IN5550: Neural Methods in Natural Language Processing
Sub-lecture 4.2
Using embeddings

Andrey Kutuzov

University of Oslo

10 February 2022

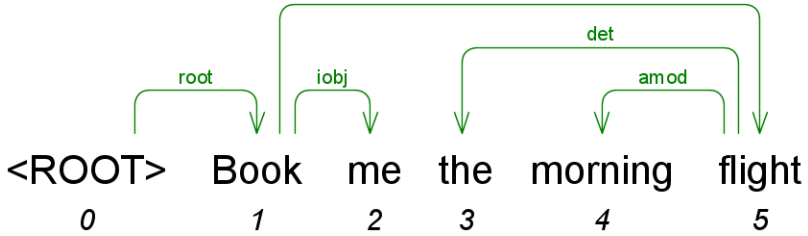


Contents



1. Dense representations (embeddings)
   - Combining embeddings
   - Sources of embeddings: external tasks

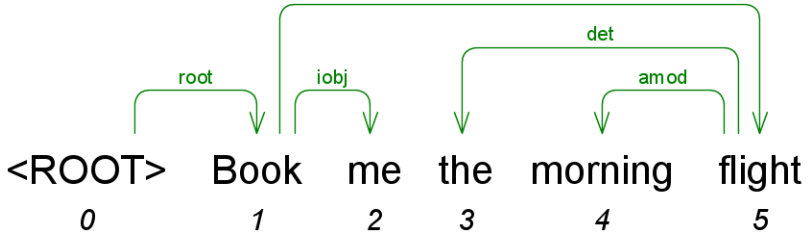
Dense representations (embeddings)



Example of dense features in parsing task

(see also the PoS tagging example in [Goldberg, 2017])

Dense representations (embeddings)

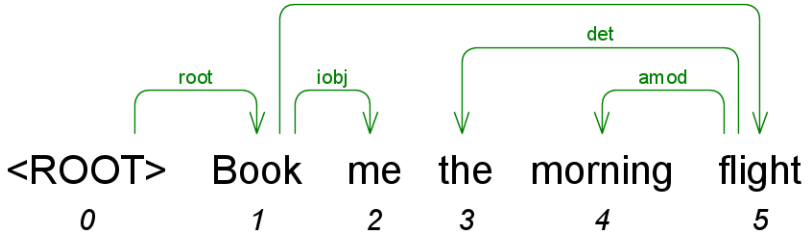


Example of dense features in parsing task

(see also the PoS tagging example in [Goldberg, 2017])

- One of the first neural dependency parsers with dense features is described in [Chen and Manning, 2014].

Dense representations (embeddings)

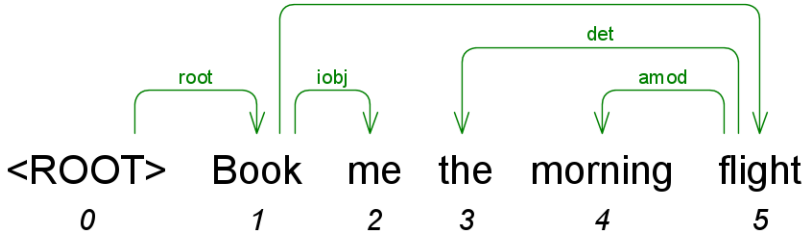


Example of dense features in parsing task

(see also the PoS tagging example in [Goldberg, 2017])

- One of the first neural dependency parsers with dense features is described in [Chen and Manning, 2014].
- Conceptually it is a classic Arc-Standard transition-based parser.

Dense representations (embeddings)

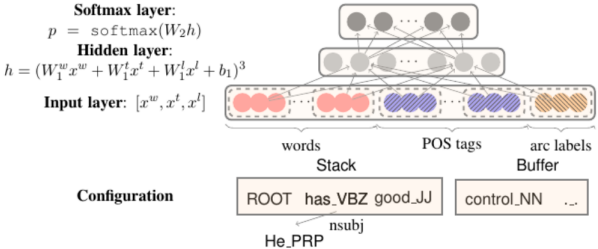


Example of dense features in parsing task

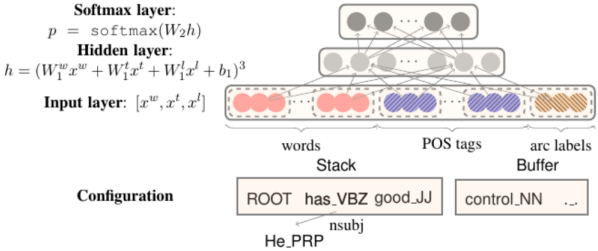
(see also the PoS tagging example in [Goldberg, 2017])

