Welcome to INFO216: Knowledge Graphs

Spring 2024

Andreas L Opdahl
<Andreas.Opdahl@uib.no>

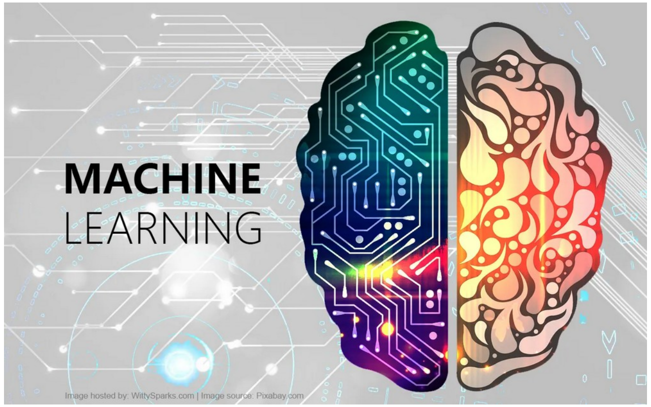
Session 11: Graph embeddings

- Themes:
  - *KGs and machine learning (ML)*
  - *what are embeddings?*
    - word embeddings
    - how to find and use them
    - other types of embeddings
  - *what are graph embeddings?*
    - how to find them...
    - ...and what to use them for

Readings

towards data science

- Resources in the wiki <http://wiki.uib.no/info216>:
  - Introduction to Machine Learning
  - Introduction to Word Embeddings
  - Introduction to Knowledge Graph Embeddings
- Supplementary (links in the wiki):
  - Mikolov et al's original word2vec paper
  - Bordes et al's original TransE paper
  - TorchKGE documentation (for the labs):
    - https://torchkge.readthedocs.io/en/latest/index.html



KGs and
Machine Learning (ML)

What are the connections?

- *Knowledge graphs are well matched with machine learning!*
- Preparing inputs to ML (varying origins, formats, modalities…)
  - also managing outputs from ML / DL
- Infusing world knowledge into ML / DL
  - language knowledge
  - common sense knowledge
  - world knowledge (domain and general), …
- More and more also a “native” ML technique

A micro-introduction to machine learning (ML)

- Sole purpose for us:
  - to be able to understand and use KG embeddings
- *Make computers do useful things based on examples (training data)*
  - *by using the examples to train a model*
- *Supervised learning:*
  - training materials comprise input-output value pairs as examples
- *Unsupervised learning:*
  - training materials comprise only input examples
- Several other variants: *semi-supervised, reinforcement learning*, …
- Learning KG (and other) embeddings is often *unsupervised*
  - but also many uses of *supervised* training

Train, evaluate, and test

- Training examples can be split in three:
  - *training data* are used to train the model
  - *validation data* are used to optimise hyper-parameters and monitor progress
  - *test data* are used only for final evaluation
  - 60%-20%-20% or 80%-10%-10% split is common
    - also minimum requirements for test examples
- *k-fold cross-validation:*
  - training and validation data are split in *k* folds
  - *k-1* folds are used for training, *1* for validation
  - repeated *k* times for each validation fold
  - finally, measures are averaged over the validation folds

Epochs and batches

- We can go through the training data many times
  - each go-through is an *epoch*
- We can go through the training examples in groups
  - each group is called a *batch*
- Each example creates a *loss*
  - numeric difference between the actual and the “correct” result
- So:
  - training consists of many epochs
  - each epoch consists of many batches
  - each batch consists of many training examples
  - each training example creates a loss
  - after each batch, model is updated to minimise future loss

Evaluation measures

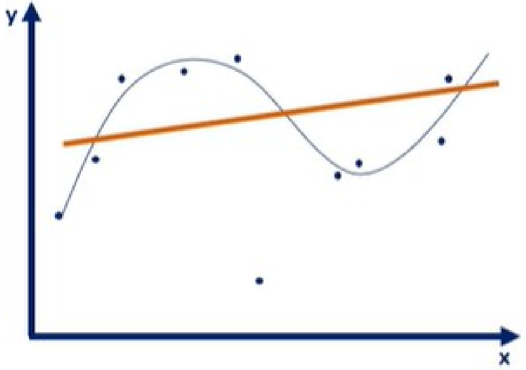
- Results without ranking:
  - *accuracy (A):* ratio of correct results
  - there are lots of others:
    - precision (P), recall (R), F1 = 2PR/(P+R), ...
- Ranked results:
  - *Hit@n:* number of correct results in the “top n”, e.g., Hit@10
  - *Mean Rank*: average rank of the correct results
  - *Mean Reciprocal Rank (MRR):* average inverse rank of the higest-ranked correct result for each query, example:
    - the “best” correct results for three queries have ranks 3, 1, 28
    - MRR = (1/3 + 1/1 + 1/28) / 3
- Other measures for other data types, e.g., time series data

Under- and overfitting

- Underfitting:
  - we have not trained for long enough, too few epochs
  - there is more to learn from the training data
  - high and decreasing loss
  - validation measures (like A) are still improving
- Overfitting:
  - we have trained for too long, too many epocs
  - the model has specialised on the training data
  - low and decreasing loss
  - validation measures (like A) have begun to worsen

