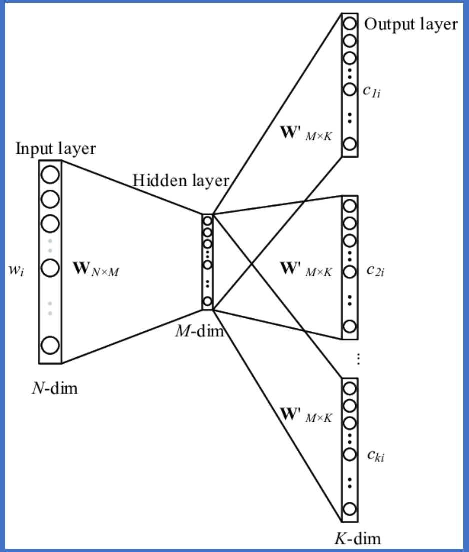
Open question: Are universal embeddings even possible or good? Should not they be task-specific?

FastText

- **FastText** was developed by the team of Mikolov (him again!), triggering the explosion of research on **universal word embeddings** [Mikolov2018].
- The main improvement of FastText over the original word2vec vectors is the inclusion of character **n-grams**, which allows computing word representations for words that did not appear in the training data (something called **“out-of-vocabulary”** words).

Input: Concatenation of word + its n-grams.
(e.g. **“eating”** + “ea” + “eat” + “ati” + “tin” + “ing”)

Model: Skip-Gram Architecture
(Predict context words based on single word + its n-grams)



Output: 2k words of context.

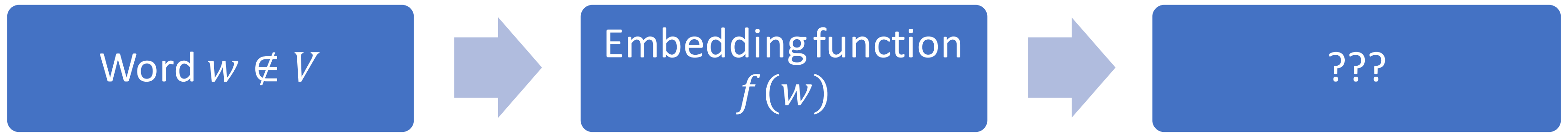
“Out of vocabulary” words

Definition (out-of-vocabulary):

Embeddings are often subject to the **out-of-vocabulary** issue.

This is a simple concept: what should the embedding function be doing if it is asked to encode a word that is not contained in the original dictionary V?

(Think new words, typos, etc.)



“Out of vocabulary” words

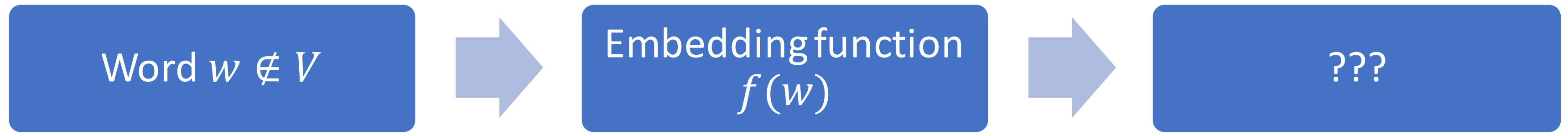
Definition (out-of-vocabulary):

Embeddings are often subject to the **out-of-vocabulary** issue.

This is a simple concept: what should the embedding function be doing if it is asked to encode a word that is not contained in the original dictionary V?

(Think new words, typos, etc.)

In the case of Word2Vec (CBoW and SkipGram algorithms), this would be very invalidating, as we would have no way to one-hot encode the missing word.



“Out of vocabulary” words

Definition (out-of-vocabulary):

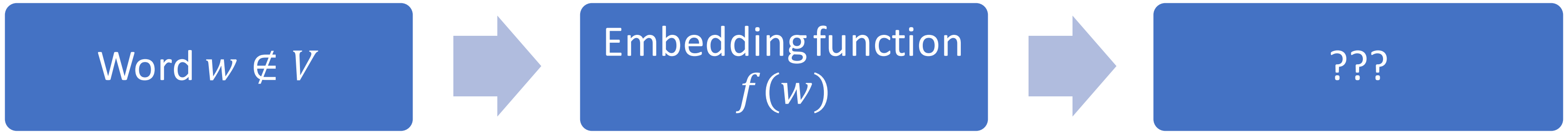
Embeddings are often subject to the **out-of-vocabulary** issue.