- One of the first neural dependency parsers with dense features is described in [Chen and Manning, 2014].
- Conceptually it is a classic Arc-Standard transition-based parser.
- The difference is in the features it uses:
- Dense embeddings $\boldsymbol{w}, \boldsymbol{t}, \boldsymbol{l} \in \mathbb{R}^{50}$ for words, PoS tags and dependency labels;
  - *nowadays, we usually use* $\mathbb{R}^{300}$ *(or so) embeddings for words*

Dense representations (embeddings)



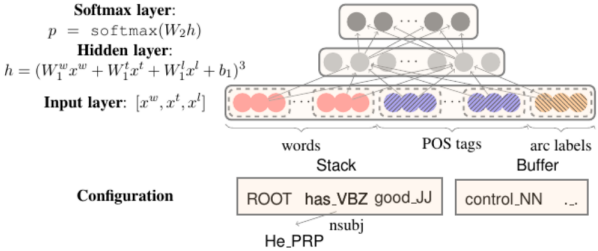
Dense representations (embeddings)



Parsing with dense representations and neural networks (simplified)

- Concatenated embeddings of words ($\boldsymbol{x}^w$), PoS tags ($\boldsymbol{x}^t$) and dependency labels ($\boldsymbol{x}^l$) from the stack are given as input layer.

Dense representations (embeddings)



Parsing with dense representations and neural networks (simplified)

- Concatenated embeddings of words ($\boldsymbol{x}^w$), PoS tags ($\boldsymbol{x}^t$) and dependency labels ($\boldsymbol{x}^l$) from the stack are given as input layer.
- 200-dimensional hidden layer represents the actual features used for predictions.

Dense representations (embeddings)



Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:

Dense representations (embeddings)



Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:
  - minimize the cross-entropy loss $L(\theta)$

Dense representations (embeddings)

Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:
  - minimize the cross-entropy loss $L(\theta)$
  - maximize the probability of correct transitions t_i in a collection of n configurations;.

Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:
  - minimize the cross-entropy loss $L(\theta)$
  - maximize the probability of correct transitions t_i in a collection of n configurations;.
  - L2 regularization (weight decay) with tunable λ:

$$L(\theta) = -\sum_{i}^{n} \log(t_i) + \frac{\lambda}{2}\|\theta\| \quad (1)$$

Dense representations (embeddings)

Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:
  - minimize the cross-entropy loss $L(\theta)$
  - maximize the probability of correct transitions t_i in a collection of n configurations;.
  - L2 regularization (weight decay) with tunable λ:

$$L(\theta) = -\sum_{i}^{n} \log(t_i) + \frac{\lambda}{2}\|\theta\| \tag{1}$$

- **Most useful feature combinations are learned automatically in the hidden layer**!

Dense representations (embeddings)

Training the network

- Neural net in [Chen and Manning, 2014] is trained by gradually updating weights θ in the hidden layer and in all the embeddings:
  - minimize the cross-entropy loss $L(\theta)$
  - maximize the probability of correct transitions t_i in a collection of n configurations;.
  - L2 regularization (weight decay) with tunable λ:

$$L(\theta) = -\sum_{i}^{n} \log(t_i) + \frac{\lambda}{2}\|\theta\| \quad (1)$$

- **Most useful feature combinations are learned automatically in the hidden layer**!
- Notably, the model employs the unusual cube activation function $g(x) = x^3$

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;
5. apply the transition with the highest probability.

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;
5. apply the transition with the highest probability.

Word embeddings

- One can start with randomly initialized word embeddings.

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;
5. apply the transition with the highest probability.

Word embeddings

- One can start with randomly initialized word embeddings.
  - They will be pushed towards useful values during the training by back-propagation

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;
5. apply the transition with the highest probability.