Underfitting and overfitting

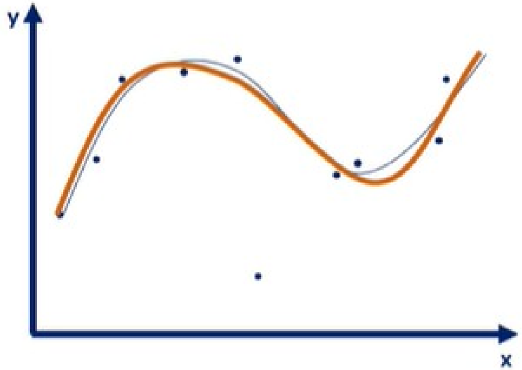
An **underfitted** model



Doesn't capture any logic

- High loss
- Low accuracy

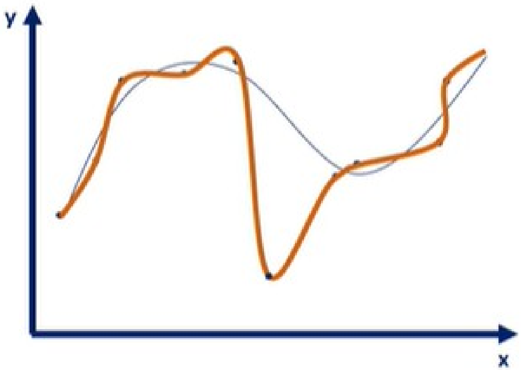
A **good** model



Captures the underlying logic of the dataset

- Low loss
- High accuracy

An **overfitted** model



Captures all the noise, thus "missed the point"

- Low loss
- Low accuracy

365√DataScience

https://365datascience.com/tutorials/machine-learning-tutorials/overfitting-underfitting/

What are embeddings?

How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *vectors* in a *latent semantic space*!

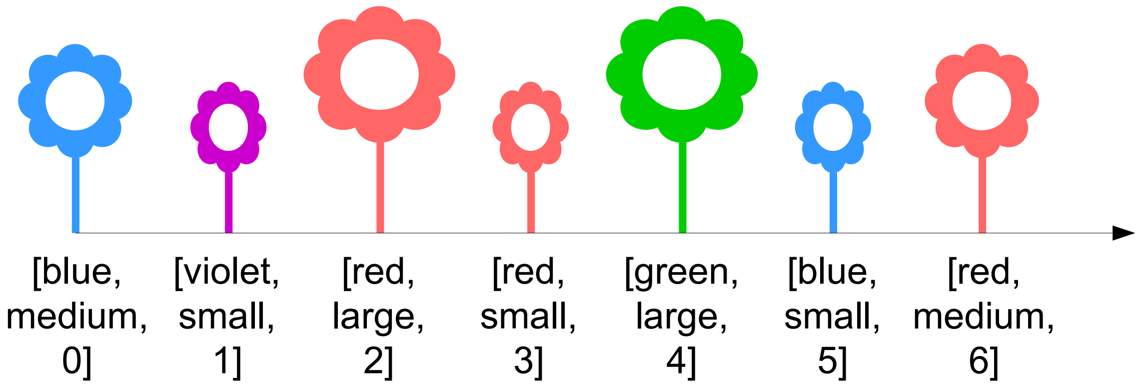
How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *vectors* in a *latent semantic space*!
- Example:
  - *FlowerWorld™*
  - *“Everything is a flower!”*
  - *a flower has three attributes:*
    - *colour*
    - *size*
    - *position*

How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *vectors* in a *latent semantic space*!
- Example:
  - *FlowerWorld™*
  - *"Everything is a flower!"*
  - *a flower has three attributes:*
    - *colour*
    - *size*
    - *position*

Everything in FlowerWorld™ can be uniquely described by its position along three dimensions!

