IN5550: Neural Methods in Natural Language Processing
Sub-lecture 4.1
Dense representations and how to use them

Andrey Kutuzov

University of Oslo



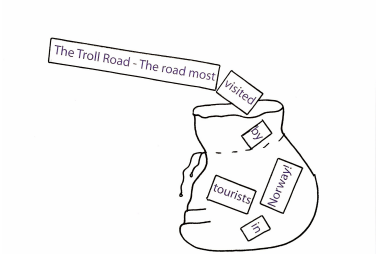
Contents



1. Dense Representations of Linguistic Features
   - One-hot representations: let's recall
   - Dense representations (embeddings)

2. Use of embeddings
   - Combining embeddings
   - Sources of embeddings: external tasks

One-hot representations: let's recall



- Bag-of-Words (BoW) feature vector of the document i can be interpreted as a sum of one-hot vectors ($\boldsymbol{o}$) for each token in it:

One-hot representations: let's recall



- Bag-of-Words (BoW) feature vector of the document i can be interpreted as a sum of one-hot vectors ($\boldsymbol{o}$) for each token in it:
  - Vocabulary V from the picture above contains 10 words (lowercased): *['-', 'by', 'in', 'most', 'norway', 'road', 'the', 'tourists', 'troll', 'visited']*.

One-hot representations: let's recall



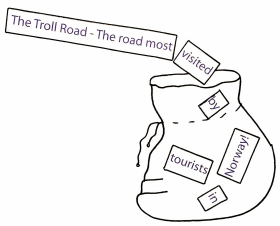
- Bag-of-Words (BoW) feature vector of the document i can be interpreted as a sum of one-hot vectors ($\boldsymbol{o}$) for each token in it:
  - Vocabulary V from the picture above contains 10 words (lowercased): *['-', 'by', 'in', 'most', 'norway', 'road', 'the', 'tourists', 'troll', 'visited']*.
  - $\boldsymbol{o}^0 = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]$ ('The')

One-hot representations: let's recall



- Bag-of-Words (BoW) feature vector of the document i can be interpreted as a sum of one-hot vectors ($\boldsymbol{o}$) for each token in it:
  - Vocabulary V from the picture above contains 10 words (lowercased): *['-', 'by', 'in', 'most', 'norway', 'road', 'the', 'tourists', 'troll', 'visited']*.
  - $\boldsymbol{o}^0 = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]$ ('The')
  - $\boldsymbol{o}^1 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$ ('Troll')
  - etc...

One-hot representations: let's recall



- Bag-of-Words (BoW) feature vector of the document i can be interpreted as a sum of one-hot vectors ($\boldsymbol{o}$) for each token in it:
  - Vocabulary V from the picture above contains 10 words (lowercased): *['-', 'by', 'in', 'most', 'norway', 'road', 'the', 'tourists', 'troll', 'visited']*.
  - $\boldsymbol{o}^0 = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]$ ('The')
  - $\boldsymbol{o}^1 = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$ ('Troll')
  - etc...
  - $\boldsymbol{i} = [1, 1, 1, 1, 1, 2, 2, 1, 1, 1]$ ('*the*' and '*road*' occurred 2 times)

One-hot representations: let's recall

- A document classifier can be trained on words represented with integer identifiers:
  - '*the*' is the word number 6 in the vocabulary
  - '*most*' is the word number 3 in the vocabulary
  - '*visited*' is the word number 9 in the vocabulary
  - etc.

One-hot representations: let's recall



- A document classifier can be trained on words represented with integer identifiers:
  - '*the*' is the word number 6 in the vocabulary
  - '*most*' is the word number 3 in the vocabulary
  - '*visited*' is the word number 9 in the vocabulary
  - etc.
- Such features are discrete (categorical).
  - a.k.a. one-hot.