This is a simple concept: what should the embedding function be doing if it is asked to encode a word that is not contained in the original dictionary V?

(Think new words, typos, etc.)

In the case of Word2Vec (CBoW and SkipGram algorithms), this would be very invalidating, as we would have no way to one-hot encode the missing word.

A good embedding function should then be able to, at least partially, operate on out-of-vocabulary words.

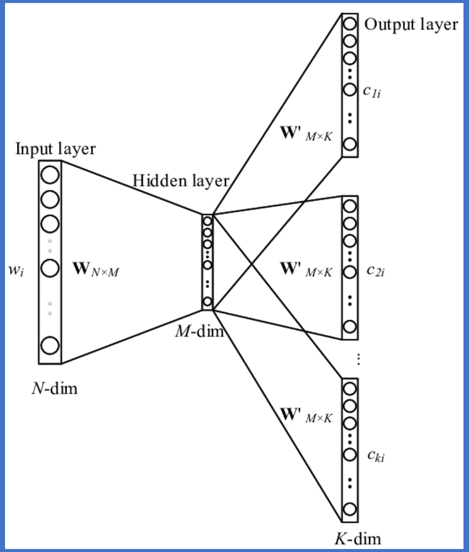


Core idea behind FastText (out-of-class)

- FastText achieves really good performance for word representations, specially in the case of rare words by making use of character level information.
- Each word is represented as a bag of characters **n-grams** in addition to the word itself.
- For example, for the word "computer", with **n = 3**, the FastText representations for the character **n-grams** is <co, com, omp, mpu, put, ute, ter, er>.

Input: Concatenation of word + its n-grams.
(e.g. **"eating"** + "ea" + "eat" + "ati" + "tin" + "ing")

Model: Skip-Gram Architecture
(Predict context words based on single word + its n-grams)



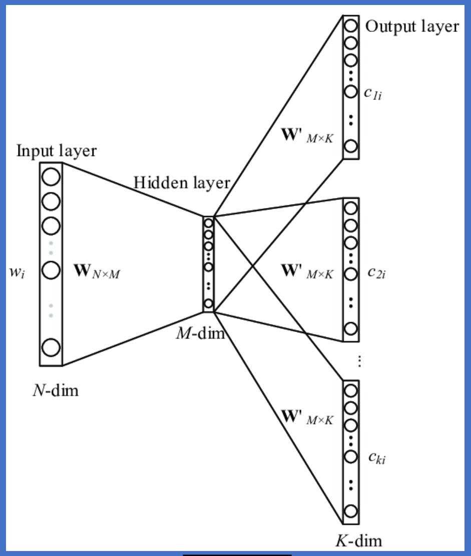
Output: 2k words of context.

Core idea behind FastText (out-of-class)

- Then, FastText uses the same approach and logic as in SkipGram, to try and predict some context words, by using not just the word, but also its n-grams.
- This typically allows to identify **etymology** in words, and **cover for potential typos**!
- Overall reinforces the performance, allows to **address** the **out of vocabulary issue**!

Input: Concatenation of word + its n-grams.
(e.g. **“eating”** + “ea” + “eat” + “ati” + “tin” + “ing”)

Model: Skip-Gram Architecture
(Predict context words based on single word + its n-grams)



Output: 2k words of context.

FastText

- **FastText** was developed by the team of Mikolov (him again!), triggering the explosion of research on **universal word embeddings** [Mikolov2018].
- The main improvement of FastText over the original word2vec vectors is the inclusion of character **n-grams**, which allows computing word representations for words that did not appear in the training data (something called **"out-of-vocabulary"** words).
- FastText vectors are super-fast to train and are available in 157 languages trained on Wikipedia and Crawl. They are a great baseline, not sacrificing too much performance.
- Ready-to-use FastText embeddings from: https://github.com/facebookresearch/fastText
- Official (and good) tutorial: https://fasttext.cc/docs/en/unsupervised-tutorial.html

ELMo

The **Deep Contextualized Word Representations (a.k.a. Embeddings for Language Model - ELMo)** have recently improved the state of the art in word embeddings by a noticeable amount.

They were developed by the Allen institute for AI and were presented at NAACL 2018 in early June [Peters2018].

