Unsupervised Learning

Dr. Gerhard Paaß
Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
Sankt Augustin

TensorFlow

Fraunhofer
BIG DATA AI

Course Overview

1.	Intro to Deep Learning	Recent successes, Machine Learning, Deep Learning & types
2.	Intro to Tensorflow	Basics of Tensorflow, logistic regression
3.	Building Blocks of Deep Learning	Steps in Deep Learning, basic components, multilayer networks
4.	Unsupervised Learning	Embeddings for meaning representation, Word2Vec, BERT
5.	Image Recognition	Analyze Images: CNN, Vision Transformer
6.	Generating Text Sequences	Text Sequences: Predict new words, RNN, GPT
7.	Sequence-to-Sequence and Dialog Models	Transformer Translator and Dialog models
8.	Reinforcement Learning for Control	Games and Robots: Multistep control
9.	Generative Models	Generate new images: GAN and Large Language Models

: link to background material, : link to images used in lecture, G. : Terms that may be asked in the exam

Unsupervised Learning

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Capturing the Meaning of Words

- There are very many words in natural language
  - Some of them are very similar in their meaning
  - Most have very different meaning
- Many words have nearly the same meaning: **Synonyms**
  - child – kid, couch – sofa, big – large, automobile- car
- This similarity cannot be seen from the writing
- Manual annotation of meaning:
  - much effort
  - no accepted meaning annotation available

"The New Sofa" by bcmom / CC BY 2.0

"My new couch" by starathena / CC BY 2.0

"File:BMW automobile 1938.jpg" by dave_7 from Lethbridge, Canada / CC BY 2.0

"My Car" by Cryostasis / CC BY-SA 2.0

Need a **model**:
which words are more (or less) similar in meaning?

➔ Use **unsupervised learning**: no annotated data

much data available

Context and Meaning

John Rupert Firth

Public domain

- Similar words usually occur in similar contexts
- a word is characterized by the company it keeps [Firth, 1957]

The child is playing in the sand.
The mother loves her child .
The young child is crying loudly.

- Without problems we may exchange kid and child.
  ➔ very high similarity

A bottle of ***tesgüino*** is on the table
Everybody likes ***tesgüino***
Tesgüino makes you drunk
We make ***tesgüino*** out of corn.

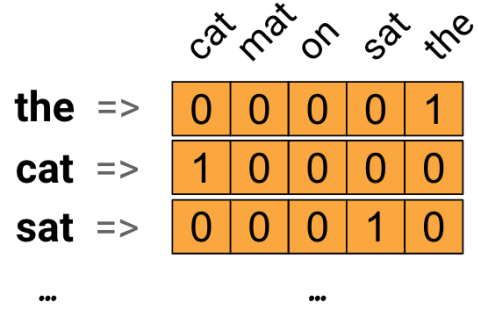
- Intuition for algorithm:
  - Two words are similar if they have similar word contexts.
  - same set of words $w_1, \ldots, w_k$ has a high probability to occur with word w_a and w_b
    ➔ w_a and w_b have similar meanings

[Dan Jurafsky Vector Semantics]

Representing Words by Numeric Vectors

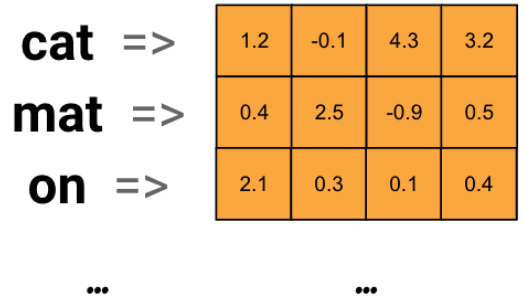
- We represent each word by a **vector of numbers**
- Represent each word by single vector component:
  One-hot encoding
  - Pro: Unique description of each word
  - Cons: Need vectors of length $\gg 10.000$ to accommodate all words
  - Cons: No way to compare the meaning of words.

One-hot encoding



- Represent each word by short vector
  with different values in many components:
  Embedding
  - Pro: Need shorter vectors, less space
  - Pro: Can compare the meaning of words.
  - Cons: Vague description of meaning
  - Cons: No intuitive way to find values.

