Natural Language and Speech Processing

Lecture 8: Neural Language Models and Word Embeddings

Tanel Alumäe

Contents

- Using neural networks for language modeling
- Side effect of language modeling: word embeddings
- Properties and applications of word embeddings

Language modeling

- Language modeling with N-gram models was covered in Lecture 4
- Problems with N-gram language models:
  - Cannot share strength among **similar words**:

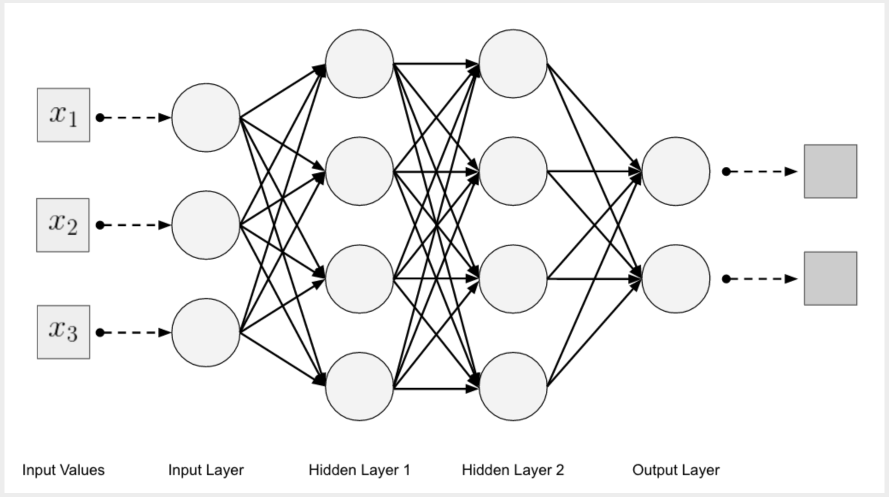
    `she bough a car`
    `she purchased a car`
  - Cannot condition on context with **intervening words**:

    `Dr. Jane Smith`
    `Dr. Gertrud Smith`
  - Cannot handle **long-distance dependencies**

    `for tennis class he wanted to buy his own racquet`
    `for programming class he wanted to buy his own computer`

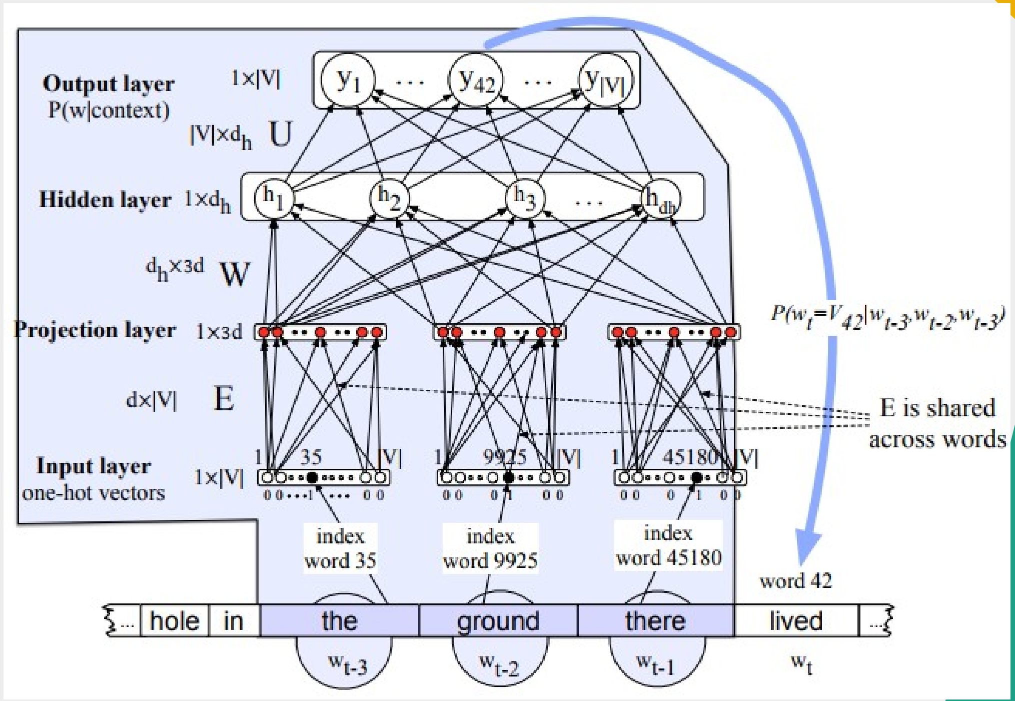
Neural network language model

- Neural network language model use previous N-1 words (context) as features
- Output layer is softmax over the whole vocabulary (+ sentence end token </s>)
- Hidden layer can be viewed as a feature extractor that computes more complex representation of the context
- But how to represent the context words as features?