- Ready-to-use ELMo embeddings https://allennlp.org/elmo
- Official and good tutorial: https://guide.allennlp.org/

ELMo

Key takeaways

1. Input of words, into a character-level network (inputs will be a list of characters!)
2. To get an embedding of the word, input the whole sentence for context, then take only the vector which corresponds to that word.
3. On top of character-level network, use multiple layers of RNNs on word-level.
4. Train the whole network for the task of predicting the next word in sentence, almost as in CBoW.
5. After training, extract embedding layer for feature representation

Core idea behind ELMo (out-of-class)

Training setup is predicting the next word in a sentence.

That is a classification task of some sort (again!), hence the loss should then be a negative log probability as before.

$$L_t = -\log(o_t = w_t | w_{-t})$$

With

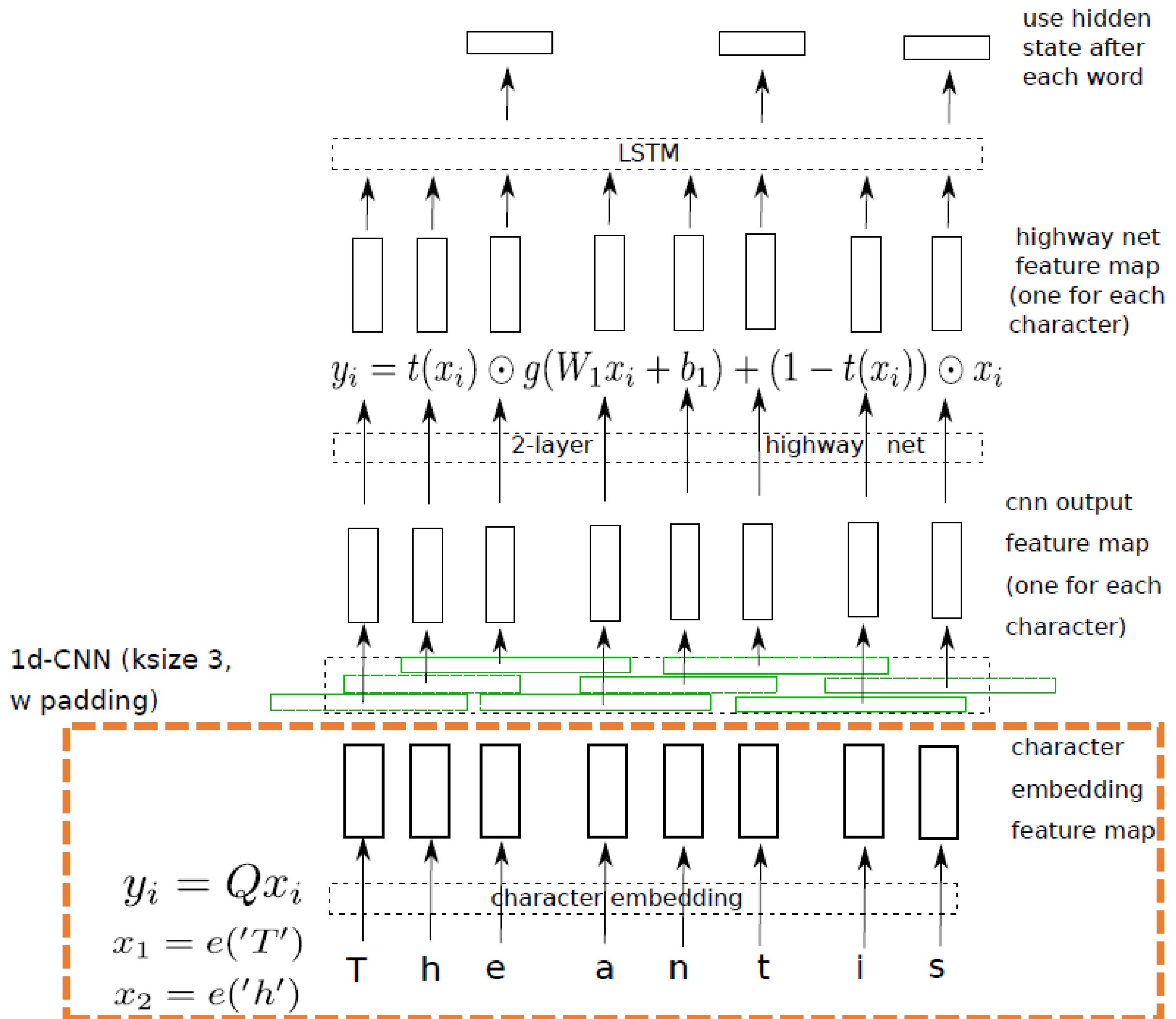
$$w_{-t=} (w_{t-1}, w_{t-2} \dots w_1)$$