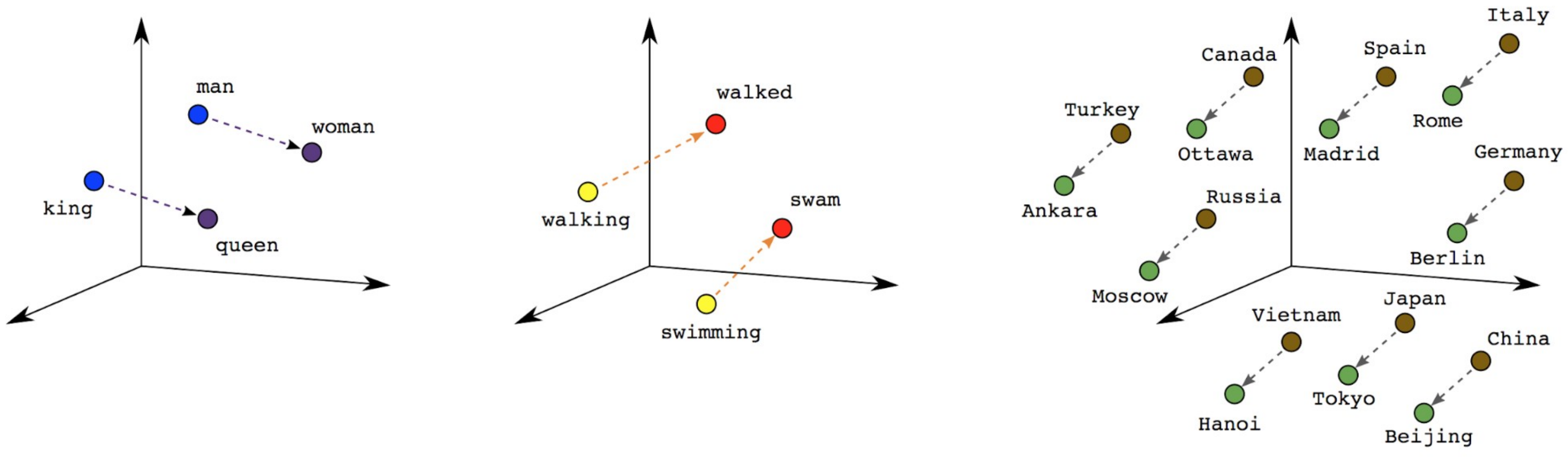


How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *vectors* in a *latent semantic space*!
- (Our conceptualisations of) Things in the "real world":
  - a bit more complex...
  - not fully describable by positions along dimensions
  - but perhaps we can describe them usefully by adding more dimensions?
  - but which dimensions to add?
    - use machine learning / neural networks to analyse large text corpora!

How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *vectors* in a *latent semantic space*!



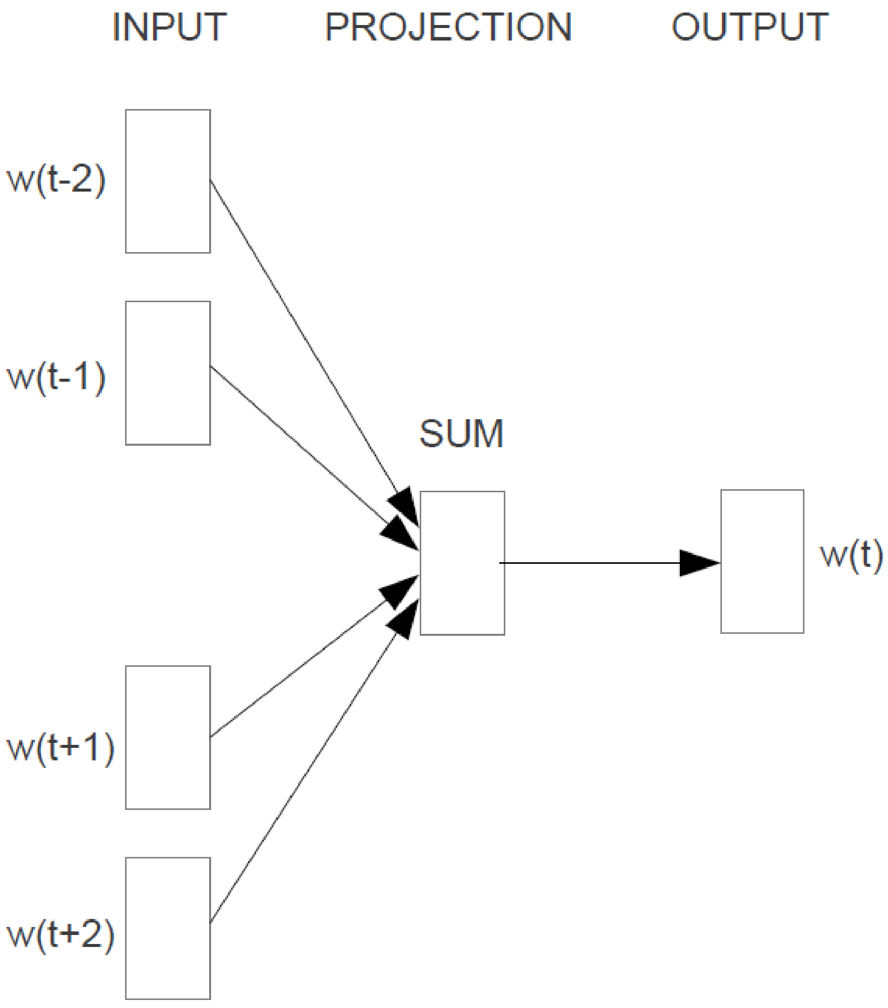
Male-Female · Verb Tense · Country-Capital

How can we represent the meaning of words?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *embeddings*, i.e., *vectors* in a *latent semantic space*!
  - [0.01 0.62 0.03 … 0.41]
  - similar words are close to one another
  - relative positions between words can be systematic
    - [Paris] – [France] + [Italy] ≈ [Rome]
  - distances between words can represent relations
    - [J. K. Rowling] + [influenced by] ≈ [J. R. R. Tolkien]
- Important use: as inputs to deep neural networks that process NL text

How to learn the vectors?

