Distributed representations and RNNs

Computational Semantics 2021

Adam Ek

Plan

- Part 1: Distributed representations (25 min)
- Part 2: Language modeling (20 min)
- Part 3: Break (15min)
- Part 4: PyTorch example: Part-of-Speech tagging (45 min)

The great switcheroo

- Previously we learned about *count* vectors and composing vectors

The great switcheroo

- Previously we learned about *count* vectors and composing vectors
- But count vectors have several problems:
  - Sparsity:
    - few degrees of similarity
    - language have a Zipfian-distribution

The great switcheroo

- Previously we learned about *count* vectors and composing vectors
- But count vectors have several problems:
  - Sparsity:
    - few degrees of similarity
    - language have a Zipfian-distribution
  - Post-processing necessary:
    - Weighting: TF-IDF, PPMI, ...
    - Dimensionality reduction: SVD

Distributed representations

- Instead of counting the context (distributional) let's predict the context!

Distributed representations

- Instead of counting the context (distributional) let's predict the context!
- We initialize word representations (embeddings) as *random* vectors of dimension D

Distributed representations

- Instead of counting the context (distributional) let's predict the context!
- We initialize word representations (embeddings) as *random* vectors of dimension D
- We train a binary classifier on a task T: does w_a occur in the context of w_b?

Distributed representations

- Instead of counting the context (distributional) let's predict the context!
- We initialize word representations (embeddings) as *random* vectors of dimension D
- We train a binary classifier on a task T: does w_a occur in the context of w_b?
- The thing we care about is the *embeddings*

Distributed representations

- Instead of counting the context (distributional) let's predict the context!
- We initialize word representations (embeddings) as *random* vectors of dimension D
- We train a binary classifier on a task T: does w_a occur in the context of w_b?
- The thing we care about is the *embeddings*
- That is, can we use the embeddings to predict other things than the context (for example word similarity)

Learning distributed representations

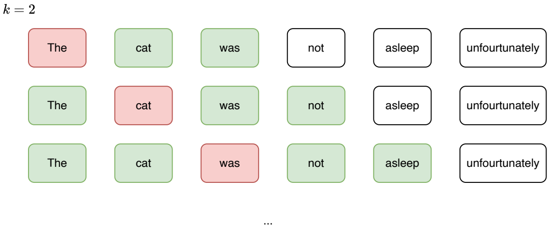
- The task we try to solve is: does w_a occur in the context of w_b?

Learning distributed representations

- The task we try to solve is: does w_a occur in the context of w_b?
- Like with count vectors we must define what a context *is*, which we do with a *hyperparameter* k (aka. window size)

Learning distributed representations

- The task we try to solve is: does w_a occur in the context of w_b?
- Like with count vectors we must define what a context *is*, which we do with a *hyperparameter* k (aka. window size)
- In this setup we have a center word w_c and some context words w_o (outside words)

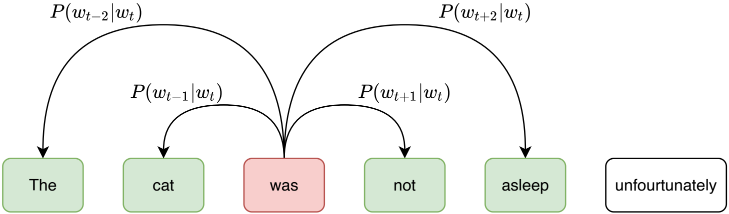


Word2Vec

- We'll consider two methods, SkipGram and Continuous Bag-of-words (CBOW)
- aka. Word2Vec