And

$$L = \sum_t L_t$$

Core idea behind ELMo (out-of-class)

- **Bottom layer 1:** input each word by as a sequence of characters into a character-level network.
- Then feed its output feature vectors character by character as input into a word-level RNN at the top.
- **No problems with out of vocabulary words, because input of a word is done as a sequence of characters!**


use hidden state after each word
LSTM
highway net feature map (one for each character)
$y_i = t(x_i) \odot g(W_1 x_i + b_1) + (1 - t(x_i)) \odot x_i$
2-layer highway net
cnn output feature map (one for each character)
1d-CNN (ksize 3, w padding)
character embedding feature map
$y_i = Qx_i$
$x_1 = e('T')$
$x_2 = e('h')$
character embedding
T h e a n t i s

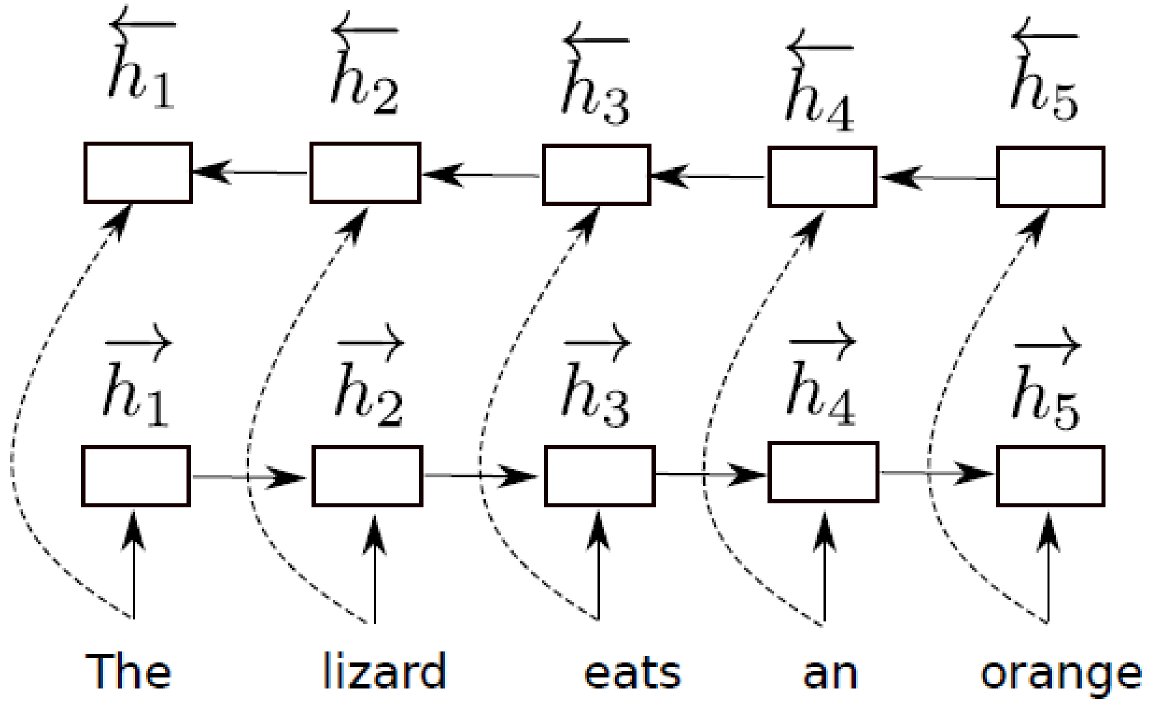
Core idea behind ELMo (out-of-class)

- **On the top:** use as word-level RNNs: a variant of **bidirectional LSTMs**.
- **Bidirectional:** Two hidden states, going in two opposite directions, denoted $\overrightarrow{\ldots}$ and $\overleftarrow{\ldots}$ respectively.

$$\overrightarrow{h_k} = RNN_r(w_1, w_2 \ldots, w_k)$$
$$\overleftarrow{h_k} = RNN_l(w_L, w_{L-1} \ldots, w_k)$$
$$h_k = [\overleftarrow{h_k}, \overrightarrow{h_k}]$$