One-hot representations: let's recall



- A document classifier can be trained on words represented with integer identifiers:
  - '*the*' is the word number 6 in the vocabulary
  - '*most*' is the word number 3 in the vocabulary
  - '*visited*' is the word number 9 in the vocabulary
  - etc.
- Such features are discrete (categorical).
  - a.k.a. one-hot.
- Each word is a feature on its own, completely independent from other words.

One-hot representations: let's recall



- A document classifier can be trained on words represented with integer identifiers:
  - '*the*' is the word number 6 in the vocabulary
  - '*most*' is the word number 3 in the vocabulary
  - '*visited*' is the word number 9 in the vocabulary
  - etc.
- Such features are discrete (categorical).
  - a.k.a. one-hot.
- Each word is a feature on its own, completely independent from other words.
- Other NLP tasks: categorical features for PoS tags, dependency labels, etc.

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.

One-hot representations: let's recall

Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.
- Not efficient:
  - A 50-words text is $\boldsymbol{x} \in \mathbb{R}^{100000}$, because 100K words in our vocabulary!

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.
- Not efficient:
  - A 50-words text is $\boldsymbol{x} \in \mathbb{R}^{100000}$, because 100K words in our vocabulary!
- A bit easier for other linguistic entities (parts of speech, etc)...

One-hot representations: let's recall

Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.
- Not efficient:
  - A 50-words text is $\boldsymbol{x} \in \mathbb{R}^{100000}$, because 100K words in our vocabulary!
- A bit easier for other linguistic entities (parts of speech, etc)...
- ...but their feature combinations yield millions of resulting features.

One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.
- Not efficient:
  - A 50-words text is $\boldsymbol{x} \in \mathbb{R}^{100000}$, because 100K words in our vocabulary!
- A bit easier for other linguistic entities (parts of speech, etc)...
- ...but their feature combinations yield millions of resulting features.
- It's very difficult to learn good weights for them all.

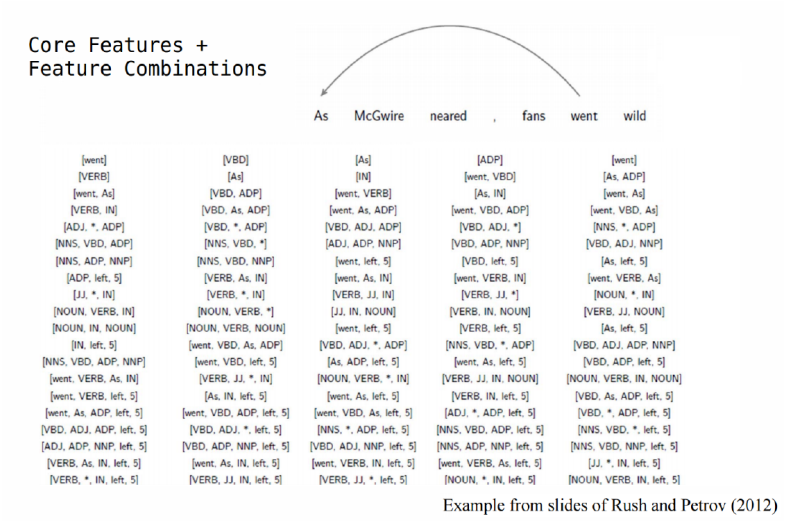
One-hot representations: let's recall



Why discrete features might be bad?

- Features for words are extremely sparse:
  - the feature '*word form*' can take any of tens or hundreds of thousands categorical values...
  - ...each absolutely unique and not related to each other.
- Classifiers have to learn weight matrices with $dim = |V|$.
- Not efficient:
  - A 50-words text is $\boldsymbol{x} \in \mathbb{R}^{100000}$, because 100K words in our vocabulary!
- A bit easier for other linguistic entities (parts of speech, etc)...
- ...but their feature combinations yield millions of resulting features.
- It's very difficult to learn good weights for them all.
- Feature extraction step haunted NLP practitioners for several decades.

One-hot representations: let's recall



Example from slides of Rush and Petrov (2012)

Feature model for parsing

'Is the 1st word to the right *wild*, and the 3rd word to the left a *verb*?'