A 4-dimensional embedding

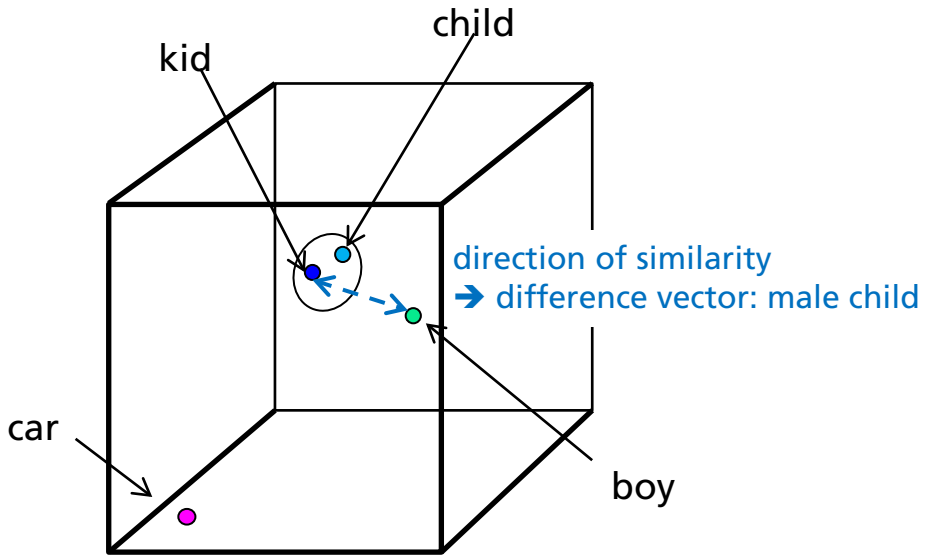


G. embedding vector G. embedding neigborhood

Embeddings

- Represent each word by a small **vector of numbers: Embedding**
- Words have similar meanings ➔ their Vectors are close together
  [Wikipedia: Distributional semantics]

- Close: very similar words
- related words: a bit further
  - different directions of
- Dissimilar words far apart



- If we have vectors of length > 100
  ➔ very many possible **directions of similarity**
  ➔ characterized by difference vector

