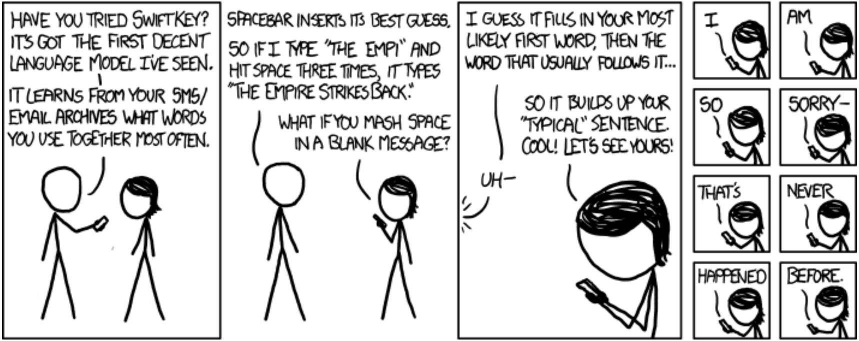
Contemporary Natural Language Processing reflected in Language Modeling

Andrey Kutuzov

UTMN

3-4 July 2019


HAVE YOU TRIED SWIFTKEY? IT'S GOT THE FIRST DECENT LANGUAGE MODEL I'VE SEEN.
IT LEARNS FROM YOUR SMS/ EMAIL ARCHIVES WHAT WORDS YOU USE TOGETHER MOST OFTEN.
SPACEBAR INSERTS ITS BEST GUESS. SO IF I TYPE "THE EMPI" AND HIT SPACE THREE TIMES, IT TYPES "THE EMPIRE STRIKES BACK."
WHAT IF YOU MASH SPACE IN A BLANK MESSAGE?
I GUESS IT FILLS IN YOUR MOST LIKELY FIRST WORD, THEN THE WORD THAT USUALLY FOLLOWS IT...
SO IT BUILDS UP YOUR "TYPICAL" SENTENCE. COOL! LET'S SEE YOURS!
UH-
I
AM
SO
SORRY-
THAT'S
NEVER
HAPPENED
BEFORE.

(XKCD)

Outline

- 15:45 - 17:15 : 1st part
- 17:15 - 17:30 : break
- 17:30 - 19:00 : 2nd part

What we will cover

1. 'Toy' language models:
   - Random LM
   - Frequency-based LM
2. Markov trigram LMs
3. LMs based on recurrent neural networks (RNNs):
   - Long short-term memory (LSTM)
   - other types, if time allows

```
git clone https://github.com/akutuzov/nlp_lm.git
```

Outline

- 15:45 - 17:15 : 1st part
- 17:15 - 17:30 : break
- 17:30 - 19:00 : 2nd part

What we will cover

1. 'Toy' language models:
   - Random LM
   - Frequency-based LM
2. Markov trigram LMs
3. LMs based on recurrent neural networks (RNNs):
   - Long short-term memory (LSTM)
   - other types, if time allows

`git clone https://github.com/akutuzov/nlp_lm.git`

Outline

- 15:45 - 17:15 : 1st part
- 17:15 - 17:30 : break
- 17:30 - 19:00 : 2nd part

What we will cover

1. 'Toy' language models:
   - Random LM
   - Frequency-based LM
2. Markov trigram LMs
3. LMs based on recurrent neural networks (RNNs):
   - Long short-term memory (LSTM)
   - other types, if time allows

```
git clone https://github.com/akutuzov/nlp_lm.git
```

Contents

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \quad (1)$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \quad (1)$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \tag{1}$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \tag{1}$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \qquad (1)$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \tag{1}$$

Language modeling task definition

Modeling linguistic sequences

- Task 1: to assign probabilities to natural language sequences:
  - 'What is the probability of *lazy dog*?'
  - 'What is the probability of *The quick brown fox jumps over the lazy dog*?'
  - 'What is the probability of *green colorless ideas sleep furiously*?'
- Task 2: to assign a probability for the likelihood of a word a to follow a word sequence S of length n:
  - 'What is the probability of seeing *jumps* after *The quick brown fox*?'
- These two tasks are mathematically equivalent.