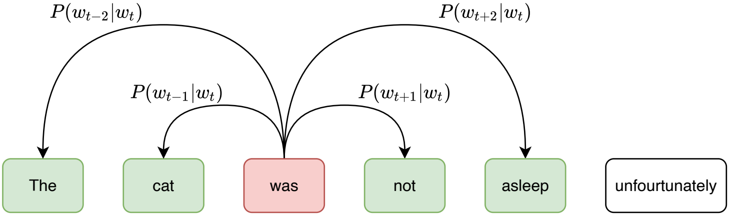
Skip-Gram Model

- We select a center word t and predict the context words



Skip-Gram Model

- We select a center word t and predict the context words

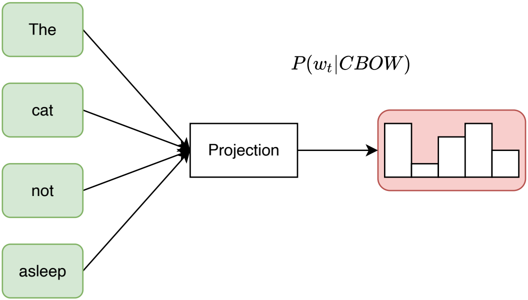


Predicting the context words

$$P(w_{t-1}|w_t) = \frac{exp(w_{t-1}^T w_t)}{\sum_{w \in V} exp(w^T w_t)}$$

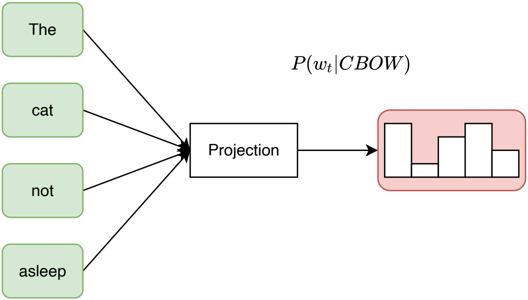
CBOW Model

- Based on a "bag of words", predict the center word



CBOW Model

- Based on a "bag of words", predict the center word



Predicting the context words

$$P(w_t|CBOW) = \frac{exp(w_t)}{\sum_{w \in V} exp(w)}$$

A neural network for Word2Vec

- To apply either of these models we'll use *neural networks*
- In both methods, the setup is simple:

A neural network for Word2Vec

- To apply either of these models we'll use *neural networks*
- In both methods, the setup is simple:
  - Perform non-linear transformations with weight matrices
  - Calculate the probability of the target word

A neural network for Word2Vec

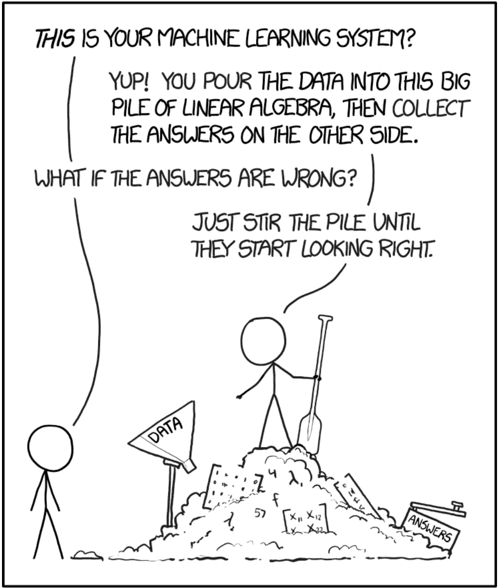
- To apply either of these models we'll use *neural networks*
- In both methods, the setup is simple:
  - Perform non-linear transformations with weight matrices
  - Calculate the probability of the target word
  - Calculate loss with Negative Log Likelihood
  - Adjust the weight matrices with gradient descent such that the accuracy goes up (that is, minimize the negative log likelihood)

What is a weight matrix?

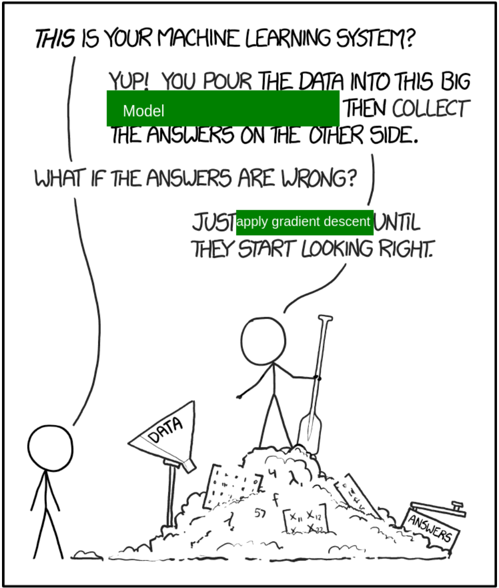
$$W^{n,d} = \begin{bmatrix} x_0^0 & \dots & x_n^0 \\ & \vdots & \\ x_0^d & \dots & x_n^k \end{bmatrix}$$