One-hot representations: let's recall



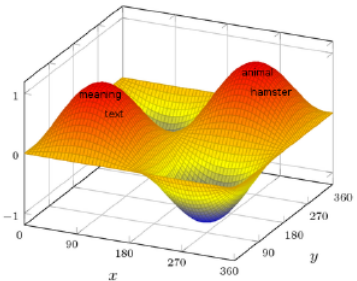
We can do better

- Is there a way to avoid using multitudes of discrete categorical features?

One-hot representations: let's recall

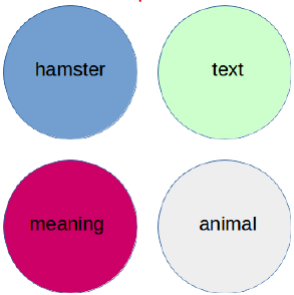
We can do better

- Is there a way to avoid using multitudes of discrete categorical features?
- Yes.
- Use dense continuous features.

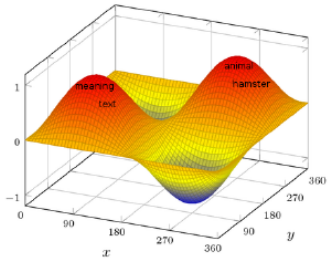


Dense representations (embeddings)

Discrete representations

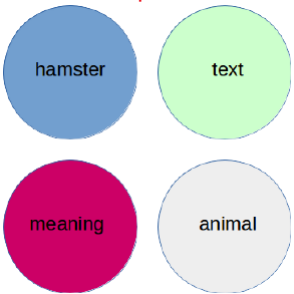


Continuous representations

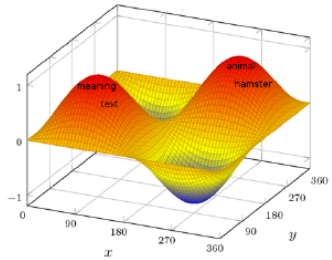


Dense representations (embeddings)

Discrete representations



Continuous representations



- We would like linguistic entities to be represented with some meaningful 'coordinates'.
- It would allow our models to understand whether entities (for example, words) are more or less similar with respect to the current task at hand.

Dense representations (embeddings)

Vectors as coordinates

- A vector is a sequence or an array of n real values:
  - $[0, 1, -2, 4]$ is a vector with 4 components/entries ($\in \mathbb{R}^4$);
  - $[200, 300, 1]$ is a vector with 3 components/entries ($\in \mathbb{R}^3$);

Dense representations (embeddings)

Vectors as coordinates

- A vector is a sequence or an array of n real values:
  - $[0, 1, -2, 4]$ is a vector with 4 components/entries ($\in \mathbb{R}^4$);
  - $[200, 300, 1]$ is a vector with 3 components/entries ($\in \mathbb{R}^3$);
- Components can be viewed as coordinates in an n-dimensional space;

Dense representations (embeddings)

Vectors as coordinates

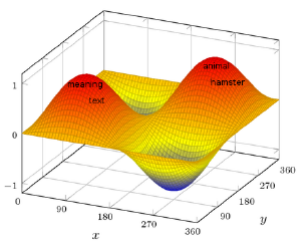
- A vector is a sequence or an array of n real values:
  - $[0, 1, -2, 4]$ is a vector with 4 components/entries ($\in \mathbb{R}^4$);
  - $[200, 300, 1]$ is a vector with 3 components/entries ($\in \mathbb{R}^3$);
- Components can be viewed as coordinates in an n-dimensional space;
- then a vector is a point in this space.

Dense representations (embeddings)

Vectors as coordinates

- A vector is a sequence or an array of n real values:
  - $[0, 1, -2, 4]$ is a vector with 4 components/entries ($\in \mathbb{R}^4$);
  - $[200, 300, 1]$ is a vector with 3 components/entries ($\in \mathbb{R}^3$);
- Components can be viewed as coordinates in an n-dimensional space;
- then a vector is a point in this space.