can cope with many dimensions of meaning

G. embedding vector G. embedding neigborhood

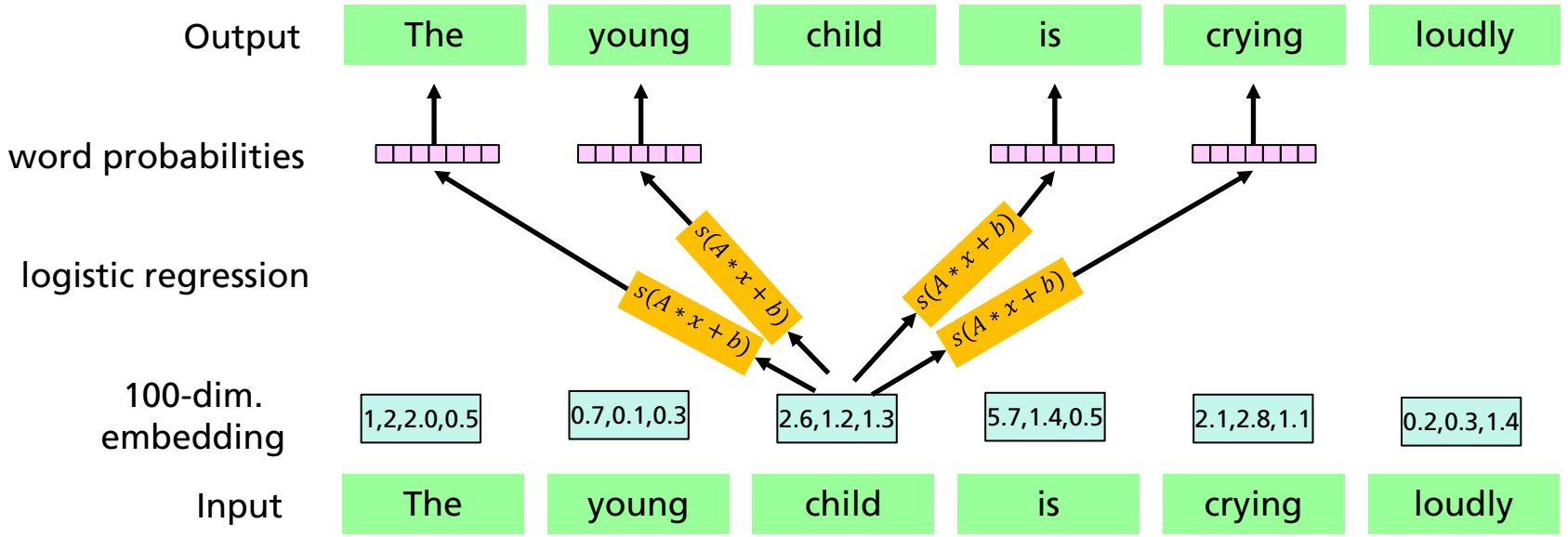
Unsupervised Learning

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Word2Vec: Skipgram

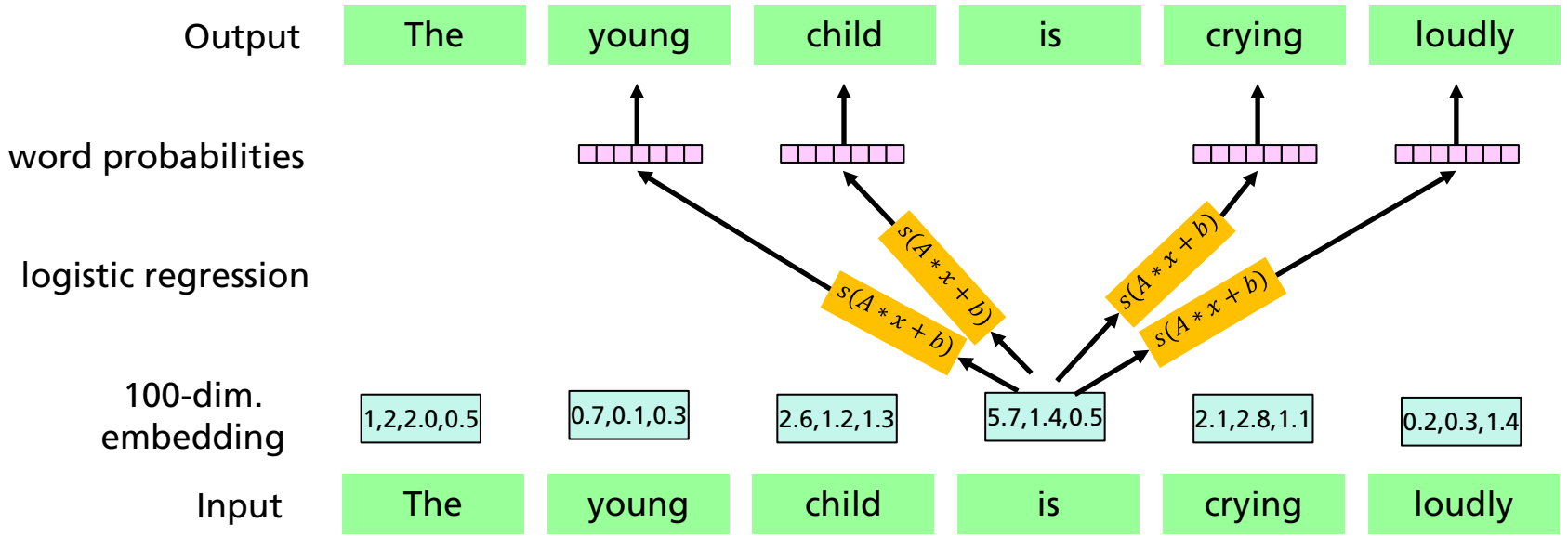
- For word w_t create an **embedding** $emb(w_t) \in \Re^k$ of size k, e.g. $k = 100$
- Predict **neighbor** words from the embedding of the **central word**
- **Skipgram** algorithm uses a simple model: linear **logistic** model: $p = softmax(A * x + b)$
- initialize parameters A, b and embeddings $emb(w_t)$ randomly



Word2Vec: Skipgram

no annotation required

- For word w_t create an **embedding** $emb(w_t) \in \Re^k$ of size k, e.g. $k = 100$
- Predict **neighbor** words from the embedding of the **central word**
- **Skipgram** algorithm uses a simple model: linear **logistic** model: $p = softmax(A * x + b)$
- initialize parameters A, b and embeddings $emb(w_t)$ randomly



- Repeat this for all words of the collection: train by stochastic gradient descent

Word2Vec: Logistic Regression Classifier

100-dim. embedding

- step 1: initialize parameters A, b and embeddings $emb(w_1), \ldots, emb(w_{60,000})$ randomly
- step 2: loop over words of a batch
  - step a: lookup embedding for input word $w_t =$ „child“ ➔ $x_t = emb(w_t) =$ 0.7,0.1,0.3
  - step b: select neighbor word to be predicted $w_{t+1} =$ „young“
  - step c: linear map $x\_t =$ 0.7,0.1,0.3 ➔ $u_t = A * x_t + b$ (Scores of all words) $\dim(u_t) = 50000$ (vocabulary size)
  - step d: convert to probabilities by softmax

$$p(w = i|w_t) = softmax_i(u_t) = \frac{\exp(u_{t,i})}{\exp(u_{t,1}) + \cdots + \exp(u_{t,50000})}$$