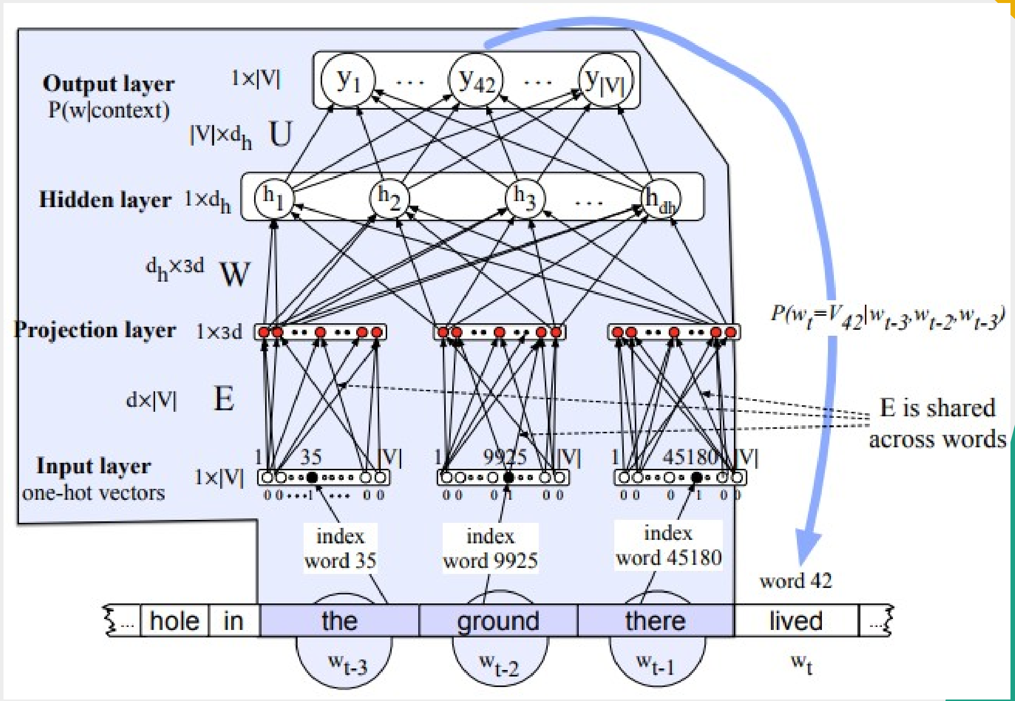
One-hot-encoding of context words

- N-1 previous words are represented using **one-hot encoding**
- Each word index is associated with a **projection vector** *e*
- The vectors together form a projection matrix *E*
- The projectior vector maps a word index to continuous vector representation (typically dimensionality: 100)
- The projection vectors of N-1 previous words are concatenated
  - Projection layer output: e_{i-1} e_{i-2} e_{i-3}
- The output of the projection layer is sent to the next hidden layer
- Projection vectors are **shared** between positions
  - i.e., we use the same *e* for the word "dog", regardless whether it is w_{i-1} or w_{i-2}



Neural network language model walkthrough

- Calculate probability of "*lived*", given previous words "*the ground there*":
  - Look up the indices of context words: *the* → 35, *ground* → 9925, *there* → 45180
  - Look up the projecton vectors, i.e. rows 35, 9925, 45180 from projection matrix *E*
  - Concatenate projection vectors: $e = e_{35}\ e_{9925}\ e_{45180}$
  - Apply the hidden layer: *h=sigmoid(W*e+b)*
  - Apply the softmax layer: y=*softmax(U*h)*
  - Look up probability of the current word: lived → 42
  - **The probability of "*lived*", given the context, is y_{42}**



Training the neural network language model

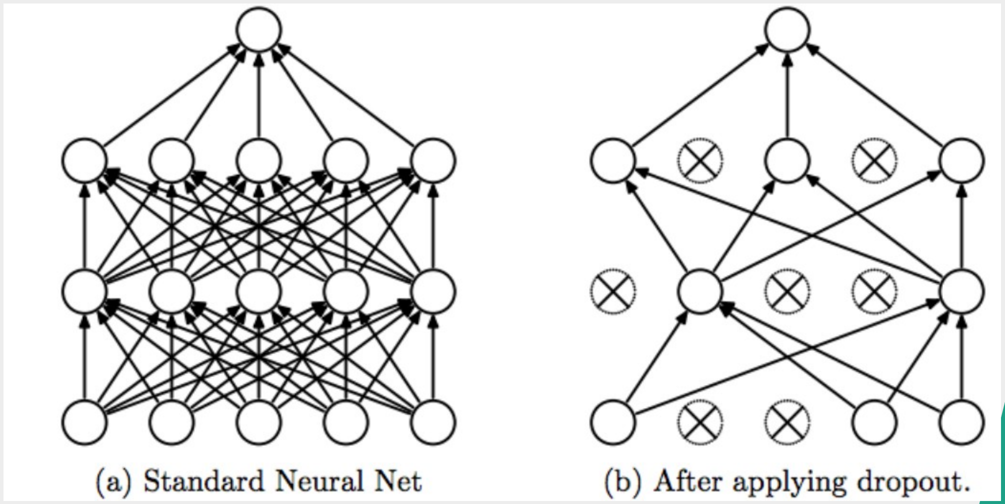
- Neural network language model can be trained using batch stochastic gradient descent, using cross entropy loss (although it’s expensive)
- Training examples are simply n-grams from the training corpus, with the first *n-1* words used as features and the last word of the n-gram as the target label
- It is important to shuffle n-grams before training (and at each epoch)