- takes an input of size n, and produces and output of size d

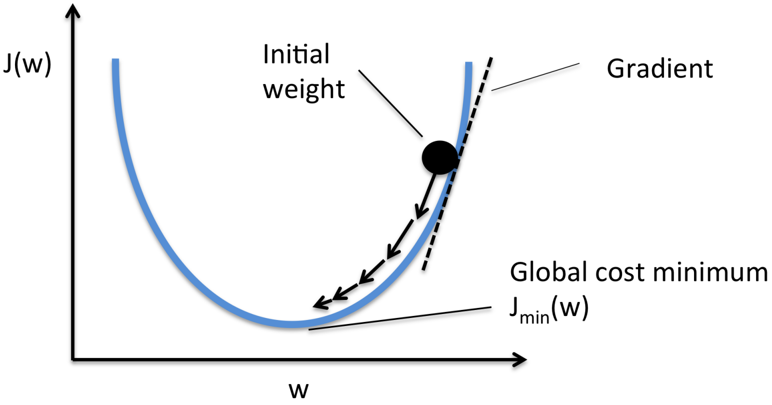
Machine learning expert visualization (https://xkcd.com/1838/)



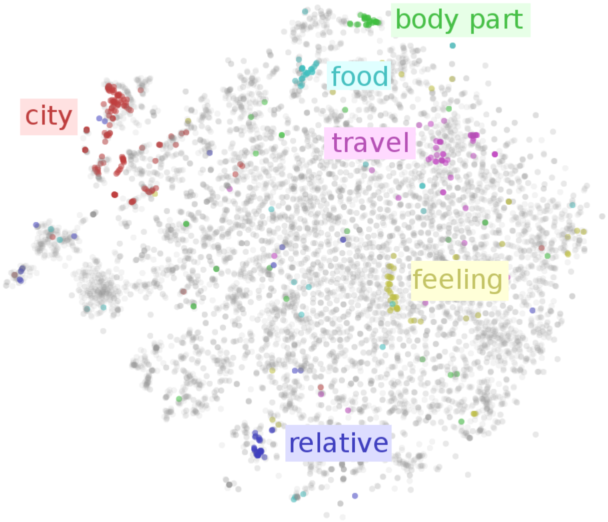
Machine learning expert visualization (https://xkcd.com/1838/)



A quick look at Stochastic Gradient Descent



Semantic Space



The big WOW: Bilingual Dictionary Induction

- If we have embeddings in language X and a language Z, we can find word translations!

The big WOW: Bilingual Dictionary Induction

- If we have embeddings in language X and a language Z, we can find word translations!

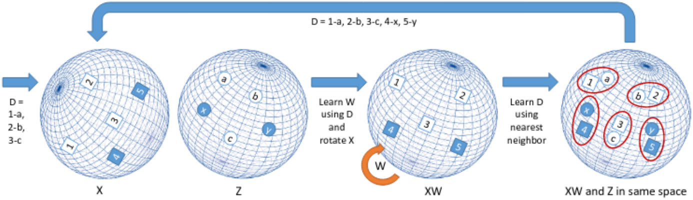


Figure 1: A general schema of the proposed self-learning framework. Previous works learn a mapping W based on the seed dictionary D, which is then used to learn the full dictionary. In our proposal we use the new dictionary to learn a new mapping, iterating until convergence.