Word embeddings

- One can start with randomly initialized word embeddings.
  - They will be pushed towards useful values during the training by back-propagation
- Or one can use pre-trained word vectors for initialization.

Dense representations (embeddings)



When parsing (at inference time):

1. Look at the current configuration;
2. lookup the necessary embeddings for $\boldsymbol{x}^w$, $\boldsymbol{x}^t$ and $\boldsymbol{x}^l$;
3. feed them as input to the hidden layer;
4. compute softmax probabilities for all possible transitions;
5. apply the transition with the highest probability.

Word embeddings

- One can start with randomly initialized word embeddings.
  - They will be pushed towards useful values during the training by back-propagation
- Or one can use pre-trained word vectors for initialization.
- More on this next week.

Dense representations (embeddings)



The neural parser by Chen and Manning achieved excellent performance in 2014

- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)

Dense representations (embeddings)



The neural parser by Chen and Manning achieved excellent performance in 2014

- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7

Dense representations (embeddings)



The neural parser by Chen and Manning achieved excellent performance in 2014

- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7
  - *MSTParser* 90.5

Dense representations (embeddings)



The neural parser by Chen and Manning achieved excellent performance in 2014

- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7
  - *MSTParser* 90.5
- 2 times faster than *MaltParser*;

Dense representations (embeddings)



The neural parser by Chen and Manning achieved excellent performance in 2014

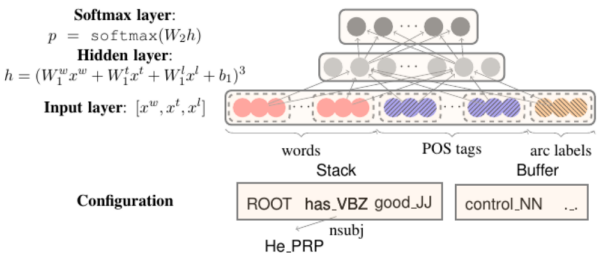
- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7
  - *MSTParser* 90.5
- 2 times faster than *MaltParser*;
- 100 times faster than *MSTParser*.

Dense representations (embeddings)

The neural parser by Chen and Manning achieved excellent performance in 2014

- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7
  - *MSTParser* 90.5
- 2 times faster than *MaltParser*;
- 100 times faster than *MSTParser*.

...started the widespread usage of dense representations in NLP.



Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...

Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...
- ...when used with deep neural networks.

Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...
- ...when used with deep neural networks.
- If you use sparse BoW as features (like in Obligatory 1), your first hidden layer size is most certainly much smaller than the size of vocabulary;

Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...
- ...when used with deep neural networks.
- If you use sparse BoW as features (like in Obligatory 1), your first hidden layer size is most certainly much smaller than the size of vocabulary;
- then it learns dense representations for the words anyway (in the first weight matrix).

Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...
- ...when used with deep neural networks.
- If you use sparse BoW as features (like in Obligatory 1), your first hidden layer size is most certainly much smaller than the size of vocabulary;
- then it learns dense representations for the words anyway (in the first weight matrix).
- When using dense inputs outright, we simply make it explicit;

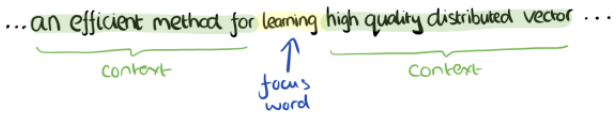
Dense representations (embeddings)



One-hot VS dense vectors

- Conceptually these two representations are similar...
- ...when used with deep neural networks.
- If you use sparse BoW as features (like in Obligatory 1), your first hidden layer size is most certainly much smaller than the size of vocabulary;
- then it learns dense representations for the words anyway (in the first weight matrix).
- When using dense inputs outright, we simply make it explicit;
- It is also usually more efficient.

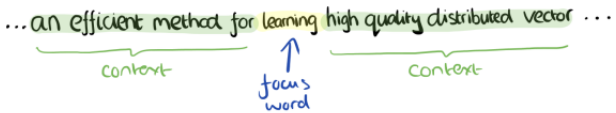
Combining embeddings



Many features, one input vector

- Before feeding embeddings into network, one must somehow combine them.