Training tricks

- Language models typically have a vocabulary of thousands of words (e.g. 200 000 is quite typical)
- This makes computing the output of the softmax layer very slow
- Solutions:
  - Use *shortlist*: use only most frequent 1000 words in the output layer, and use a regular N-gram model for the rest
    - But this kind of ruins the point of NNLM, as it is supposed to better model rare context/word combinations
  - Use a hierarchical softmax
    - Allows to compute the probability of a single word in O(log(|V|), rather than O(|V|)
  - Self-normalizing approaches
    - E.g. noise-contrastive estimation: model the target word against a small set of randomly drawn words
    - Modify the training algorithm so that the outputs of the last layer are encouraged to sum to approximately 1, so that normalizing (which is very expensive) can be left out

Using Dropout for regularization

- To prevent overfitting to training data, a technique called **dropout** is often used during training
- Applicable to all kinds of neural nets, not only to NNLMs
- Method: at each training stage (e.g. minibatch), ignore randomly selected neurons
  - This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass
- Use all neurons during the prediction phase
  - End effect: prediction is much like the average of the predictions across all these smaller nets
- Result: the network becomes less sensitive to the specific weights of neurons
  - This in turn results in a network that is capable of better generalization and is less likely to overfit the training data

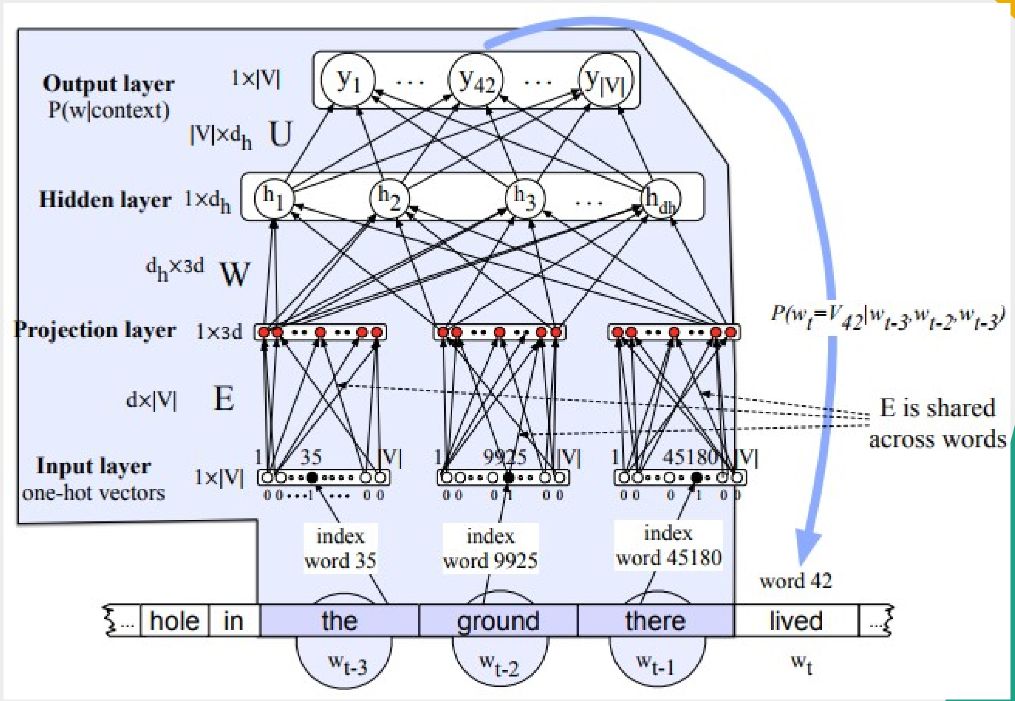


Benefits of Neural Network Language Model

- Improvements over the simple Ni-gram model
  - **Can** share strength among **similar words**:
    ```
    she bough a car
    she purchased a car
    ```
  - **Can** condition on context with **intervening words**:
    ```
    Dr. Jane Smith
    Dr. Gertrud Smith
    ```
- Does not solve the following problem
  - Cannot handle **long-distance dependencies**
    ```
    for tennis class he wanted to buy his own racquet
    for programming class he wanted to buy his own computer
    ```
  - This is handled by the recurrent neural network language model (we will study it in future lectures)
- In reality, NNLMs are better than N-grams with Kneser-Ney smoothing, but not a lot
  - Why not a lot? Probably because generalization sometimes hurts
- Typically used together with N-grams (probabilities are interpolated)

Useful side effect of NNLMs: word embeddings

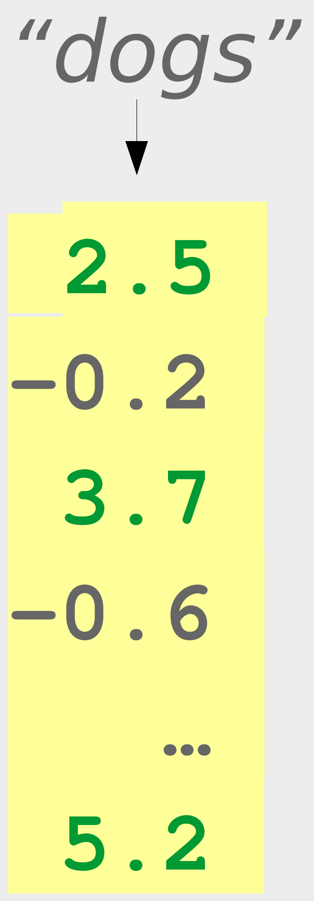
- When training a NNLM, we train a projection vector for each word
- The projection vectors are optimized using gradient descent, like all other model parameters
- The projection vectors map discrete words to continuous (typically 100 dimensional) space
- Training finds projections that are useful for language modeling
- Actual result: words that are **similar** (syntactically and semantically) have **similar** vectors!
- The projection vectors are also known as **word embeddings**