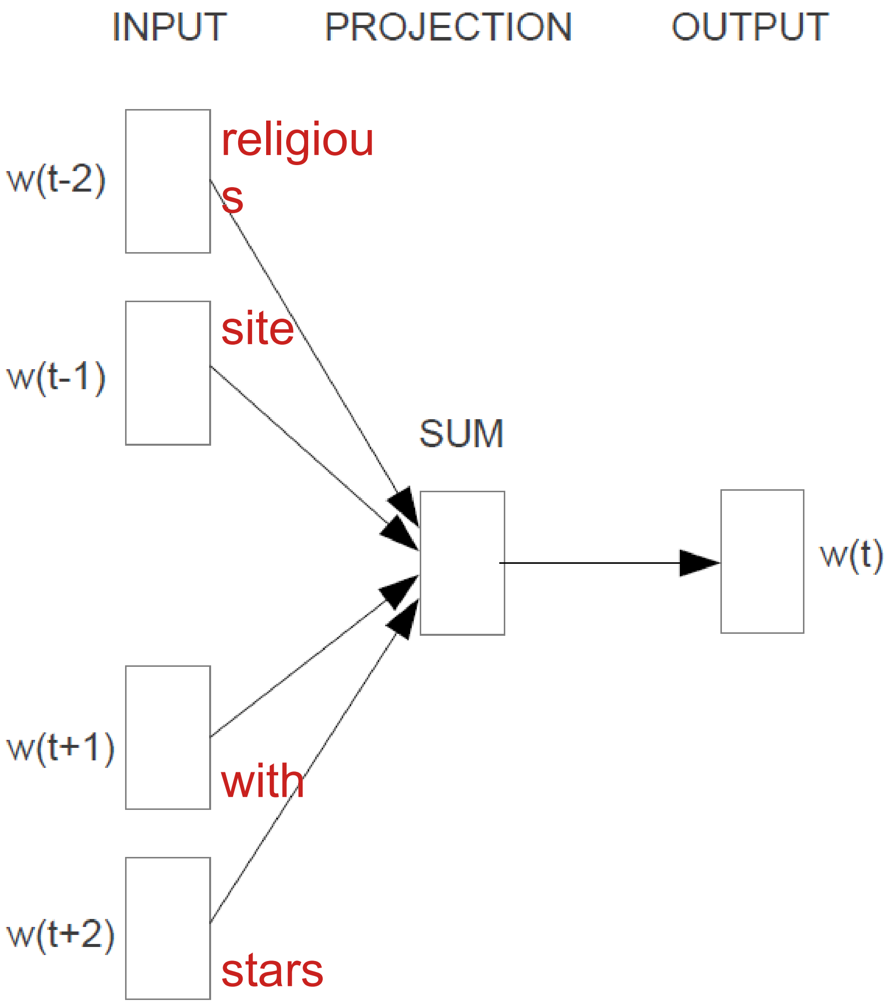
- CBOW (Continuous Bag of Words):
  - part of *word2vec*
  - neural network with one hidden layer
  - trained on large corpus of NL text (1.6 billion words)
  - input examples: sentences with one word missing
  - expected output: the missing word
  - the weights in the neural network are used as word vectors
- Also: Skip-Gram, GloVe, FastText, …
- Ubiquitous as *inputs to deep neural networks that process NL text*



CBOW

How to learn the vectors?

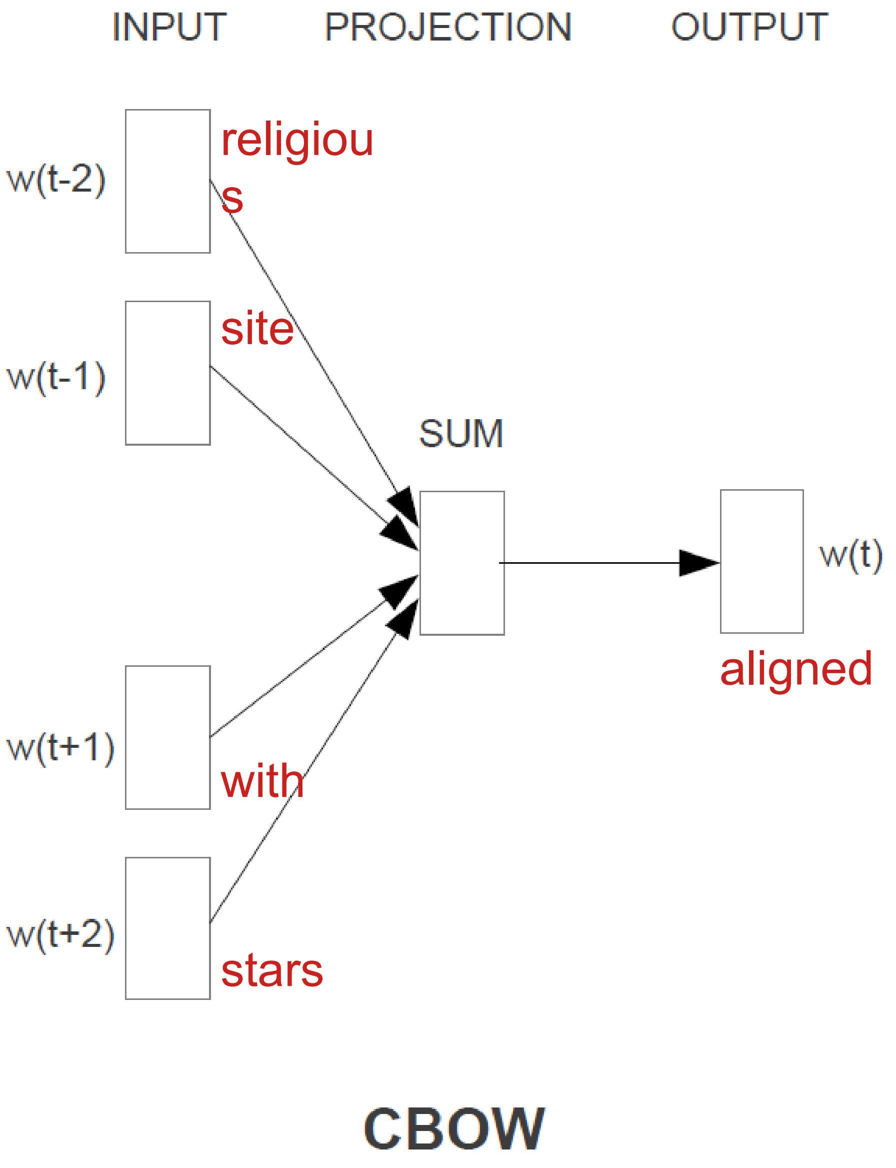
- CBOW (Continuous Bag of Words):
  - part of *word2vec*
  - neural network with one hidden layer
  - trained on large corpus of NL text (1.6 billion words)
  - input examples: sentences with one word missing
  - expected output: the missing word
  - the weights in the neural network are used as word vectors
- Also: Skip-gram, GloVe, FastText, …
- Ubiquitous as *inputs to deep neural networks that process NL text*