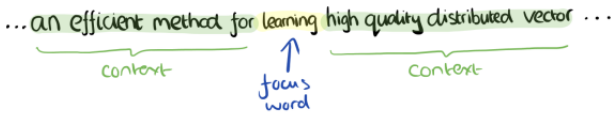
Combining embeddings



Many features, one input vector

- Before feeding embeddings into network, one must somehow combine them.
- Consider the focus word '*learning*' above...
- ...and the context words in 2-token window to its right and left.

Combining embeddings



Many features, one input vector

- Before feeding embeddings into network, one must somehow combine them.
- Consider the focus word '*learning*' above...
- ...and the context words in 2-token window to its right and left.
- We want to somehow represent the focus word using only its context.

Combining embeddings



Many features, one input vector

- Before feeding embeddings into network, one must somehow combine them.
- Consider the focus word '*learning*' above...
- ...and the context words in 2-token window to its right and left.
- We want to somehow represent the focus word using only its context.
- Each unique word is assigned a dense vector:
  - '*method*' $\rightarrow \boldsymbol{a}$
  - '*for*' $\rightarrow \boldsymbol{b}$
  - '*high*' $\rightarrow \boldsymbol{c}$
  - '*quality*' $\rightarrow \boldsymbol{d}$

Combining embeddings



What can be the input vector x representing 'learning'?

Combining embeddings



What can be the input vector $\boldsymbol{x}$ representing 'learning'?

- We can concatenate:
  $\boldsymbol{x} = [\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$

Combining embeddings



What can be the input vector $\boldsymbol{x}$ representing 'learning'?

- We can concatenate:
  $\boldsymbol{x} = [\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$
- We can sum (the case for Obligatory 1):
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$

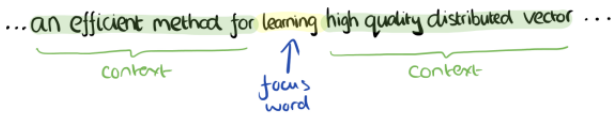
Combining embeddings



What can be the input vector $\boldsymbol{x}$ representing 'learning'?

- We can concatenate:
  $\boldsymbol{x} = [\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$
- We can sum (the case for Obligatory 1):
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$
- We can average:
  $\boldsymbol{x} = \frac{\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}}{4}$

Combining embeddings



What can be the input vector $\boldsymbol{x}$ representing 'learning'?

- We can concatenate:
  $\boldsymbol{x} = [\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$
- We can sum (the case for Obligatory 1):
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$
- We can average:
  $\boldsymbol{x} = \frac{\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}}{4}$
- Various weights may be applied to the vectors...
- etc.

Combining embeddings



What can be the input vector $\boldsymbol{x}$ representing 'learning'?

- We can concatenate:
  $\boldsymbol{x} = [\boldsymbol{a}; \boldsymbol{b}; \boldsymbol{c}; \boldsymbol{d}]$
- We can sum (the case for Obligatory 1):
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$
- We can average:
  $\boldsymbol{x} = \frac{\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}}{4}$
- Various weights may be applied to the vectors...
- etc.

Question: what information is preserved only by concatenation?

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)
- Often a better solution is to get good pre-trained embeddings from elsewhere;
  - '*good*' here means '*similar entities have similar embeddings*'.

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)
- Often a better solution is to get good pre-trained embeddings from elsewhere;
  - '*good*' here means '*similar entities have similar embeddings*'.
- If only we had an auxiliary supervised task with more annotated data!

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)
- Often a better solution is to get good pre-trained embeddings from elsewhere;
  - '*good*' here means '*similar entities have similar embeddings*'.
- If only we had an auxiliary supervised task with more annotated data!
- This task could produce feature embeddings as a byproduct.

Dense representations (embeddings)



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...and train them with the rest of the network...
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)
- Often a better solution is to get good pre-trained embeddings from elsewhere;
  - '*good*' here means '*similar entities have similar embeddings*'.
- If only we had an auxiliary supervised task with more annotated data!
- This task could produce feature embeddings as a byproduct.

This is usually not the case :-(. What about unsupervised auxiliary tasks? Here comes language modeling.

References I


Chen, D. and Manning, C. (2014).
A fast and accurate dependency parser using neural networks.
In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 740–750.


Goldberg, Y. (2017).
Neural network methods for natural language processing.
Synthesis Lectures on Human Language Technologies, 10(1):1–309.