  - step e: cross-entropy loss to compare probabilities with actual word „young“:

$$loss = -\log\left[softmax_{w_2}(A * emb(w_1) + b)\right] - \cdots - \log\left[softmax_{w_{N-1}}(A * emb(w_N) + b)\right]$$

predicted probability of word at pos. 2

predicted probability of word at pos. $N-1$

  - step f: compute gradient of loss w.r. elements of parameters A, b and embeddings $emb(w_t)$
- step 3: averages gradients and update parameters A, b and embeddings $emb(w_t)$ go to step 2

Word2Vec: Optimization

- Final loss function is **minimized** ➔ maximize probability of neighbor word
  change embeddings $emb(w_t)$ and parameters W, b
  - compute gradients for a batch with several words w_t and their neigbors $w_r \in Neighborhood(w_t)$
  - update parameters with gradient
  - repeat for all w_t in the corpus
- For the best values $embedding, \widehat{W}, \hat{b}$ the probability of
  neighboring words of the training set is maximal

Word2Vec: Loss function

Score for „degree“ at pos. t

- softmax has very many terms in the denominator

$$\text{softmax}_i(u_t) = \frac{\exp(u_{t,i})}{\exp(u_{t,1}) + \cdots + \exp(u_{t,50000})}$$

- approximate by a sample
  ➔ **negative sampling**

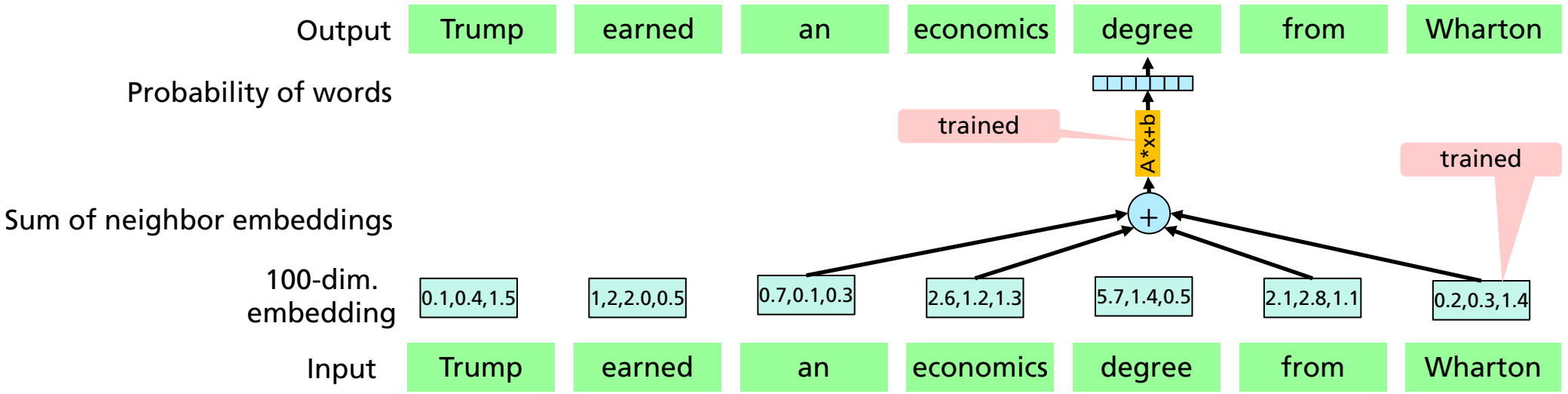
$$\text{softmax}_i(u_t) \approx \frac{\exp(u_{t,i})}{\exp(u_{t,i}) + \sum_{j \in sample} \exp(u_{t,j})}$$

- Approximates the true cross-entropy
  ➔ gradient
- Optimize by **Stochastic Gradient Descent**: (SGD)
  - compute gradient just for a small batch of sentences
  - Gradient varies randomly from true gradient
- Computationally efficient ➔ works for large corpora
- escapes local minima

?