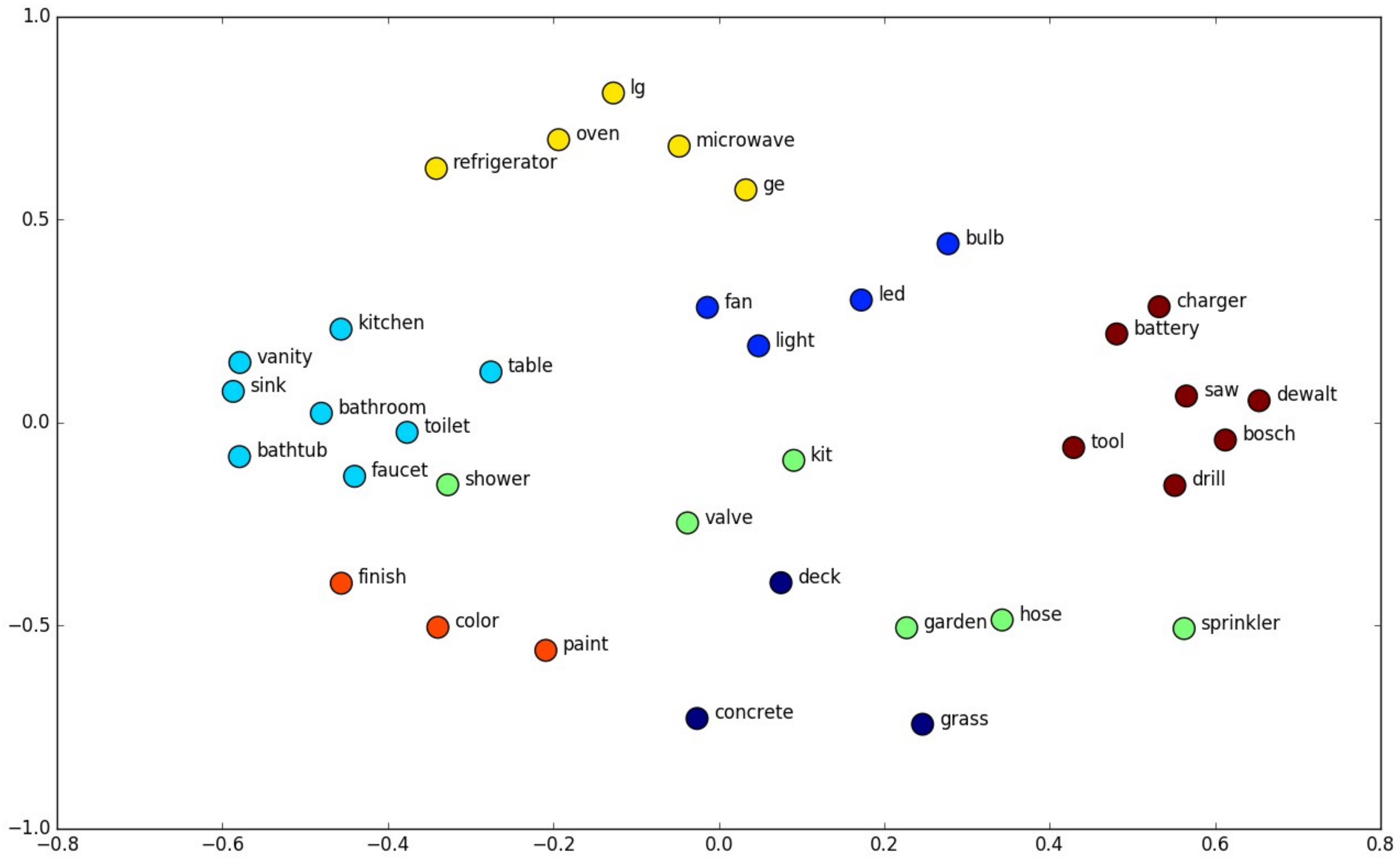
Output layer
P(w|context)
1×|V|
y_1 . . . y_{42} . . . $y_{|V|}$
$|V|\times d_h$ U
Hidden layer $1\times d_h$
h_1 h_2 h_3 . . . h_{dh}
$d_h\times 3d$ W
Projection layer 1×3d
$P(w_t=V_{42}|w_{t-3},w_{t-2},w_{t-3})$
d×|V| E
E is shared across words
Input layer one-hot vectors
1×|V|
1 35 |V|
1 9925 |V|
1 45180 |V|
index word 35
index word 9925
index word 45180
word 42
... hole in the ground there lived ...
w_{t-3} w_{t-2} w_{t-1} w_t

Intuition behind word embeddings

- Dimension 1 might be **more positive for nouns**
- Dimension 3 might be **more positive for plural nouns**
- Dimension N might be **more postive for domestic pets**
- Dimension 2 might have **no meaning whatsoever**