3-dimensional space:



Dense representations (embeddings)

Feature embeddings

- Say we have a vocabulary of size $|V|$;

Dense representations (embeddings)

Feature embeddings

- Say we have a vocabulary of size $|V|$;
- Let's embed these words into a d-dimensional space.

Dense representations (embeddings)

Feature embeddings

- Say we have a vocabulary of size $|V|$;
- Let's embed these words into a d-dimensional space.
- $d \ll |V|$
- e.g., $d = 100$ for a vocabulary of 100 000 words.

Dense representations (embeddings)

Feature embeddings

- Say we have a vocabulary of size $|V|$;
- Let's embed these words into a d-dimensional space.
- $d \ll |V|$
- e.g., $d = 100$ for a vocabulary of 100 000 words.
- Each word is associated with its own d-dimensional embedding vector;

Dense representations (embeddings)

Feature embeddings

- Say we have a vocabulary of size $|V|$;
- Let's embed these words into a d-dimensional space.
- $d \ll |V|$
- e.g., $d = 100$ for a vocabulary of 100 000 words.
- Each word is associated with its own d-dimensional embedding vector;
- These embeddings are part of θ;
- can be trained together with the rest of the network.

Dense representations (embeddings)

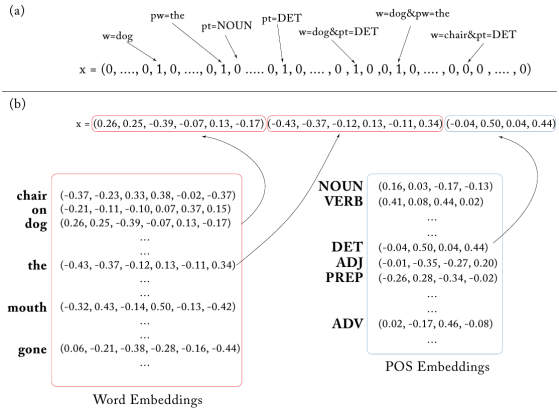
Feature embeddings

- Say we have a vocabulary of size $|V|$;
- Let's embed these words into a d-dimensional space.
- $d \ll |V|$
- e.g., $d = 100$ for a vocabulary of 100 000 words.
- Each word is associated with its own d-dimensional embedding vector;
- These embeddings are part of θ;
- can be trained together with the rest of the network.

Neural networks often do it implicitly during the training (in the hidden layers).
This is representation learning.

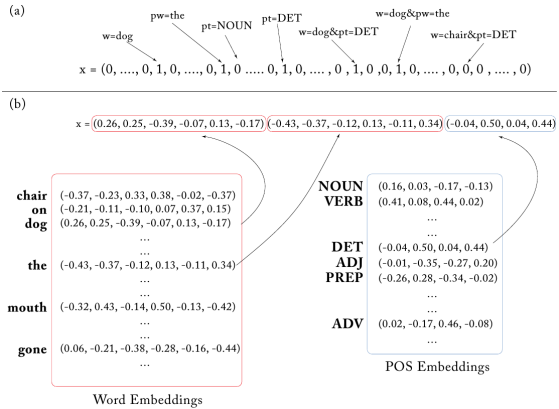
Representation learning

Sparse (a) and dense (b) feature representations for
'the_DET dog'
(part of speech tagging task)



Representation learning

Sparse (a) and dense (b) feature representations for
'the_DET dog'
(part of speech tagging task)



Q: what are the dimensionalities of word and PoS embeddings here?

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).
- Feature vectors are dense, not sparse (more computationally efficient).