CBOW

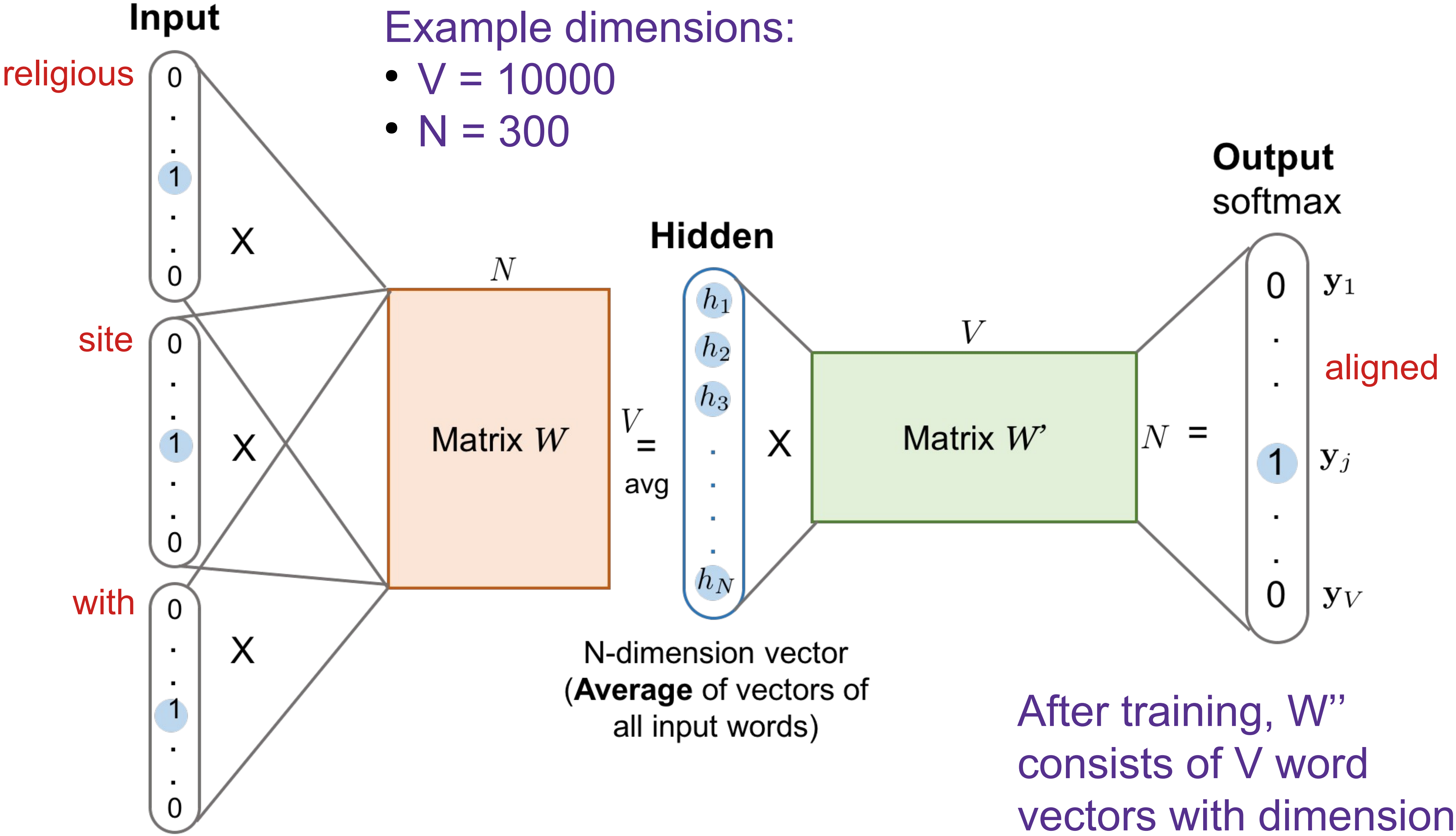
How to learn the vectors?