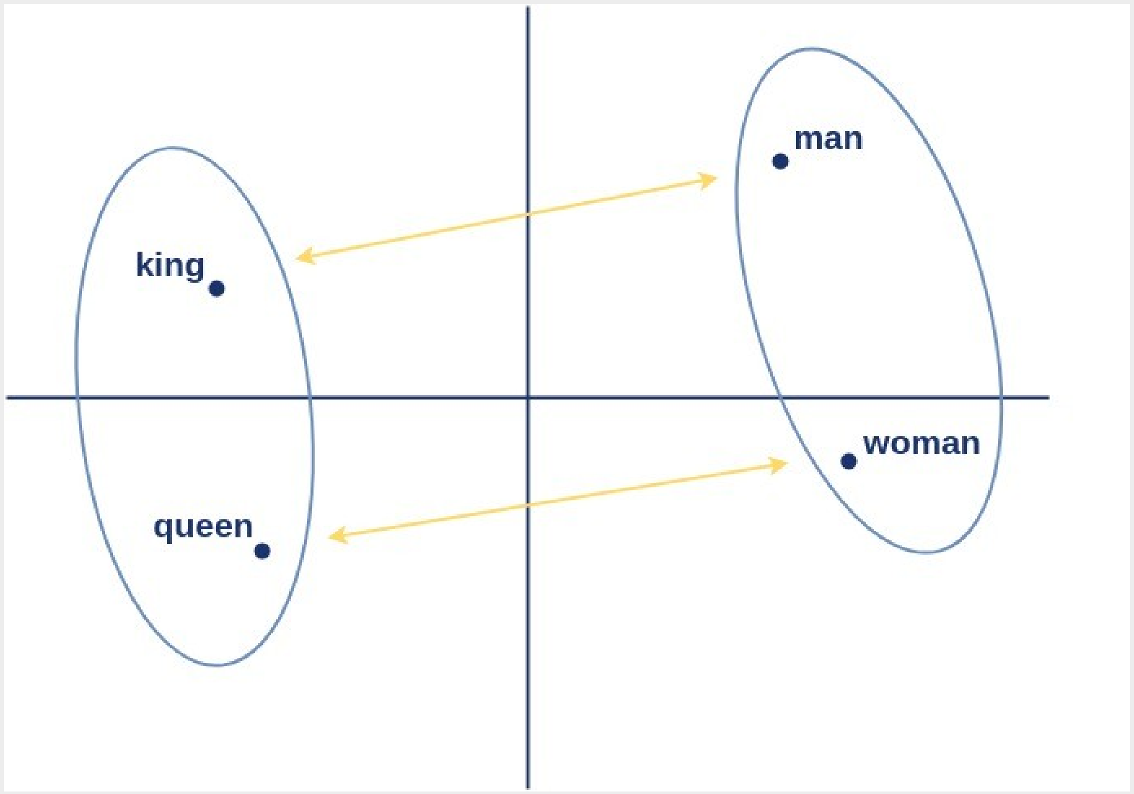


Example of word embeddings

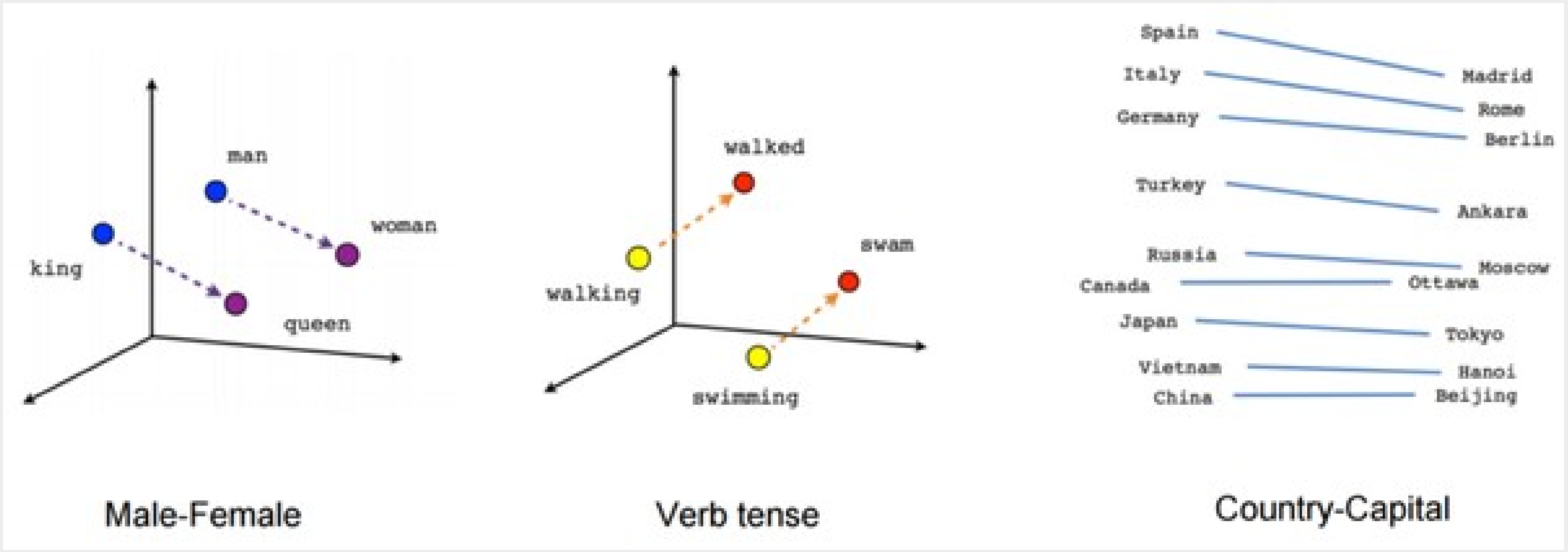


Word embeddings encode relationships between words

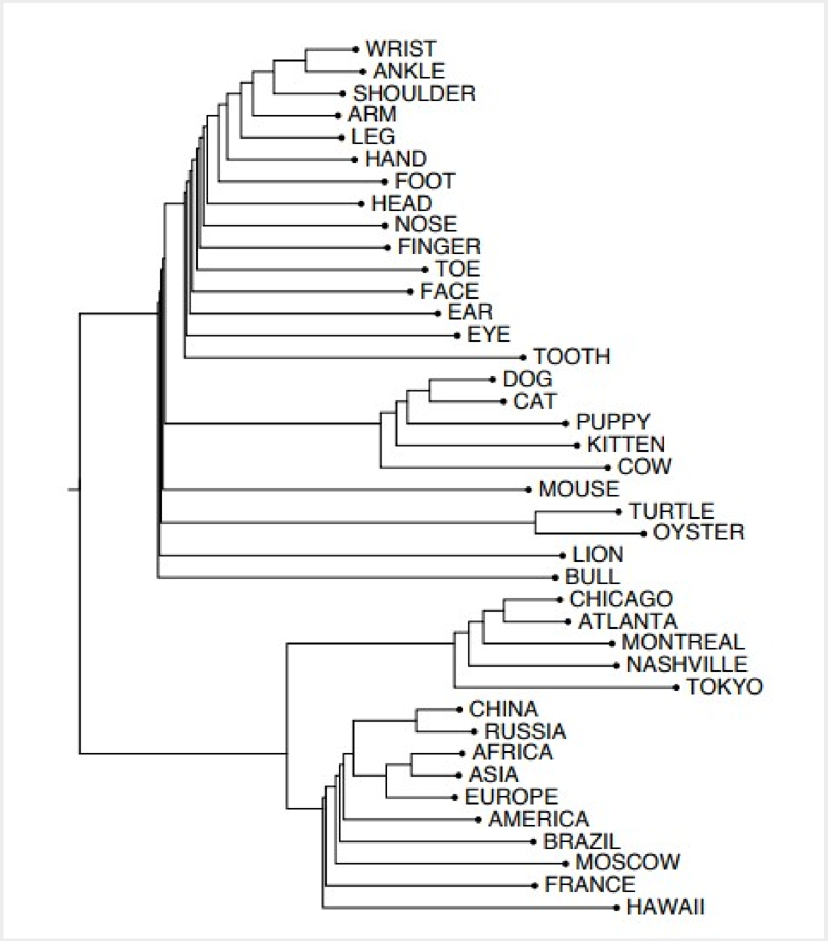
- Since words are now vectors, we can add them
- Result: *king – man + woman = queen*



Relationships in word embeddings



Semantic structures in word embeddings



- Word vectors can be structured into a tree, using clustering
- Interesting semantic structures appear

How to train word embeddings

- Train on the task:
  - e.g., if your model does named entity recognition, train word embeddings from scratch, together with the rest of the model
  - Good: embeddings will be optimized
  - Bad: needs large amounts of annotated (with names labeled) data

- Pre-train on large amounts of (unsupervised) text data
  - Use distributional similarity techniques to obtain “universal” word vectors
  - Good: can take advantage of large amounts of unlabeled text data
  - Bad: not necesserily optimal for the task

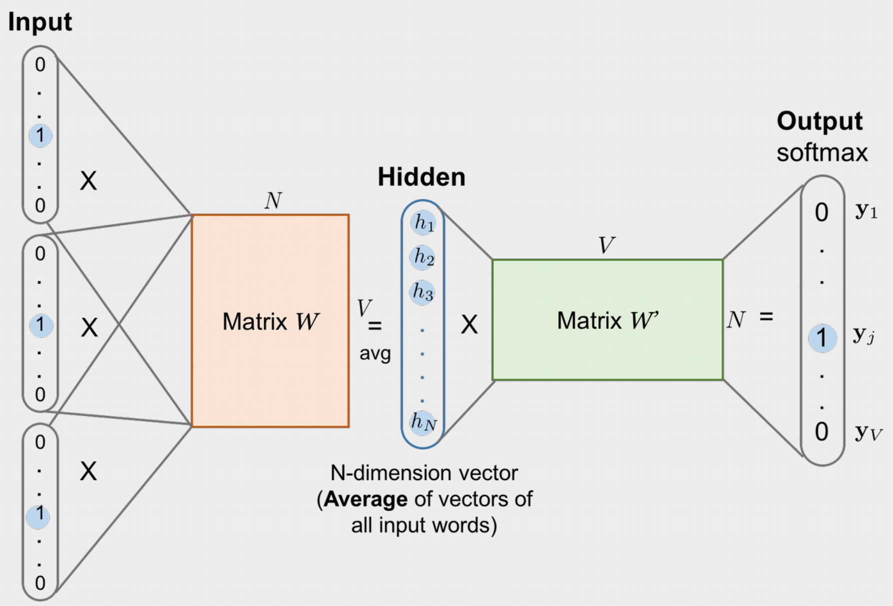
Best of both: use pre-trained embeddings,
and retrain them a little on the task