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \quad (1)$$

Language modeling task definition

Markov assumption

- Multiplying hundreds or thousands of probabilities can be cumbersome.
- Hence, the Markov assumption a.k.a. Markov property:
  - **future is independent of the past given the present**.
- In LM context: we can look at only the last k words.
- It is a simplification, but it produces good results anyway.

Language modeling task definition

Markov assumption

- Multiplying hundreds or thousands of probabilities can be cumbersome.
- Hence, the Markov assumption a.k.a. Markov property:
  - **future is independent of the past given the present**.
- In LM context: we can look at only the last k words.
- It is a simplification, but it produces good results anyway.

Markov assumption

- Multiplying hundreds or thousands of probabilities can be cumbersome.
- Hence, the Markov assumption a.k.a. Markov property:
  - **future is independent of the past given the present**.
- In LM context: we can look at only the last k words.
- It is a simplification, but it produces good results anyway.

Language modeling task definition

Markov assumption

- Multiplying hundreds or thousands of probabilities can be cumbersome.
- Hence, the Markov assumption a.k.a. Markov property:
  - **future is independent of the past given the present**.
- In LM context: we can look at only the last k words.
- It is a simplification, but it produces good results anyway.

Language modeling task definition

- Language modeling is widely used in NLP applications (machine translation, chat-bots, summarization...).
- LMs are evaluated by perplexity (how surprised is the model by test word sequences, the lower the better).
- For a test corpus of n word tokens:

$$probs = \sum_{i=1}^{n} \log_2 LM(w_i | w_{1:i-1})$$
$$perplexity = 2^{-\frac{1}{probs}} \tag{2}$$

- Perplexity is the exponentiated average negative log-likelihood per token.

Language modeling task definition

- Language modeling is widely used in NLP applications (machine translation, chat-bots, summarization...).
- LMs are evaluated by perplexity (how surprised is the model by test word sequences, the lower the better).
- For a test corpus of n word tokens:

$$probs = \sum_{i=1}^{n} \log_2 LM(w_i | w_{1:i-1}) \qquad (2)$$
$$perplexity = 2^{-\frac{1}{probs}}$$

- Perplexity is the exponentiated average negative log-likelihood per token.

Language modeling task definition

- Language modeling is widely used in NLP applications (machine translation, chat-bots, summarization...).
- LMs are evaluated by perplexity (how surprised is the model by test word sequences, the lower the better).
- For a test corpus of n word tokens:

$$probs = \sum_{i=1}^{n} \log_2 LM(w_i | w_{1:i-1}) \qquad (2)$$
$$perplexity = 2^{-\frac{1}{probs}}$$

- Perplexity is the exponentiated average negative log-likelihood per token.

Contents

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Toy algorithms

Random LM

- Extract vocabulary V from the training corpus.
- Always predict a random word from V.
- Uniform probability for all words: $\frac{1}{|V|}$

Frequency-based LM

- Extract frequency vocabulary V from the training corpus of size n.
- Predict a random word from V with the probabilities proportional to word frequencies.
- Frequent words are predicted more often.
- Probability for word w: $\frac{\#(w)}{n}$

Probabilities are extracted from corpus counts.

Traditional approaches to LM

Take context into account: Markov n-gram models

1. Take a large enough corpus;
2. count all word sequences of length k;
3. use maximum likelihood estimate for each word m:
   $\hat{P}\left((w_{i+1} = m) \,|\, w_{i-k:i}\right) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$
4. where $\#$ are corpus counts.
5. Et voila! You have probabilities for all seen words given previous sequences, for example:
   $\hat{P}\left((w_4 = jumps) \,|\, [the, quick, brown, fox]\right) = 0.6$
6. .. because your corpus had 10 occurrences of '*the quick brown fox*', and in 6 cases it was followed by '*jumps*', while in another 4 cases by '*barks*'.