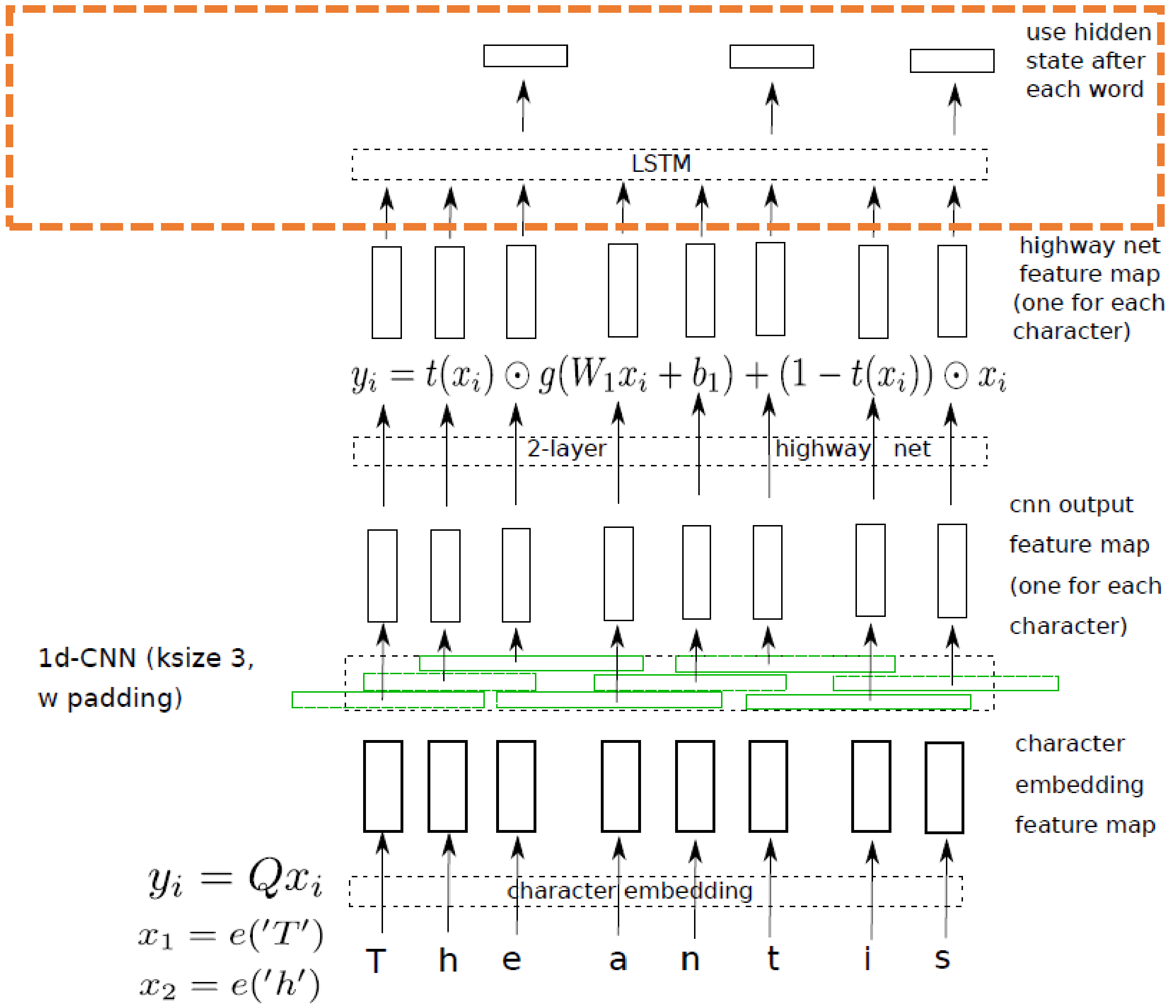
With L being the length of the sentence

$$h_k = [\overleftarrow{h}_k, \overrightarrow{h}_k]$$



Core idea behind ELMo (out-of-class)

- **On the top:** use as word-level RNNs: a variant of **bidirectional LSTMs**.
- This bidirectional LSTM will be able to compute **some context using the LSTM architecture** for the word embedding w_k.
- Final word embedding will be stored in $h_k = [\overleftarrow{h_k}, \overrightarrow{h_k}]$!