Unsupervised training of word embeddings

- Neural network language model gives us unsupervised word embeddings
  - Why unupervised – we don’t need words annotated with POS tags or NER labels
- However, NNLM is slow to train
- But we don’t actually need a properly normalized language model, we just want embeddings!
- Then it’s possible to use some tricks to make training of word embeddings much faster

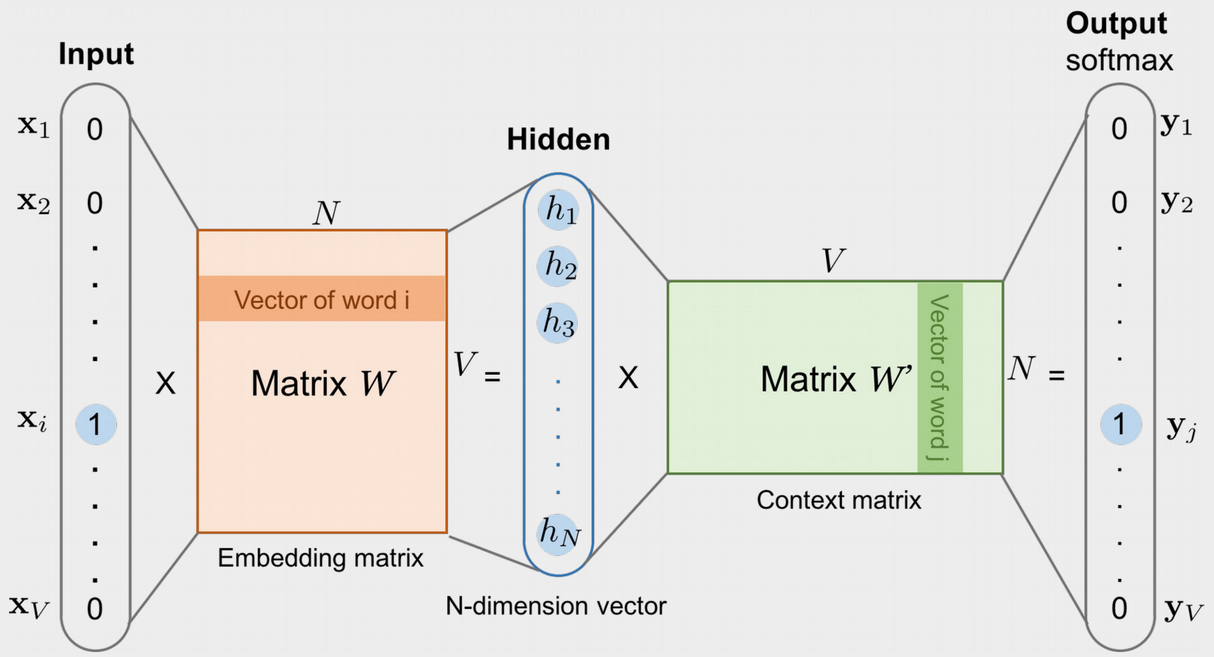
CBOW

- CBOW: Continuous Bag of Words
- Predict **center word** from **(a bag of) context words**
- The embeddings of context words are averaged
- The average embedding is as a feature vector to predict the center word
- At the end, the embeddings corresponding to context words are our word embeddings



Skip-ngrams

- Predict context words (position independent) given the center word
  - e.g., using a 5-word window
- Each pair of (center word, context word) is treated as a new observation when training the model
- E.g., given the sentence “the dog **eats** a bone”, the target word “*eats*” produces independent training examples:
  - `("eats", "dog"), ("eats", "a"), ("eats", "the"), ("eats", "bone")`



Negative sampling

- Negative sampling allows very fast training of word embeddings, when we don’t really care about the NNLM probabilities
- Softmax over words is replaces with a single sigmoid that is treated as a probability
- Training maximizes the probability of correct (seen) `(word, context)` pairs
- And minimizes the probability of randomly sampled `(word, context)` pairs
- In other words, the model tries to differentiate “real” word pairs from “fake” (unobserved) ones, using logistic regression
- The negative words are sampled from distribution *Q* that is something like a unigram distribution