Traditional approaches to LM

Take context into account: Markov n-gram models

1. Take a large enough corpus;
2. count all word sequences of length k;
3. use maximum likelihood estimate for each word m:
   $\hat{P}\left((w_{i+1} = m) \,|\, w_{i-k:i}\right) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$
4. where $\#$ are corpus counts.
5. Et voila! You have probabilities for all seen words given previous sequences, for example:
   $\hat{P}\left((w_4 = jumps) \,|\, [the, quick, brown, fox]\right) = 0.6$
6. .. because your corpus had 10 occurrences of '*the quick brown fox*', and in 6 cases it was followed by '*jumps*', while in another 4 cases by '*barks*'.

Traditional approaches to LM

Take context into account: Markov n-gram models

1. Take a large enough corpus;
2. count all word sequences of length k;
3. use maximum likelihood estimate for each word m:
   $\hat{P}\left((w_{i+1} = m) \,|\, w_{i-k:i}\right) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$
4. where $\#$ are corpus counts.
5. Et voila! You have probabilities for all seen words given previous sequences, for example:
   $\hat{P}\left((w_4 = jumps) \,|\, [the, quick, brown, fox]\right) = 0.6$
6. .. because your corpus had 10 occurrences of '*the quick brown fox*', and in 6 cases it was followed by '*jumps*', while in another 4 cases by '*barks*'.

Traditional approaches to LM

Take context into account: Markov n-gram models

1. Take a large enough corpus;
2. count all word sequences of length k;
3. use maximum likelihood estimate for each word m:
   $\hat{P}\left((w_{i+1} = m) \,|\, w_{i-k:i}\right) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$
4. where $\#$ are corpus counts.
5. Et voila! You have probabilities for all seen words given previous sequences, for example:
   $\hat{P}\left((w_4 = jumps) \,|\, [the, quick, brown, fox]\right) = 0.6$
6. .. because your corpus had 10 occurrences of '*the quick brown fox*', and in 6 cases it was followed by '*jumps*', while in another 4 cases by '*barks*'.

Traditional approaches to LM

Take context into account: Markov n-gram models

1. Take a large enough corpus;
2. count all word sequences of length k;
3. use maximum likelihood estimate for each word m:
   $\hat{P}\left((w_{i+1} = m) \,|\, w_{i-k:i}\right) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$
4. where $\#$ are corpus counts.
5. Et voila! You have probabilities for all seen words given previous sequences, for example:
   $\hat{P}\left((w_4 = jumps) \,|\, [the, quick, brown, fox]\right) = 0.6$
6. .. because your corpus had 10 occurrences of '*the quick brown fox*', and in 6 cases it was followed by '*jumps*', while in another 4 cases by '*barks*'.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Many shortcomings

- Sequence not seen in the training data? $\hat{P} = 0$
- There are ways to deal with unseen events...
  - but they are tricky...
  - ...and do not scale well to larger n-grams.
- Unseen events become more frequent as one increases k;
- number of possible word combinations is $|V|^k$;
- for the vocabulary of 10 000 words and 5-grams: 10000^5.
- Number of parameters raises polynomially when increasing $|V|$ and exponentially when increasing k.
- Model size quickly goes insane.

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

`git clone https://github.com/akutuzov/nlp_lm.git`

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

`git clone https://github.com/akutuzov/nlp_lm.git`

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

`git clone https://github.com/akutuzov/nlp_lm.git`

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

`git clone https://github.com/akutuzov/nlp_lm.git`

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

`git clone https://github.com/akutuzov/nlp_lm.git`

Traditional approaches to LM

Lack of generalization power

Words are discrete features:

- Representation power not shared between similar words
- we saw '*fox eats*' and '*dog eats*' 1000 times each
- we never saw '*wolf eats*'
- the probability of '*wolf eats*' will still be 0.

However, Markov models still work surprisingly well.