Bilingual Dictionary Induction

- "Learning bilingual word embeddings with (almost) no bilingual data" (Artexe, et. al (2017))
- We formalize our objective as the euclidean distance between the items in our seed dictionary:

$$W^* = \arg\min_W \sum_i \sum_j D_{ij} ||X_{i*}W - Z_{j*}||^2$$

Bilingual Dictionary Induction

- "Learning bilingual word embeddings with (almost) no bilingual data" (Artexe, et. al (2017))
- We formalize our objective as the euclidean distance between the items in our seed dictionary:

$$W^* = \arg\min_W \sum_i \sum_j D_{ij} ||X_{i*}W - Z_{j*}||^2$$

	English-Italian			English-German			English-Finnish		
	5,000	**25**	**num.**	**5,000**	**25**	**num.**	**5,000**	**25**	**num.**
Mikolov et al. (2013a)	34.93	0.00	0.00	35.00	0.00	0.07	25.91	0.00	0.00
Xing et al. (2015)	36.87	0.00	0.13	41.27	0.07	0.53	28.23	0.07	0.56
Zhang et al. (2016)	36.73	0.07	0.27	40.80	0.13	0.87	28.16	0.14	0.42
Artetxe et al. (2016)	39.27	0.07	0.40	**41.87**	0.13	0.73	**30.62**	0.21	0.77
Our method	**39.67**	**37.27**	**39.40**	40.87	**39.60**	**40.27**	28.72	**28.16**	**26.47**

Table 1: Accuracy (%) on bilingual lexicon induction for different seed dictionaries

Language modeling (1)

- Predict which word from a vocabulary that comes next given a piece of text
- "the model predicts `<BLANK>`" but *what* is `<BLANK>`?

Language modeling (1)

- Predict which word from a vocabulary that comes next given a piece of text
- "the model predicts `<BLANK>`" but *what* is `<BLANK>`?

Formal definition (1)

$P(x_{t+1}|x_t, ..., x_1)$

Language modeling (1)

- Predict which word from a vocabulary that comes next given a piece of text
- "the model predicts `<BLANK>`" but *what* is `<BLANK>`?
  - the next word
  - a word

Formal definition (1)

$P(x_{t+1}|x_t, ..., x_1)$

Language modeling (1)

- Predict which word from a vocabulary that comes next given a piece of text
- "the model predicts `<BLANK>`" but *what* is `<BLANK>`?
  - the next word
  - a word
  - something meaningful

Formal definition (1)

$P(x_{t+1}|x_t, ..., x_1)$

Language modeling (1)

- Predict which word from a vocabulary that comes next given a piece of text
- "the model predicts `<BLANK>`" but *what* is `<BLANK>`?
  - the next word
  - a word
  - something meaningful
  - a grammatically plausible word
  - a semantically plausible word

Formal definition (1)

$P(x_{t+1}|x_t, ..., x_1)$

Language modeling (2)

- But what is x_{t+1}? And $x_t, ..., x_0$?

Language modeling (2)

- But what is x_{t+1}? And $x_t, ..., x_0$?
- "Items" in our vocabulary; a collection of strings extracted from our training corpus

Language modeling (2)

- But what is x_{t+1}? And $x_t, ..., x_0$?
- "Items" in our vocabulary; a collection of strings extracted from our training corpus
- We build the vocabulary by *tokenizing* our text, but *how* we tokenize is up to us

Language modeling (2)

- But what is x_{t+1}? And $x_t, ..., x_0$?
- "Items" in our vocabulary; a collection of strings extracted from our training corpus
- We build the vocabulary by *tokenizing* our text, but *how* we tokenize is up to us
- Thus, what we do in language modeling is...

Formal definition (2)

$P(w_1, ..., w_T) = \prod_1^T P(w_t | w_{t-1}, ..., w_1)$

Language modeling (2)

- But what is x_{t+1}? And $x_t, ..., x_0$?
- "Items" in our vocabulary; a collection of strings extracted from our training corpus
- We build the vocabulary by *tokenizing* our text, but *how* we tokenize is up to us
- Thus, what we do in language modeling is...
- assign probability (and consequently a representation) to a sequence of symbols

Formal definition (2)

$P(w_1, ..., w_T) = \prod_1^T P(w_t | w_{t-1}, ..., w_1)$

Applications

Ok, cool! What is the use in that?

Applications

Ok, cool! What is the use in that?

- Auto-complete (e.g. google search)

Applications

Ok, cool! What is the use in that?