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).
- Feature vectors are dense, not sparse (more computationally efficient).
- Generalization power: similar entities get similar embeddings (hopefully).
  - '*town*' vector is closer to '*city*' vector than to '*banana*' vector (semantics);

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).
- Feature vectors are dense, not sparse (more computationally efficient).
- Generalization power: similar entities get similar embeddings (hopefully).
  - '*town*' vector is closer to '*city*' vector than to '*banana*' vector (semantics);
  - *NOUN* vector is closer to *ADJ* vector than to *VERB* vector (grammar);

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).
- Feature vectors are dense, not sparse (more computationally efficient).
- Generalization power: similar entities get similar embeddings (hopefully).
  - '*town*' vector is closer to '*city*' vector than to '*banana*' vector (semantics);
  - *NOUN* vector is closer to *ADJ* vector than to *VERB* vector (grammar);
  - *iobj* vector is closer to *obj* vector than to *punct* vector (syntax).

Dense representations (embeddings)

Main benefits of continuous features

- Dimensionality of representations usually low (50-300).
- Feature vectors are dense, not sparse (more computationally efficient).
- Generalization power: similar entities get similar embeddings (hopefully).
  - '*town*' vector is closer to '*city*' vector than to '*banana*' vector (semantics);
  - *NOUN* vector is closer to *ADJ* vector than to *VERB* vector (grammar);
  - *iobj* vector is closer to *obj* vector than to *punct* vector (syntax).
- Same features in different positions can share statistical strength:
  - A token 2 words to the right and a token 2 words to the left can be one and the same word.
  - Would be good for the model to use this knowledge.

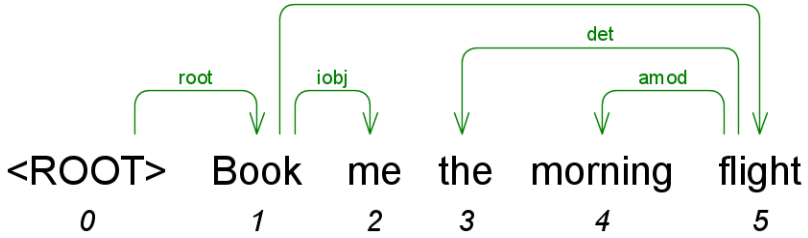
Contents



1. Dense Representations of Linguistic Features
   - One-hot representations: let's recall
   - Dense representations (embeddings)

2. Use of embeddings
   - Combining embeddings
   - Sources of embeddings: external tasks

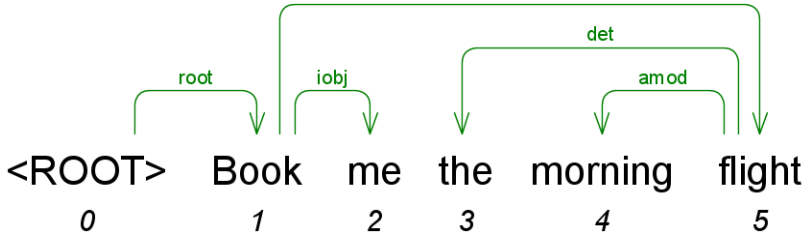
Use of embeddings



Example of dense features in parsing task

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

Use of embeddings

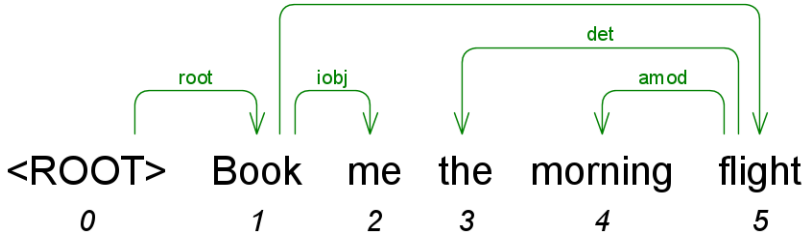


Example of dense features in parsing task

(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].

Use of embeddings

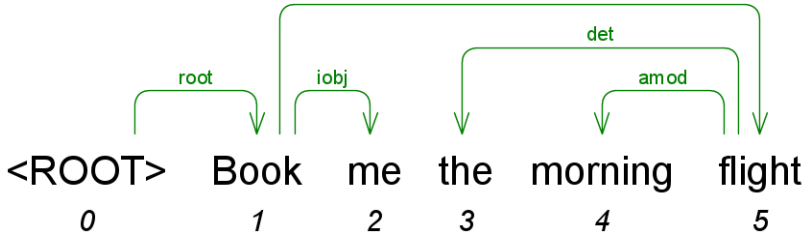


Example of dense features in parsing task

(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.