Let's experiment a bit with the code.

```
git clone https://github.com/akutuzov/nlp_lm.git
```

Question:

why using frequencies does not result in better perplexity?

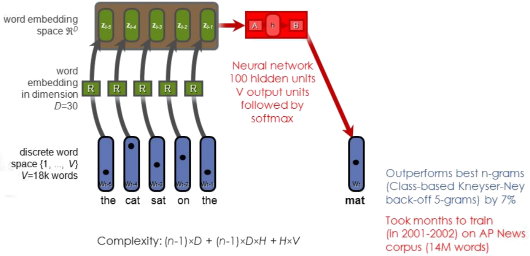
Answer:

Because of Zipf law

Contents

Neural language modeling

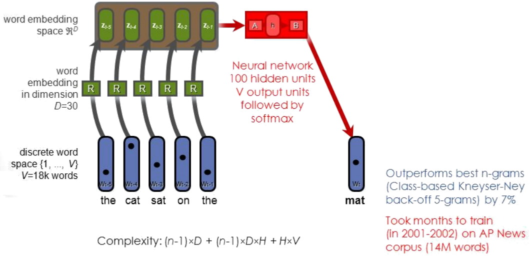
- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", *ICASSP* 2002]

Neural language modeling

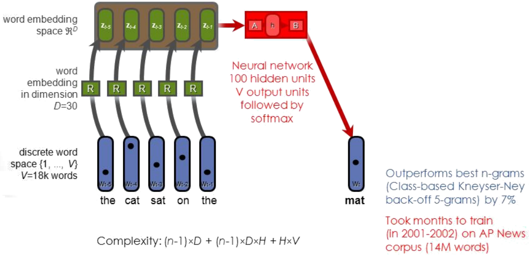
- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", ICASSP 2002]

Neural language modeling

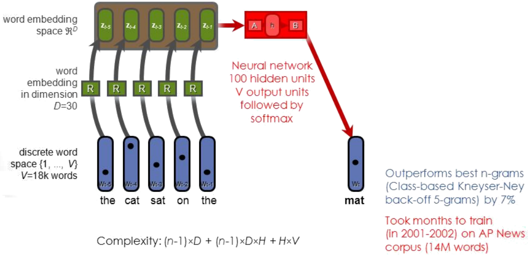
- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", ICASSP 2002]

Neural language modeling

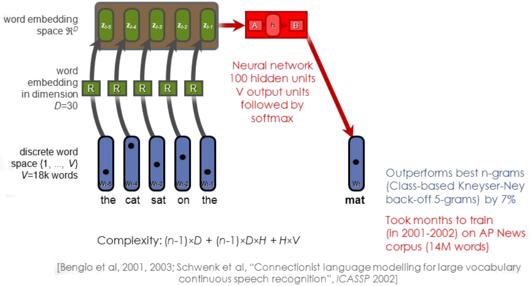
- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", ICASSP 2002]

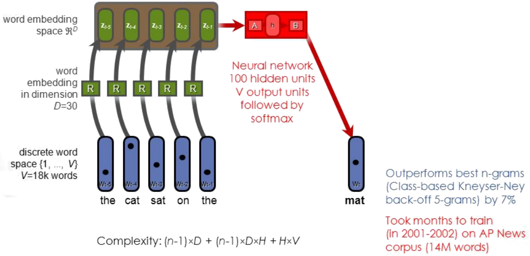
Neural language modeling

- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



Neural language modeling

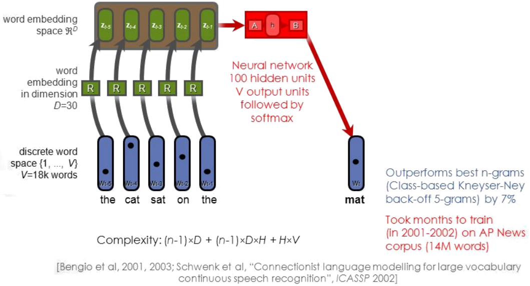
- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", ICASSP 2002]

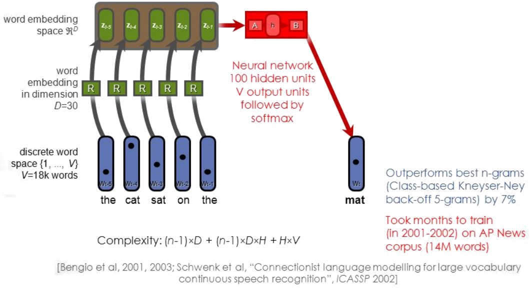
Neural language modeling

- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.