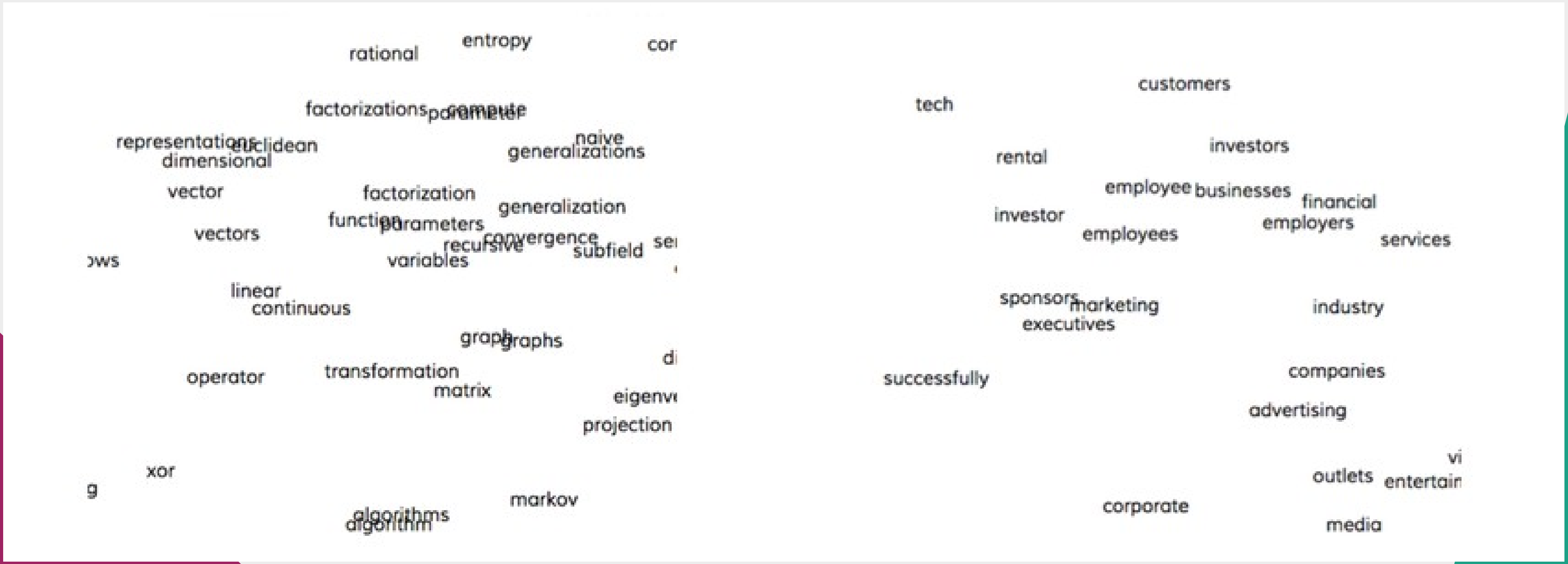
$$L = -\log(\operatorname{sigm}(u_{word}^{T} v_{contextword})) - \sum_{\substack{k=1 \\ w_i \sim Q}}^{K} \log(\operatorname{sigm}(-u_k^{T} v_{contextword}))$$

Effect of context size

- Context size in CBOW and skip-ngram is a tunable parameter that has a large effect on the nature of the embeddings
- Small context window: more syntax-based embeddings
- Large context window: more semantics-based topical embeddings

Visualization of embeddings

- Embeddings are high-dimensional
- 100-dimensional space is impossible to visualize
- Non-linear projections (e.g. SNE/t-SNE) group things that are close in high-dimensional space
- Projection from 100-dimensional space to 2D is lossy, so the result can be misleading



More interactive visualizations

- https://projector.tensorflow.org/
- http://bionlp-www.utu.fi/wv_demo/

Limitation of word embeddings

- Sensitive to **superficial differences** (*dog*/*dogs*)
- **Insensitive to context** (financial *bank*, *bank* of a river)
- **Not necessarily coordinated** with knowledge or across languages
- **Not interpretable**
- Can **encode bias** (encode stereotypical gender roles, racial biases)

Sub-word embeddings

- The internal character structure is ignored by regular word embedding models
- In inflected languages (e.g., Estonian), many words (e.g., “*koertelegi*”) occur rarely, making it difficult to learn good word embeddings for them
- But in inflected languages (and in most other languages), word formation follows some rules
- Is it possible to use character-level information for improving word embeddings?

Sub-word embeddings

- In the sub-word based model, each word is represented as a bag of character n-grams, and a special token corresponding to itself
  - `Where → [_wh, whe, her, ere, re_, <where>]`
  - Use all n-grams with *n=3…6*
- A word embedding is obtained by summing the embeddings of its components
- There are **a lot of** character n-grams of length 3…6
- Hashing trick is used to map all n-grams to integers from 1 to K (e.g. K=1000000)
- This technique is used in the *fastText* toolkit by Facebook

De-biasing embeddings

- Word embeddings tend to **amplify** biases in training data
- E.g. man-woman ~ architect-receptionist
- Occupations that are related to
  - Females: homemaker, nurse, receptionist, librarian, hairdresser, nanny, bookkeeper
  - Males: skipper, philosopher, captain, architect, warrior, broadcaster, boss
- Why is it bad? It reflects the bias in society, doesn't it?
  - Consider a web search "*TTU computer science Master student*" (e.g., by a potential employer)
  - If the search engine ranks web pages (partly) based on word embeddings, then a homepage of a female (e.g. *Mary*) might be ranked lower than that of a male (e.g. *John*), because *John* and "*computer science*" have closer embeddings
  - Therefore, gender bias in society could be **amplified** by systems using word embeddings
- Bias can be reduced by removing the gender-associations of gender-neutral words