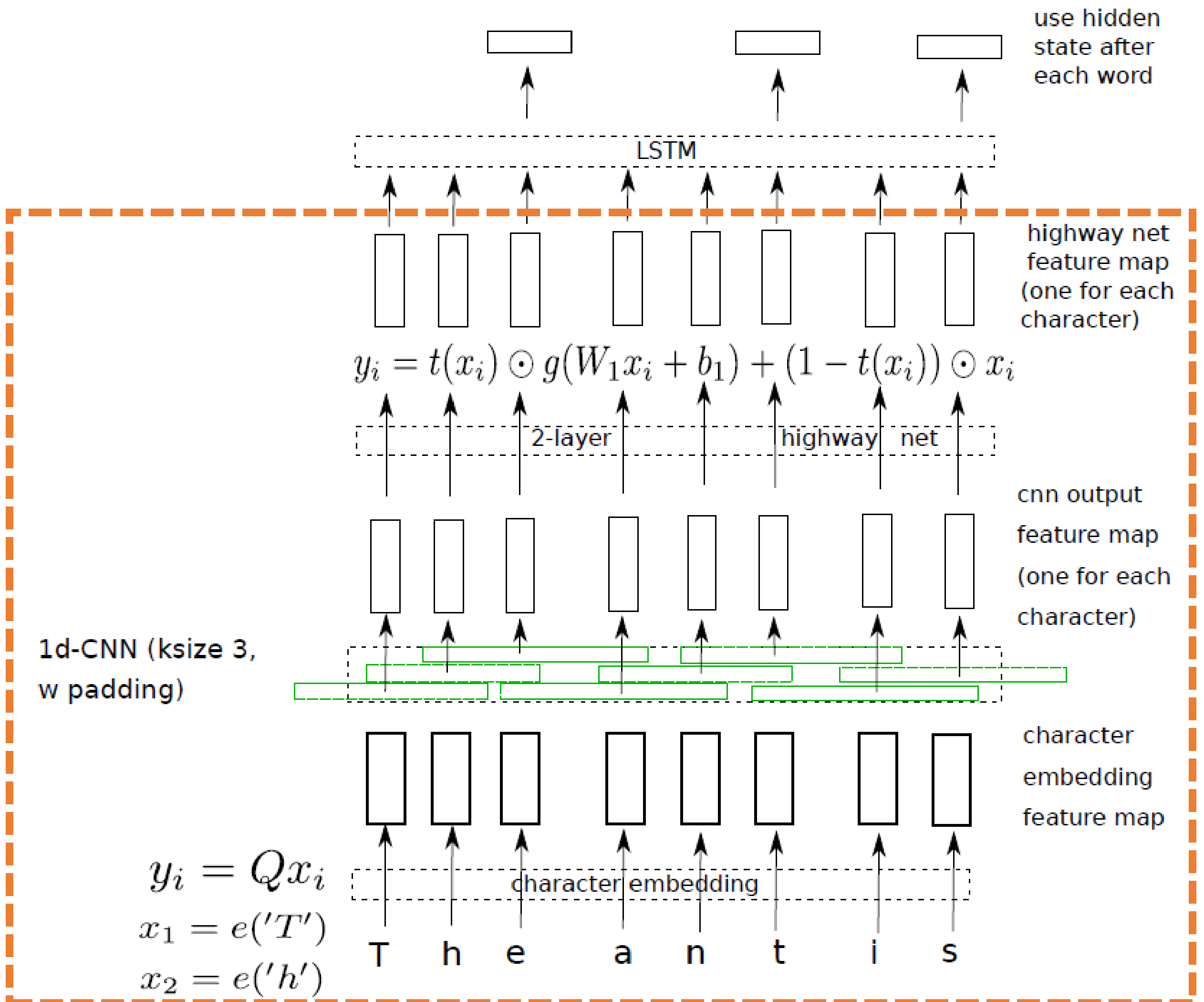


use hidden state after each word
LSTM
highway net feature map (one for each character)
$y_i = t(x_i) \odot g(W_1 x_i + b_1) + (1 - t(x_i)) \odot x_i$
2-layer highway net
cnn output feature map (one for each character)
1d-CNN (ksize 3, w padding)
character embedding feature map
$y_i = Qx_i$
$x_1 = e('T')$
$x_2 = e('h')$
character embedding
T h e a n t i s

Core idea behind ELMo (out-of-class)

- **In the bottom and middle:** The character embedding is a simple embedding layer $y = Qx$, with trainable Q.