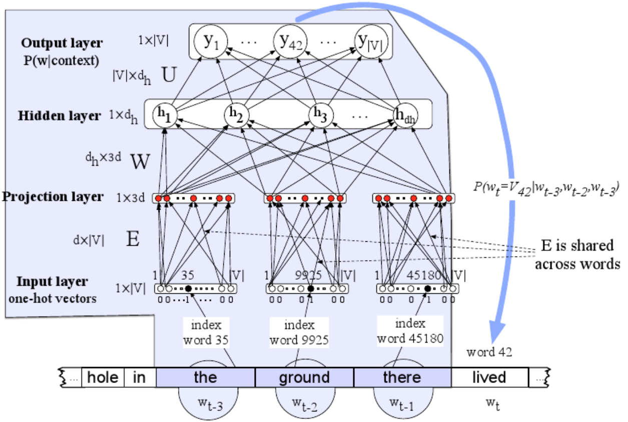


Neural language modeling

- Neural LM model proposed in [Bengio et al., 2003]:
- concatenate learned embeddings of the previous k words;
- this concatenation is fed into a feed-forward neural network...
- ...with hidden layers and non-linearities;
- cross-entropy loss, the next words as the gold predictions.
- Output probability distribution over possible next words across the whole vocabulary V (using softmax and the second embedding matrix).
- Input and output vocabularies can be different.



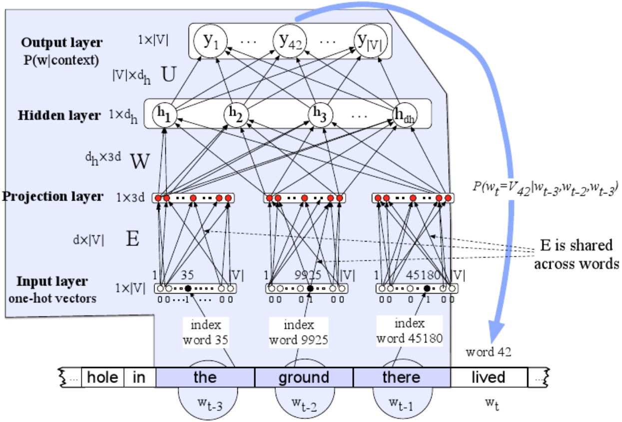
Neural language modeling



Feed-forward neural LM moving through a text

(from Jurafsky and Martin, 2018)

Modern neural LMs are mostly recurrent (gated RNNs like LSTM or GRU).



Feed-forward neural LM moving through a text

(from Jurafsky and Martin, 2018)

Modern neural LMs are mostly recurrent (gated RNNs like LSTM or GRU).

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Neural language modeling

Benefits

- Outperforms traditional LMs in perplexity.
- Scales well: higher k leads to linear increase in the parameters number...
- ...in traditional LMs it was exponential.
- Words in different positions share statistical strength.
- Generalizations to unseen data: similar words get similar representations in the embedding and the output layers:
  - '*fox eats*': seen 1000 times; '*dog eats*': seen 1000 times; '*wolf eats*': seen 0 times; $\hat{P}([wolf, eats]) \gg 0$, because '*wolf*' is similar to '*fox*' and '*dog*'.
- Can easily add more hidden layers.

Shortcomings

- Expensive softmax over V in the output layer.
- Increasing the output $|V|$ can significantly slow down the network (already slower than traditional models).
- There are ways to deal with this.