- Auto-complete (e.g. google search)
- Text generation

Applications

Ok, cool! What is the use in that?

- Auto-complete (e.g. google search)
- Text generation
- Translation

Applications

Ok, cool! What is the use in that?

- Auto-complete (e.g. google search)
- Text generation
- Translation
- Actually, as it turn out: 99% (*handwavy number*) of NLP benefit from language modeling (Nikolai will tell you more about this in the next lecture)

HOW!? OH GOD HOW!? Part 1

Let us consider some history (oh god, no...)

- First statistical (language) models were based on n-grams and co-occurrence
- They worked "ok" but suffered some major drawbacks:

HOW!? OH GOD HOW!? Part 1

Let us consider some history (oh god, no...)

- First statistical (language) models were based on n-grams and co-occurrence
- They worked "ok" but suffered some major drawbacks:
    - Context limited to n tokens

HOW!? OH GOD HOW!? Part 1

Let us consider some history (oh god, no...)

- First statistical (language) models were based on n-grams and co-occurrence
- They worked "ok" but suffered some major drawbacks:
  - Context limited to n tokens
  - Rare words

HOW!? OH GOD HOW!? Part 2

Then, in 2015-ish neural networks entered the scene (YAY!) and has alleviated many of the issues:

- Neural Networks use RNNs to process sequences, which can model an arbitrarily long contexts

HOW!? OH GOD HOW!? Part 2

Then, in 2015-ish neural networks entered the scene (YAY!) and has alleviated many of the issues:

- Neural Networks use RNNs to process sequences, which can model an arbitrarily long contexts
  - Not really, RNNs still have trouble with long dependencies

HOW!? OH GOD HOW!? Part 2

Then, in 2015-ish neural networks entered the scene (YAY!) and has alleviated many of the issues:

- Neural Networks use RNNs to process sequences, which can model an arbitrarily long contexts
  - Not really, RNNs still have trouble with long dependencies
- Words are represented as distributed *vectors*
- which allows a model to compute *similarities* which helps with rare words (and frequent words for that matter)

A recipe for LMs

- Encode the tokens in your sentence as distributed representations

A recipe for LMs

- Encode the tokens in your sentence as distributed representations
- Can we use word2vec for this? ...

A recipe for LMs

- Encode the tokens in your sentence as distributed representations
- Can we use word2vec for this? ...
- Feed the encoded tokens to an RNN

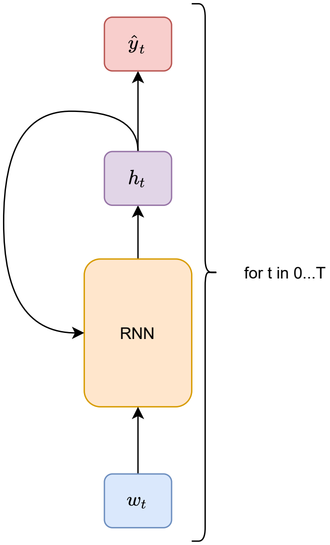
A recipe for LMs

- Encode the tokens in your sentence as distributed representations
- Can we use word2vec for this? ...
- Feed the encoded tokens to an RNN
    - Predict the next token