Use of embeddings

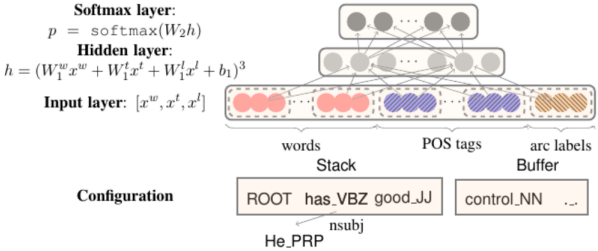


Example of dense features in parsing task

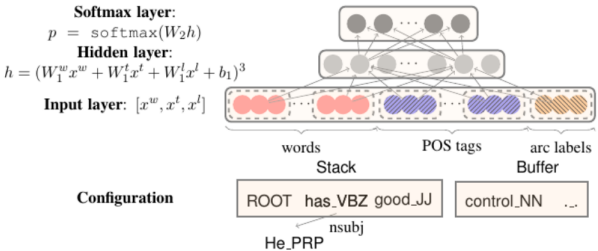
(see also the PoS tagging example in [Goldberg, 2017])

- 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* $\mathbb{R}^{768}$*) embeddings for words*

Use of embeddings



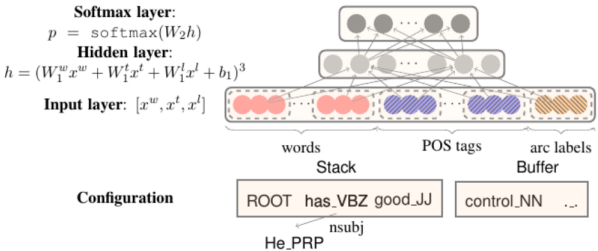
Use of embeddings



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.

Use of 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.

Use of 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:

Use of 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 $L(\theta)$

Use of 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;.

Use of 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;.
- **Most useful feature combinations are learned automatically in the hidden layer**!

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

When parsing (at inference time):

1. Look at the current configuration;

Use of 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;.
- **Most useful feature combinations are learned automatically in the hidden layer**!
- Notably, the model employs the unusual cube activation function $g(\boldsymbol{x}) = \boldsymbol{x}^3$

When parsing (at inference time):

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

Use of 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;.
- **Most useful feature combinations are learned automatically in the hidden layer**!
- Notably, the model employs the unusual cube activation function $g(\boldsymbol{x}) = \boldsymbol{x}^3$

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;

Use of 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;.
- **Most useful feature combinations are learned automatically in the hidden layer**!
- Notably, the model employs the unusual cube activation function $g(\boldsymbol{x}) = \boldsymbol{x}^3$

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;

Use of 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;.
- **Most useful feature combinations are learned automatically in the hidden layer**!
- Notably, the model employs the unusual cube activation function $g(\boldsymbol{x}) = \boldsymbol{x}^3$

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.

Use of embeddings



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

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

Use of 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

Use of 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

Use of 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*;

Use of embeddings



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

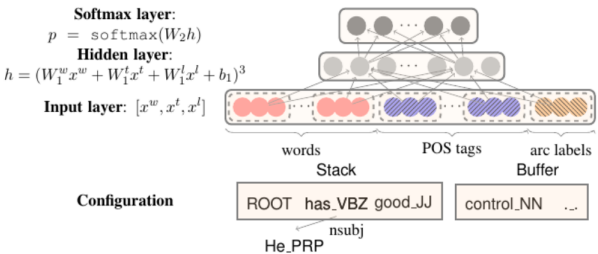
- Labeled Attachment Score (LAS) 90.7 on English *Penn TreeBank* (PTB)
  - *MaltParser* 88.7
  - *MSTParser* 90.5
- 2 times faster than *MaltParser*;
- 100 times faster than *MSTParser*.