- It is followed by a character-level 1D CNN layer on the embedded character encoding vectors Qx.
- It then passes these embeddings to a 2-layer **highway network**.


use hidden state after each word
LSTM
highway net feature map (one for each character)
$y_i = t(x_i) \odot g(W_1 x_i + b_1) + (1 - t(x_i)) \odot x_i$
2-layer highway net
cnn output feature map (one for each character)
1d-CNN (ksize 3, w padding)
character embedding feature map
$y_i = Qx_i$
$x_1 = e('T')$
$x_2 = e('h')$
character embedding
T h e a n t i s

Core idea behind ELMo (out-of-class)

Definition (Highway Layer):

The **Highway Layer** is another variant of the fully connected layer, with an additional gated residual connection.

You only need to know that it produces a standard mapping with a non-linear activation function $y = g(W_1 x + b_1)$ like any standard fully connected layer.

However, its full propagation formula will include a gated residual, as shown below. It consists of a transform gate and a carry gate.

$$z = t \odot g(W_1 x + b_1) + (1 - \mathrm{t}) \odot x$$

With

$$t = \sigma(W_2 x + b_2)$$

Similar to residual connections, this layer helps with information flow.

ELMo

Key takeaways on ELMo

- ELMo is a pretty recent word embedding (2018!), which has, for a long time, held the **state-of-the-art performance for word embeddings**.
- These days (2021-2022), the state of the art is the **BERT embedding.** (to be discussed on the next lecture, after the lesson on transformers!)
- It is **able to operate on out-of-vocabulary words**, as it **takes characters as inputs**, and can process context to address Problem #1 (as opposed to CBoW/SG & FastText)
- **Its main downside is the computational cost** of the architecture. Preferable to use **FastText** if you need speed!

Supervised and combined learning?

- Word2Vec, GloVe and ELMo approaches are commonly referred to as **unsupervised (or semi-supervised) approaches to embedding,** both relying on the **distributional hypothesis** we mentioned on the previous lecture.
- Unsupervised representation learning of sentences had been the norm for quite some time.
- **More advanced versions of unsupervised approaches exist!**

- However, the last few years have seen a shift toward **supervised** and **multi-task learning schemes** with a number of notably interesting proposals in late 2017/early 2018.

From unsupervised to combined learning

For a long time, there was a general consensus (see [Wang2018] for details) in the field that

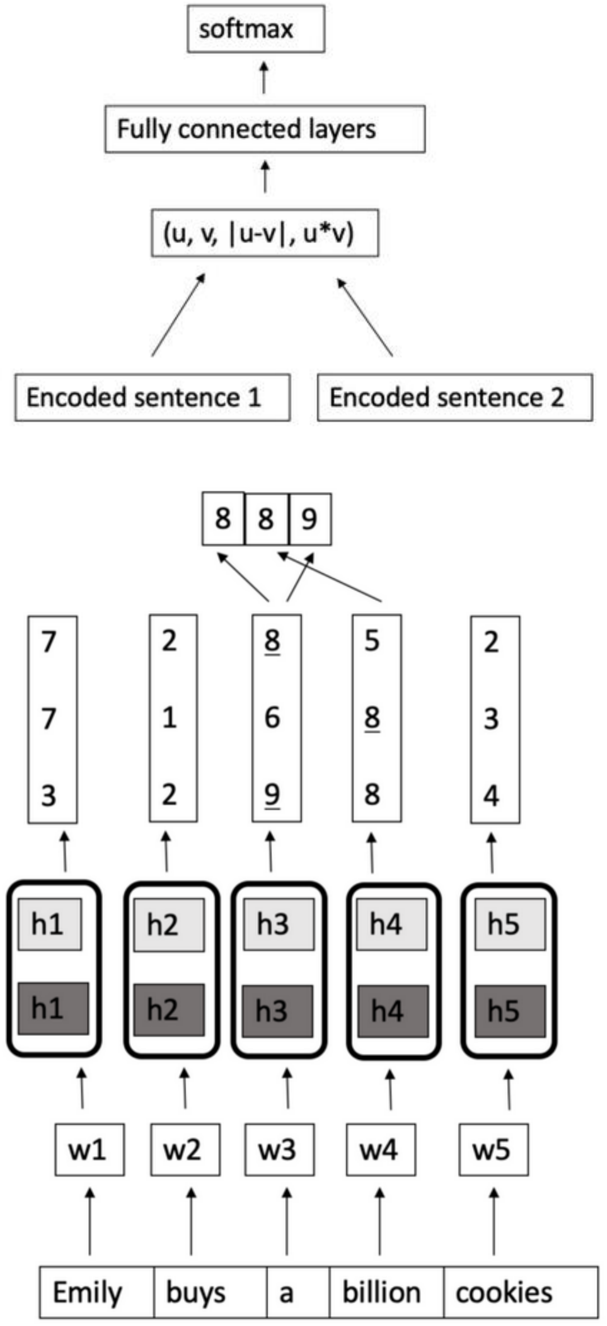
- the simple approach of **simply averaging a sentence's word vectors** (as in CBoW approach) gives a strong baseline for context, which is good enough for many downstream tasks.
- **supervised** learning of sentence/word embeddings was costly and gave lower-quality embeddings than unsupervised/semi-supervised approaches.