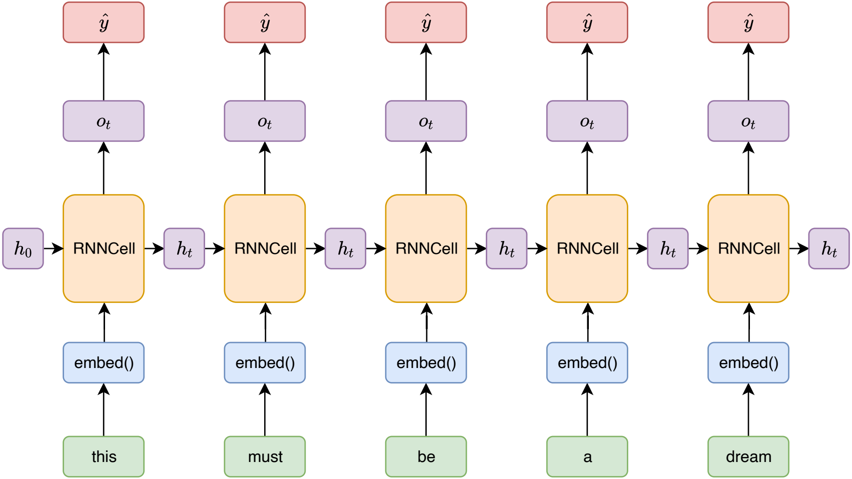
A recipe for LMs

- Encode the tokens in your sentence as distributed representations
- Can we use word2vec for this? ...
- Feed the encoded tokens to an RNN
  - Predict the next token
- Continue until you've reached the end of the sentence

RNN



RNN un-rolled



Question time!

- What does the final h_t represent?

Question time!

- What does the final h_t represent?

  $\rightarrow$ A "summary" representation of the sentence

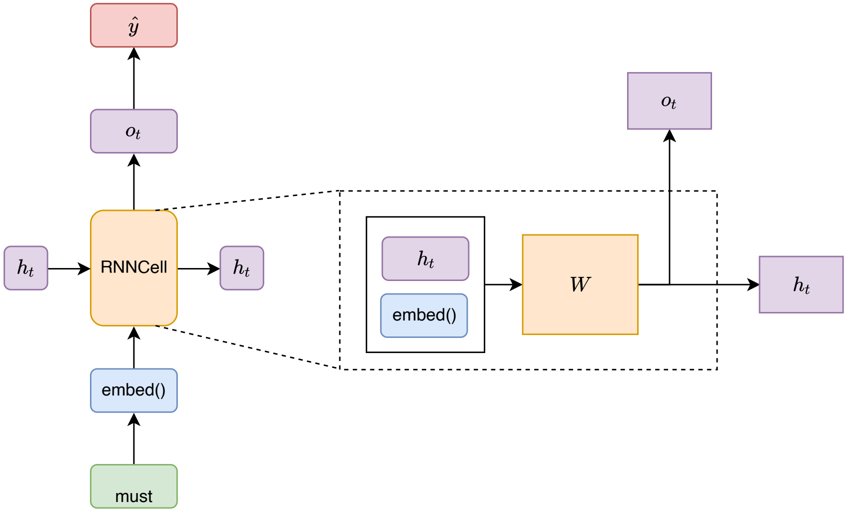
Question time!

- What does the final h_t represent?

  $\rightarrow$ A "summary" representation of the sentence

  $\rightarrow$ It's a summary because this is the accumulated hidden state from the entire input sequence

The RNN Cell



Question time!

- How do we go from o_t to $\hat{y}$

Question time!

- How do we go from o_t to $\hat{y}$
  $\rightarrow h_t W + b$ where $W \in \mathbb{R}^{d(h_t),|V|}$

Predicting the word

- The target word is predicted by $\hat{y}' = softmax(\hat{y})$
- $softmax(\hat{y}_i) = \frac{e^{y_i}}{\sum_{T=0}^{K} e^{y_t}}$ for i in $1 \ldots K$

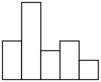
Predicting the word

- The target word is predicted by $\hat{y}' = softmax(\hat{y})$
- $softmax(\hat{y}_i) = \frac{e^{y_i}}{\sum_{T=0}^{K} e^{y_t}}$ for i in $1 \ldots K$

softmax(

The	12.1
nay	24.3
...	...
not	13.4
final	3.2

) =



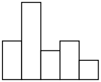
Predicting the word

- The target word is predicted by $\hat{y}' = softmax(\hat{y})$
- $softmax(\hat{y}_i) = \frac{e^{y_i}}{\sum_{T=0}^{K} e^{y_t}}$ for i in $1...K$

softmax(

The	12.1
nay	24.3
...	...
not	13.4
final	3.2

) =



- This yields a vector $\hat{y}'$ that contain the probabilities for each item in our vocabulary