- CBOW (Continuous Bag of Words):
  - part of *word2vec*
  - neural network with one hidden layer
  - trained on large corpus of NL text (1.6 billion words)
  - input examples: sentences with one word missing
  - expected output: the missing word
  - the weights in the neural network are used as word vectors
- Also: Skip-gram, GloVe, FastText, …
- Ubiquitous as *inputs to deep neural networks that process NL text*



Example dimensions:

- V = 10000
- N = 300



After training, W'' consists of V word vectors with dimension N

Word similarity

- Extremely powerful and much used, *but be careful*
- The distributional hypothesis:
  - “words that occur in the same contexts tend to have similar meanings” (Harris 1954)
  - hence, word similarity can be measured in terms of vector similarity
  - *this is not true in general*
    - synonyms will often appear close to the same words
    - *but so will many antonyms* (“love”, “hate”)
  - syntagmatic similarity:
    the words are able to combine in sentences with the same other words
  - paradigmatic similarity:
    the words can be substituted with one another

Other types of embeddings

- Contextual embeddings (ELMo):
  - how to deal with words that are
    - homonymous (different words that look/sound the same)
    - polysemous (same word form has several meanings)
    - words have different embeddings in different neighbourhoods
- The idea has caught on:
  - phrase embeddings (“baseball bat”, “linear algebra”, …)
  - word piece embeddings ([lin-] + [-ear], [al-] + [-ge-]+ [-bra])
  - sentence and paragraph embeddings:
    - *transformer* models with *attention*: ChatGPT, GPT-4...
  - *graph embeddings!*

What are graph embeddings?

How can we represent the meaning of graphs?

- By designation (e.g., textual descriptions in a dictionary)
- As nodes in a network (e.g., in a knowledge graph like ConceptNet)
- Formally (e.g., adding axioms to a RDFS vocabulary or OWL ontology)
- As *embeddings*, i.e., *vectors* in a *latent semantic space*!
  - node vectors
  - edge vectors
  - graph vectors

What can we do with graph embeddings?

- Graph completion and validation:
  - node classification: given a node which type should it have?
  - link prediction: given a node and a edge, what should be at the end?
  - relation prediction: given two nodes, which edge type should link them?
  - triple classification: given two nodes and an edge, is the triple correct?
- Graph (or sub-graph) classification:
  - what type of entity/situation/event does the graph represent?
  - which class does the graph represent?
- Input to deep networks:
  - perhaps in combination with text, images, ...
  - deep multi-stream networks
  - early or late fusion of streams

How to learn the vectors?

- Early and simple example:
  - *Deepwalk* (2014)
- Algorithm:
  1) drop a marker randomly onto a graph node
  2) let the marker traverse the graph randomly along edges for *n* steps
     - additional parameters can guide traversal
  3) treat each resulting walk of *n* nodes as a sentence of *n* words
  4) feed a corpus of *n*-node walks into CBOW or similar
- *Instead of a vector for each word, this produces a vector for each node*
- Limitations:
  - all relations are treated as (nearly) equal
  - sampling may not fully reflect graph structure

Translational embeddings (TransE)

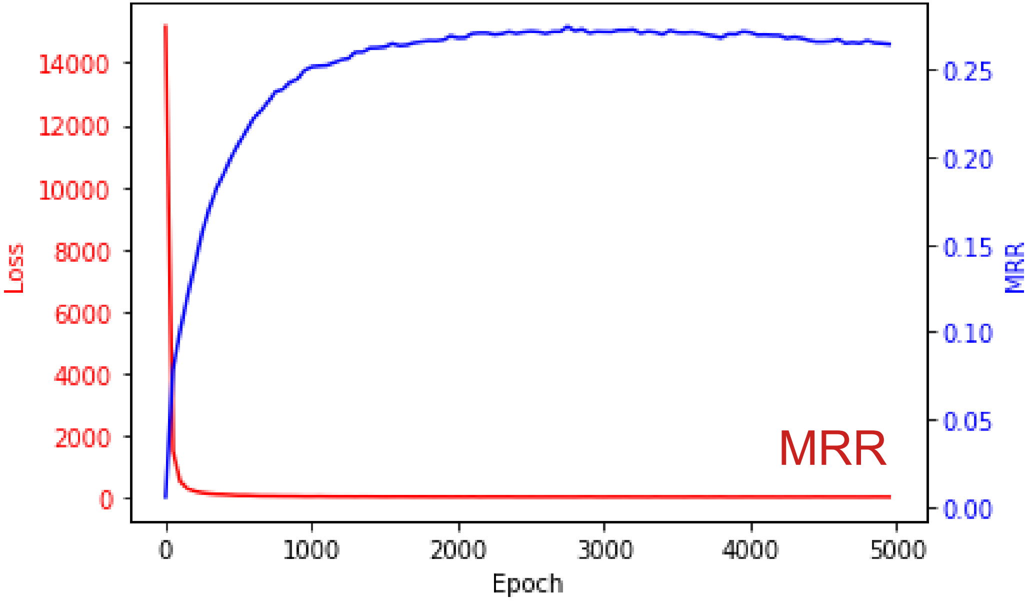
TransE is a simple example with a few known problems… There are many other models!