Use of embeddings

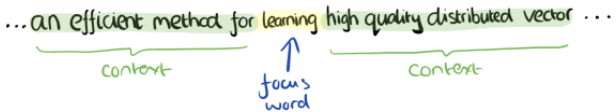
The neural parser by Chen and Manning achieved excellent performance in 2014

- 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.



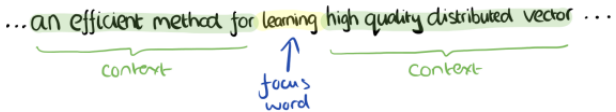
Combining embeddings



Many features, one input vector

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

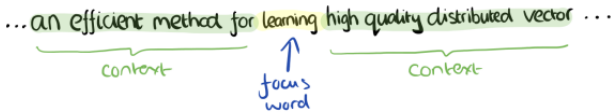
Combining embeddings



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.

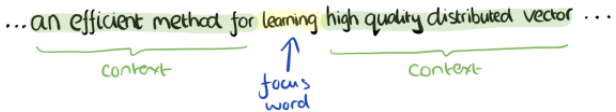
Combining embeddings



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

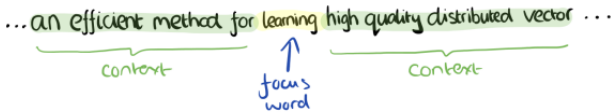
- Before feeding embeddings into network, one must somehow combine them.
- Consider the focus word '*learning*' above...
- ...and the context words in 2-token window to its right and left.
- We want to somehow represent the focus word using only its context.
- Each unique word is assigned a dense vector: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

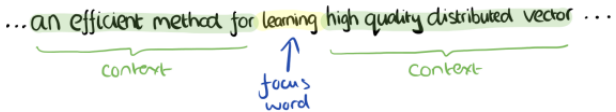
Combining embeddings



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

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

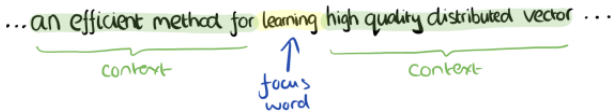
Combining embeddings



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

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

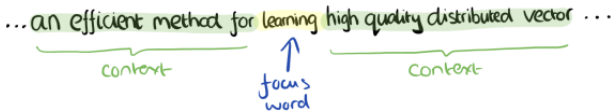
Combining embeddings



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:
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$
- We can average:
  $\boldsymbol{x} = \frac{\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}+\boldsymbol{d}}{4}$

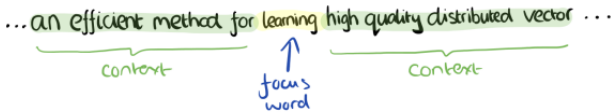
Combining embeddings



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

- We can concatenate:
  $x = [a; b; c; d]$
- We can sum:
  $x = a + b + c + d$
- We can average:
  $x = \frac{a+b+c+d}{4}$
- 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:
  $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c} + \boldsymbol{d}$
- 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?

Sources of embeddings: external tasks



I want good vectors for my features!

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

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.
- ...but then you must have enough supervised data to learn good representations.

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.
- ...but then you must have enough supervised data to learn good representations.
- Especially difficult for words (too many of them!)

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.
- ...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*'.

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.
- ...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 unsupervised auxiliary task...
- which could produce good word embeddings as a byproduct.

Sources of embeddings: external tasks



I want good vectors for my features!

- It is possible to treat feature embeddings as all other θ parameters...
- ...start with randomly initialized embeddings...
- ...and train them with the rest of the network...
- ...they will be pushed towards useful values by back-propagation.
- ...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 unsupervised auxiliary task...
- which could produce good word embeddings as a byproduct.

Here comes distributional hypothesis (see *sub-lecture 4.2*).

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.