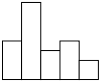
Predicting the word

- The target word is predicted by $\hat{y}' = softmax(\hat{y})$
- $softmax(\hat{y}_i) = \frac{e^{y_i}}{\sum_{T=0}^{K} e^{y_t}}$ for i in $1 \ldots K$

softmax(

The	12.1
nay	24.3
...	...
not	13.4
final	3.2

) =



- This yields a vector $\hat{y}'$ that contain the probabilities for each item in our vocabulary
- Then we calculate the Negative Log Likelihood loss :) and stir the pile (with gradient descent)

Evaluation

- Intrinsic: Use the model on a specific intermediate task

Evaluation

- Intrinsic: Use the model on a specific intermediate task
  - For Word2Vec: Analogy tests, word similarity
  - For language models: Perplexity (does the model assign high probability to the test set?)

Evaluation

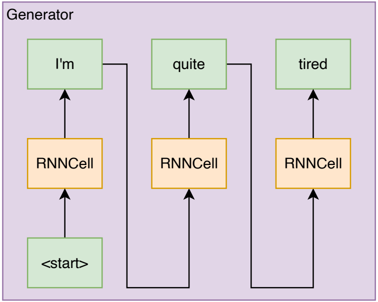
- Intrinsic: Use the model on a specific intermediate task
  - For Word2Vec: Analogy tests, word similarity
  - For language models: Perplexity (does the model assign high probability to the test set?)
- Extrinsic: Use the model in some application or downstream task

Evaluation

- Intrinsic: Use the model on a specific intermediate task
  - For Word2Vec: Analogy tests, word similarity
  - For language models: Perplexity (does the model assign high probability to the test set?)
- Extrinsic: Use the model in some application or downstream task
  - Plug the word representations obtained into another model (parsing/QA-model/...)

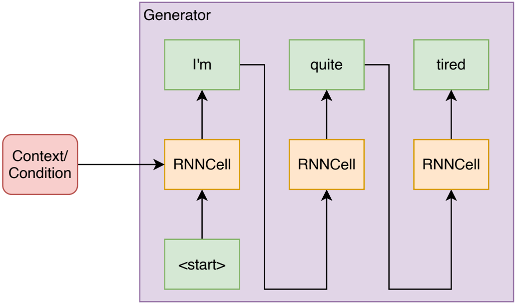
Natural Language Generation (NLG)

- We can use language models to *generate* language



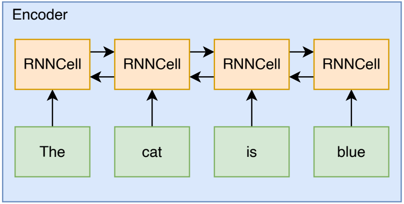
Conditioned Natural Language Generation (NLG)

- But generating based on nothing will most likely just give us nonsense...
- Language usually happens in some *context*, or as an response to *something*



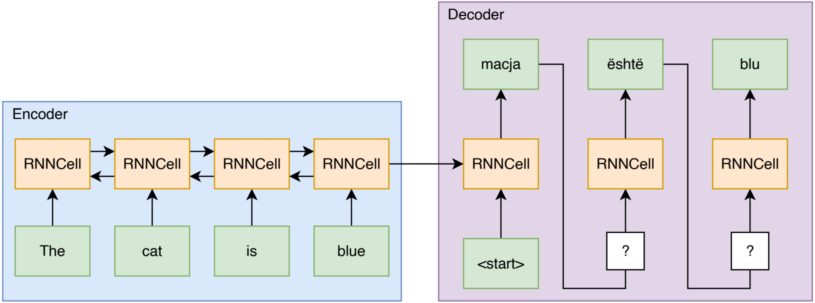
Machine translation (MT)

- We use a so-called *seq2seq*-model
- Encode a sentence from the source language



Machine translation (MT)

- We use a so-called *seq2seq*-model
- Encode a sentence from the source language
- *Generate* a sentence in the target language



Next up...

Questions?

A short look at the attention mechanism

- When using NLG or MT, the contextual information (etc...) are encoded only in the initial hidden state h_t
- This information will easily be forgotten after some time-steps
- We can use *attention* to alleviate this problem by looking back at the context at *every time-step*

Attention