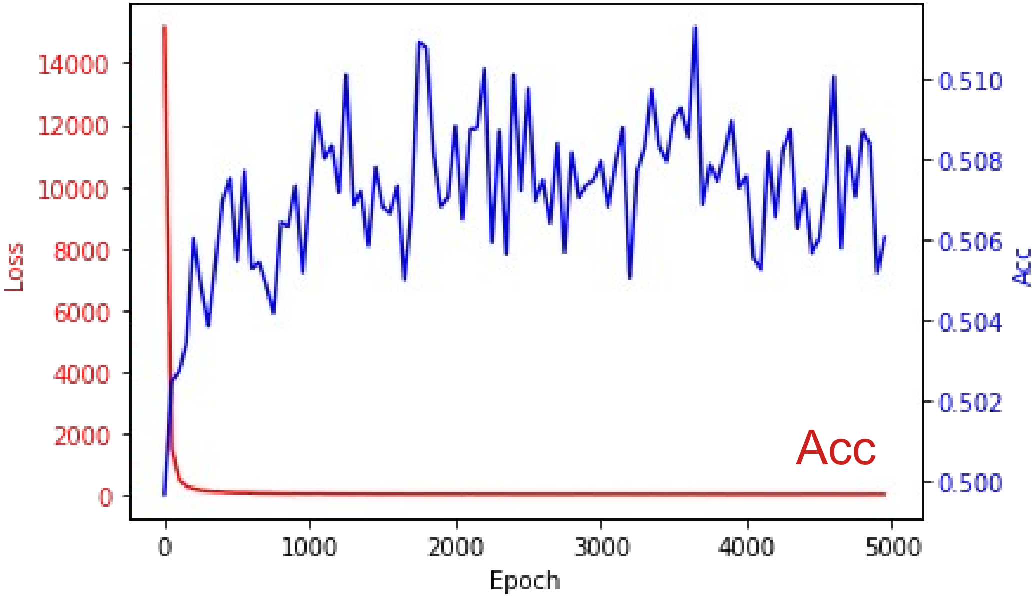
- The *translational property*:
  - *if (h, r, t) ϵ KG, then [h] + [r] ≈ [t]*
- Approach:
  - start out with random vectors for nodes and edges
  - repeat:
    - for each examle (h, r, t) ϵ KG, generate a corrupted (h’, r, t’) that is *not* in KG (because either h’ or t’ is changed)
    - adjust vectors to
      - *minimise dist*([h] + [r], [t])
      - *maximise dist*([h’] + [r], [t’])
      - *loss* per example is calculated from the *difference* between *dist*([h] + [r], [t]) and *dist*([h’] + [r], [t’])

Evaluation

- *Link prediction:*
  - h + r ≈ which t?
  - measures: MRR, Mean Rank, Hit@n (@10).
  - filtered and raw variants
- *Relation prediction:*
  - h – t ≈ which r?
  - measures: MRR, Mean Rank, Hit@n (@10).
  - filtered and raw variants
- *Relation classification:*
  - are (h, t, r) and (h', t, r') in KG?
  - accuracy (A)