But these assumptions have recently been overturned, in part following the publication of the InferSent ([Conneau2018]) results.

Counter-example: InferSent (out-of-class)

Simple key takeaways

- The InferSent model (in [Conneau2018]) is an example of a supervised classification model, with three classes: Entailment, Neutral, Contradiction.
- It uses the same forward-backward LSTM architecture as ELMo, and attempts to compare two sentences and classify the pair.
- Feature representation extracted from model showed good performance.



From unsupervised to combined learning

- Following these recent results, showing that supervised approaches are not necessarily bad…
- The current direction in research for word embeddings suggests to combine both supervised and unsupervised learning techniques. This is commonly referred to as a **multi-task learning based embedding.**
- The most notable work in this direction would be **Google's Universal Sentence and Word Encoder** (in [Cer2019]). Kept rather hidden, have only seen a pre-release on Tensorflow: https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder
- (But nothing for PyTorch yet?)

Conclusion (W9S2)

We have seen a few approaches to embeddings.

- Train an AI to figure out embeddings? Basic approaches.
    - (CBoW)
    - SkipGram
- Out of dictionary entries?
    - Use ELMo!
- Need something a bit less universal but faster?
    - Use FastText!

Advanced methods

- Supervised: InferSent
- Multi-task: Google's Universal Embedding.

A few more problems are still open at the moment for these word embeddings.

- Can these embedding be biased? Yes, unfortunately.
- Can we help the network in computing context? BERT.

Learn more about these topics

Out of class, for those of you who are curious

- [Mikolov2013] **Mikolov** et al., “Efficient Estimation of Word Representations in Vector Space”, 2013. https://arxiv.org/abs/1301.3781
- [Mikilov2014] **Mikolov** et al., “Distributed Representations of Words and Phrases and their Compositionality”, 2014. https://arxiv.org/abs/1310.4546
- [GloVe2014] Pennington et al., “GloVe: Global Vectors for Word Representation”, 2014. https://nlp.stanford.edu/projects/glove/

Learn more about these topics

Out of class, for those of you who are curious

- [Mikolov2018] **Mikolov** et al., “Advances in Pre-Training Distributed Word Representations”, 2018. https://arxiv.org/abs/1301.3781
- [Peters2018] **Peters** et al., “Deep contextualized word representations”, 2018. https://arxiv.org/abs/1802.05365
- [Wang2018] Wang et al., “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding” https://arxiv.org/abs/1804.07461

Learn more about these topics

Out of class, for those of you who are curious

- [Conneau2018] **Conneau** et al., “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”, 2018. https://arxiv.org/abs/1705.02364
- [Cer2019] **Cer** et al., “Universal Sentence Encoder”, 2019. https://arxiv.org/abs/1803.11175

Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Alexis Conneau:** Senior Researcher at **Google**.
  https://scholar.google.fr/citations?user=45KfCpgAAAAJ&hl=fr
- **Matthew E. Peters:** Senior researcher at **Allen Institute.**
  https://scholar.google.com/citations?user=K5nCPZwAAAAJ&hl=en
- **Daniel Cer:** Researcher at **Google** and Lecturer at **UC Berkeley.**
  Has a really good course about NLP and Embeddings.
  https://scholar.google.com/citations?user=BrT1NW8AAAAJ&hl=en
  https://www.ischool.berkeley.edu/user/4406
  https://www.ischool.berkeley.edu/courses/datasci/266