Neural language modeling

Shortcomings

- Expensive softmax over V in the output layer.
- Increasing the output $|V|$ can significantly slow down the network (already slower than traditional models).
- There are ways to deal with this.

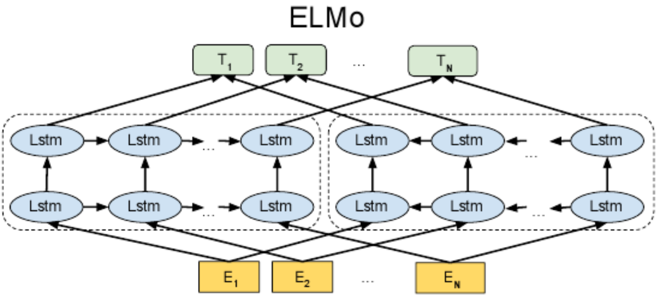
Shortcomings

- Expensive softmax over V in the output layer.
- Increasing the output $|V|$ can significantly slow down the network (already slower than traditional models).
- There are ways to deal with this.

Current state: pre-trained language models

Language models can provide contextualized word embeddings, with different representations in different contexts.

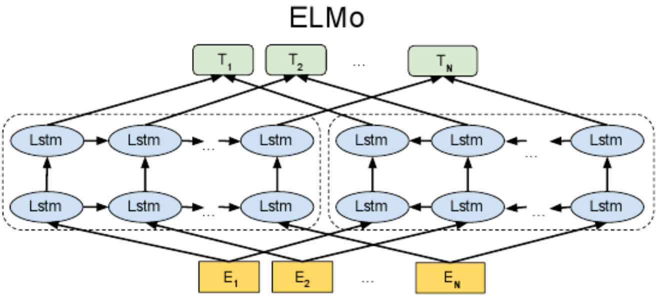
- Embeddings from Language MOdels (ELMo) use LSTMs [Peters et al., 2018]
- Bidirectional Encoder Representations from Transformer (BERT) use bidirectional transformers [Devlin et al., 2019]



Current state: pre-trained language models

Language models can provide contextualized word embeddings, with different representations in different contexts.

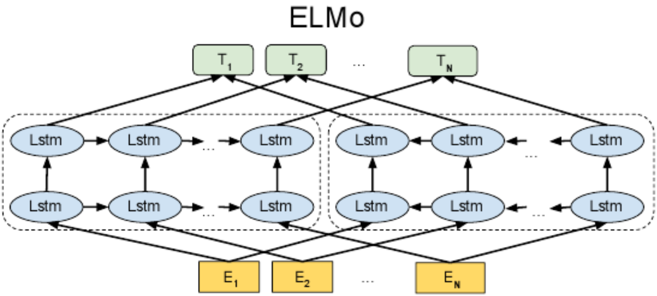
- Embeddings from Language MOdels (ELMo) use LSTMs [Peters et al., 2018]
- Bidirectional Encoder Representations from Transformer (BERT) use bidirectional transformers [Devlin et al., 2019]



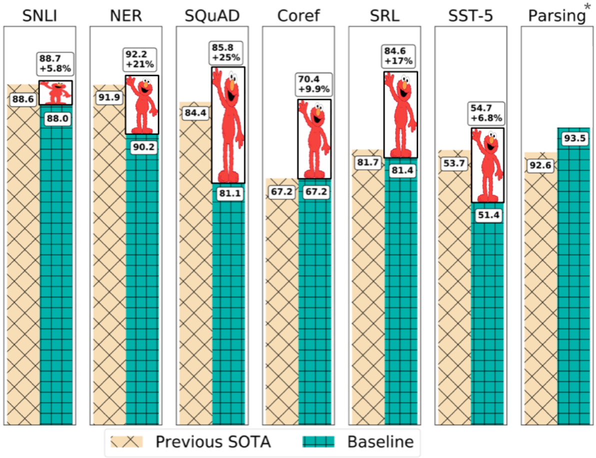
Current state: pre-trained language models

Language models can provide contextualized word embeddings, with different representations in different contexts.