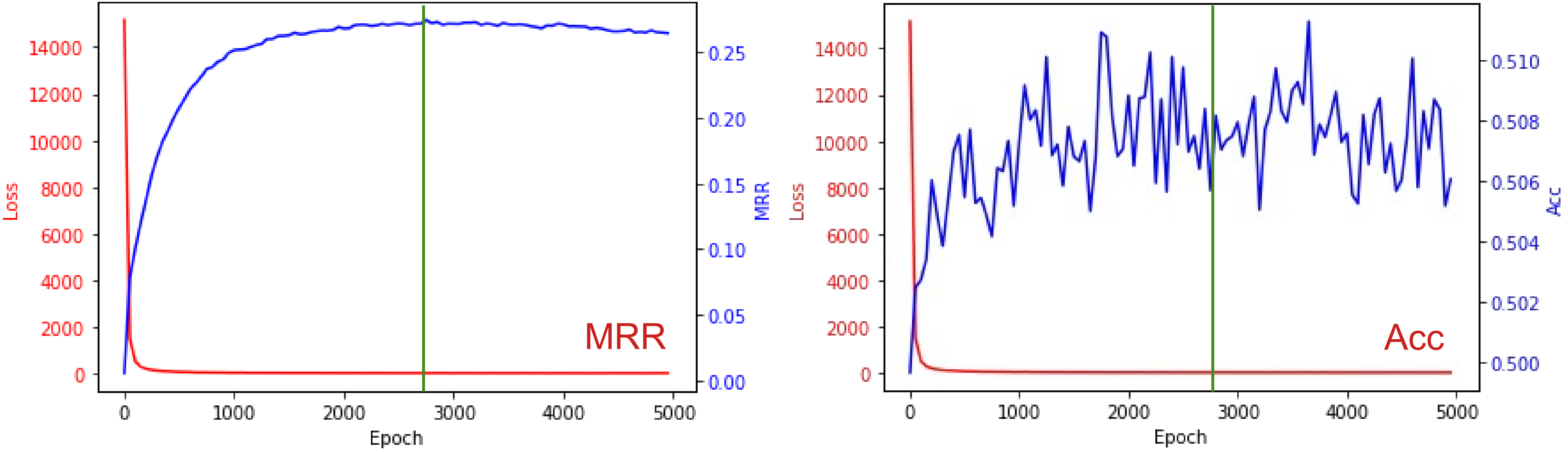
Learning curves





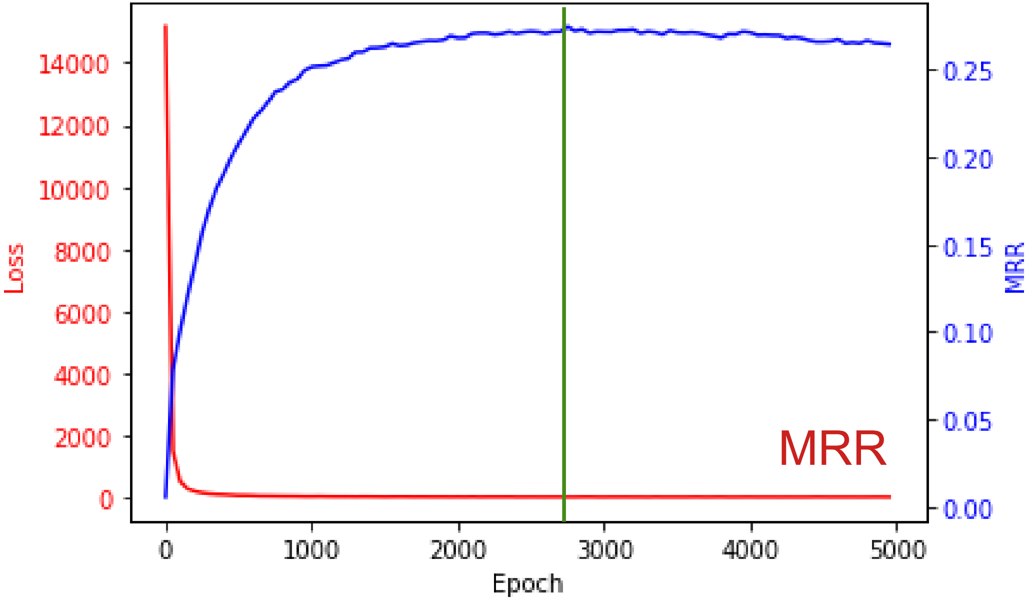
TransE on FB15k237 with 5000 epochs

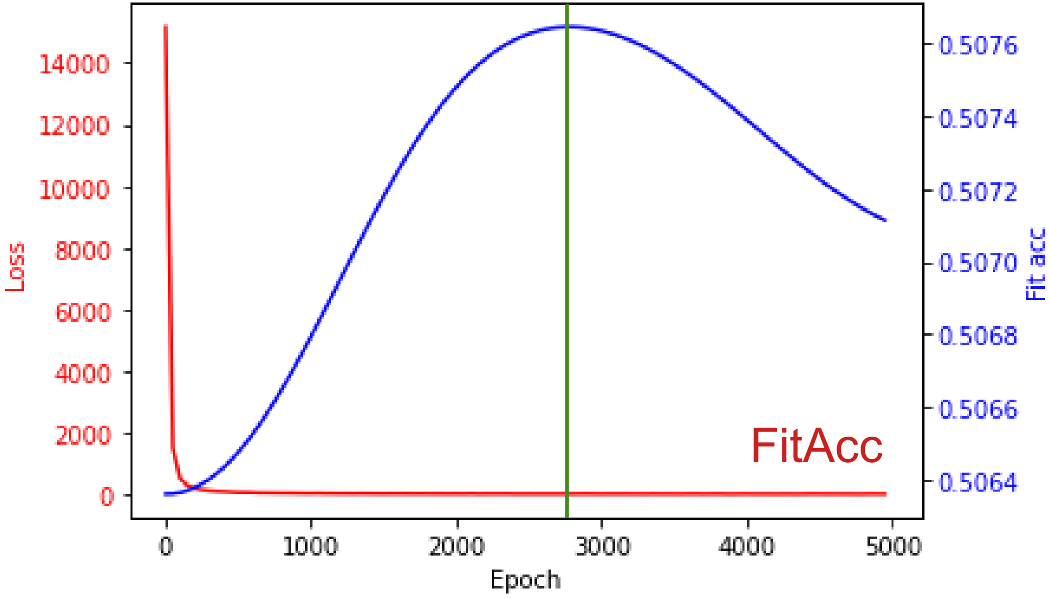
Learning curves



TransE on FB15k237 with 5000 epochs

Learning curves





TransE on FB15k237 with 5000 epochs

Datasets and pre-trained models

- Datasets:
  - Freebase extract (FB15k)
  - WordNet synsets (WN)
  - both have problems with training/validation/test overlap:
    - use FB15k237 and WN18RR instead
- Pre-trained models:
  - for example TransE already trained on FB15k237

DATA SET	WN	FB15K	FB1M
ENTITIES	40,943	14,951	1×10^6
RELATIONSHIPS	18	1,345	23,382
TRAIN. EX.	141,442	483,142	17.5×10^6
VALID EX.	5,000	50,000	50,000
TEST EX.	5,000	59,071	177,404

Limitations

- *TransE* is powerful and simple, but has limitations:
  - works best for 1-1 relations
  - trained on corrupted (h’, r, t) and (h, r, t’) variants, but never (h, r’, t)
  - therefore (terribly) bad on relation prediction
  - several derivations:
    - *TransH, TransR, TransD, TorusE,* …
  - more recent developments:
    - *Graph Neural Netwoks (GNNs)*
    - e.g., *Graph Convolutional Networks (CGNs)*
    - combine ideas from:
      - Convolutional/Recurrent Neural Networks (CNNs/RNNs)
      - big graph databases

The week after next:
KGs and LLMs