- Embeddings from Language MOdels (ELMo) use LSTMs [Peters et al., 2018]
- Bidirectional Encoder Representations from Transformer (BERT) use bidirectional transformers [Devlin et al., 2019]



ELMo seem to improve any NLP task you apply them for:

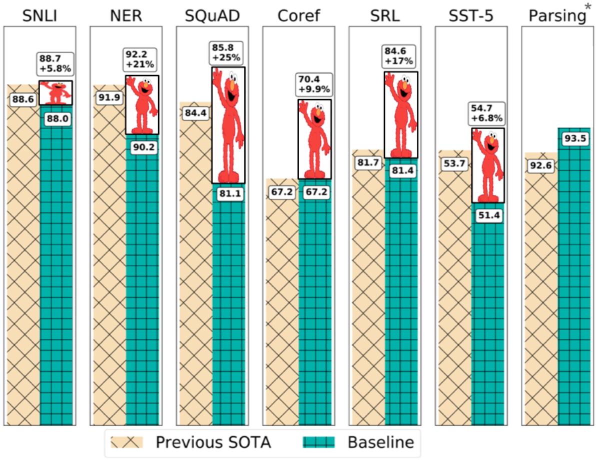


*Kitaev and Klein, ACL 2018 (see also Joshi et al., ACL 2018)

'*ImageNet for NLP*' (Sebastian Ruder)

Current state: pre-trained language models

ELMo seem to improve any NLP task you apply them for:



*Kitaev and Klein, ACL 2018 (see also Joshi et al., ACL 2018)

'*ImageNet for NLP*' (Sebastian Ruder)

Current state: pre-trained language models

Modes of usage

1. 'as is': contextualized representations are fed into the overarching architecture like the old-school 'static' embeddings;
2. the whole model is fine-tuned on target task data.

Layers of ELMo reflect language tiers

- word embedding layer: morphology;
- the first LSTM layer: syntax;
- the second LSTM layer: semantics (including word senses).

More info

- https://allennlp.org/elmo
- https://github.com/allenai/bilm-tf
- https://github.com/google-research/bert

Current state: pre-trained language models

Modes of usage

1. 'as is': contextualized representations are fed into the overarching architecture like the old-school 'static' embeddings;
2. the whole model is fine-tuned on target task data.

Layers of ELMo reflect language tiers

- word embedding layer: morphology;
- the first LSTM layer: syntax;
- the second LSTM layer: semantics (including word senses).

More info

- https://allennlp.org/elmo
- https://github.com/allenai/bilm-tf
- https://github.com/google-research/bert

Current state: pre-trained language models

Modes of usage

1. 'as is': contextualized representations are fed into the overarching architecture like the old-school 'static' embeddings;
2. the whole model is fine-tuned on target task data.

Layers of ELMo reflect language tiers

- word embedding layer: morphology;
- the first LSTM layer: syntax;
- the second LSTM layer: semantics (including word senses).

More info

- https://allennlp.org/elmo
- https://github.com/allenai/bilm-tf
- https://github.com/google-research/bert

Current state: pre-trained language models

Modes of usage

1. ‘as is’: contextualized representations are fed into the overarching architecture like the old-school ‘static’ embeddings;
2. the whole model is fine-tuned on target task data.

Layers of ELMo reflect language tiers

- word embedding layer: morphology;
- the first LSTM layer: syntax;
- the second LSTM layer: semantics (including word senses).

More info

- `https://allennlp.org/elmo`
- `https://github.com/allenai/bilm-tf`
- `https://github.com/google-research/bert`

References I

Bengio, Y., Ducharme, R., and Vincent, P. (2003).
A neural probabilistic language model.
Journal of Machine Learning Research, 3:1137–1155.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019).
BERT: Pre-training of deep bidirectional transformers for language understanding.
In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018).
Deep contextualized word representations.
In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237. Association for Computational Linguistics.