G. negative sampling

Alternative CBOW: Predicting the center word



- Continuous Bag of Words (**CBOW**) [Mikolov et al. 2013b]
  - task: predict word w_t from neighbors $w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+k}$
  - lookup **embeddings** $emb(w_{t\pm i})$ of neighbors $w_{t\pm i}$ and sum up
    $h_t = emb(w_{t-k}) + \cdots + emb(w_{t-1}) + emb(w_{t+1}) + \cdots + emb(w_{t+k})$
  - Predict center word w_t

$$\text{softmax}_i(u_t) \approx \frac{\exp(u_{t,i})}{\exp(u_{t,i}) + \sum_{j \in sample} \exp(u_{t,j})}$$

usually a less efficient than skipgram

04-a

Unsupervised Learning

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Preparing the Data

- part of the English Wikipedia from March 2006 (text8)
  - 100MB of cleaned text without markup

```
filename = maybe_download(JUPYTER_DATA+'text8.zip', 31344016)
```

- Read data in a long list of strings

```
Read wikipedia data. Data size (number of words) 17005207

anarchism originated as a term of abuse first used against early working class radicals including
the diggers of the english
```

- Build vocabulary of 50000 most frequent words
  - Count frequency of different words
  - Sort words by frequency
  - Keep 50000 most frequent words
  - Replace other words by `UNK`.

count

```
Most common words (+UNK)
[['UNK', 418391], ('the', 1061396), ('of', 593677), ('and', 416629), ('one', 411764)]
```

Preparing the Data

- Convert words to integers

```
Sample data [5239, 3084, 12, 6, 195, 2, 3137, 46, 59, 156, 128, 742, 477, 1
0572, 134]
['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', 'first', 'us
ed', 'against', 'early', 'working', 'class', 'radicals', 'including']
```

- Generate **batches** containing Input and Output
  e.g.: batch_size = 8: block of batch_size examples for parallel processing
  skip_window = 2: number words to consider left and right.

```
train_inputs
as          as          as          as          a           a           a           a
train_labels
a           anarchism originated term      of          term        originated as
```

Neighbor of „as“

Neighbor of „a“

Use word2vec implementation in gensim

- sets up model according to parameter specification
- automatically forms vocabulary and assigns integer to words
- C++ implementation: very fast (fasttext)

```
from gensim.models.wrappers.fasttext import FastText as FT_wrapper
import time;  t0 = time.time()

# train the model without n-grams
model_wrapper = FT_wrapper.train(ft_home, text8File,
                                 model='skipgram',          # 'cbow', 'skipgram'.
                                 size=100,                  # size of embedding
                                 max_n=0,                   # maxlength of n-gram
                                 window=5,                  # size of neighborhood
                                 min_count=5,               # ignore all words with lower frequency
                                 threads=10,                # number of threads to use
                                 iter=5)                    # number of epochs
print('used',round(time.time()-t0, 2),'sec')
print(model_wrapper)
```

Unsupervised Learning

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Computing nearest neighbors

```
print('night' in model_wrapper.wv.vocab) # check if word in vocabulary
print('night\n',model_wrapper['night'])
print('nights' in model_wrapper.wv.vocab) # check if word in vocabulary
print('nights\n',model_wrapper['nights'])
```

```
True
night
 [ 0.6204168  -0.01493791 -0.12941618 -0.08685803 -0.10108315  0.40013582
 -0.32517195  0.01283947  0.12554175  0.19119295  0.03272424  0.59937114
  0.21048859 -0.37226528  0.13689515  0.26681447 -0.03403868 -0.39834082
  0.1299107  -0.1342344   0.12227379  0.6185183   0.03057314  0.08545388
 -0.48146752 -0.48143086 -0.05053187 -0.3227574   0.00466404  0.52115923
  0.21988557  0.06906822 -0.52175367 -0.06833102 -0.06808479 -0.3245593
  0.0137263  -0.26763853 -0.4524306   0.18784973  0.12353216  0.5543284
  0.13308579  0.18003018  0.20626368  0.04571371 -0.16112524  0.05146308
  0.23227286  0.62925917 -0.32259712  0.29065502 -0.20876545  0.20170932
 -0.00320623  0.21869478  0.06318002  0.4256705   0.1282726   0.8083877
 -0.27307126  0.07764974  0.37447548  0.23471321 -0.28535303  0.2615321
  0.32404765  0.10719042 -0.0287949  -0.22758937 -0.35373595 -0.0997579
  0.47504428  0.7266672   0.09146578 -0.19294515  0.17256027  0.36353913
 -0.22565459  0.12918787  0.8135199  -0.00447693  0.1344563   0.27139822
  0.1661349   0.5556222   0.08137181  0.42539957 -0.20297147 -0.26081067
  0.39592758 -0.08948609  0.39880133 -0.46983415  0.03422422 -0.7961935
 -0.21045214 -0.5354332   0.20080912  0.05690358]
```

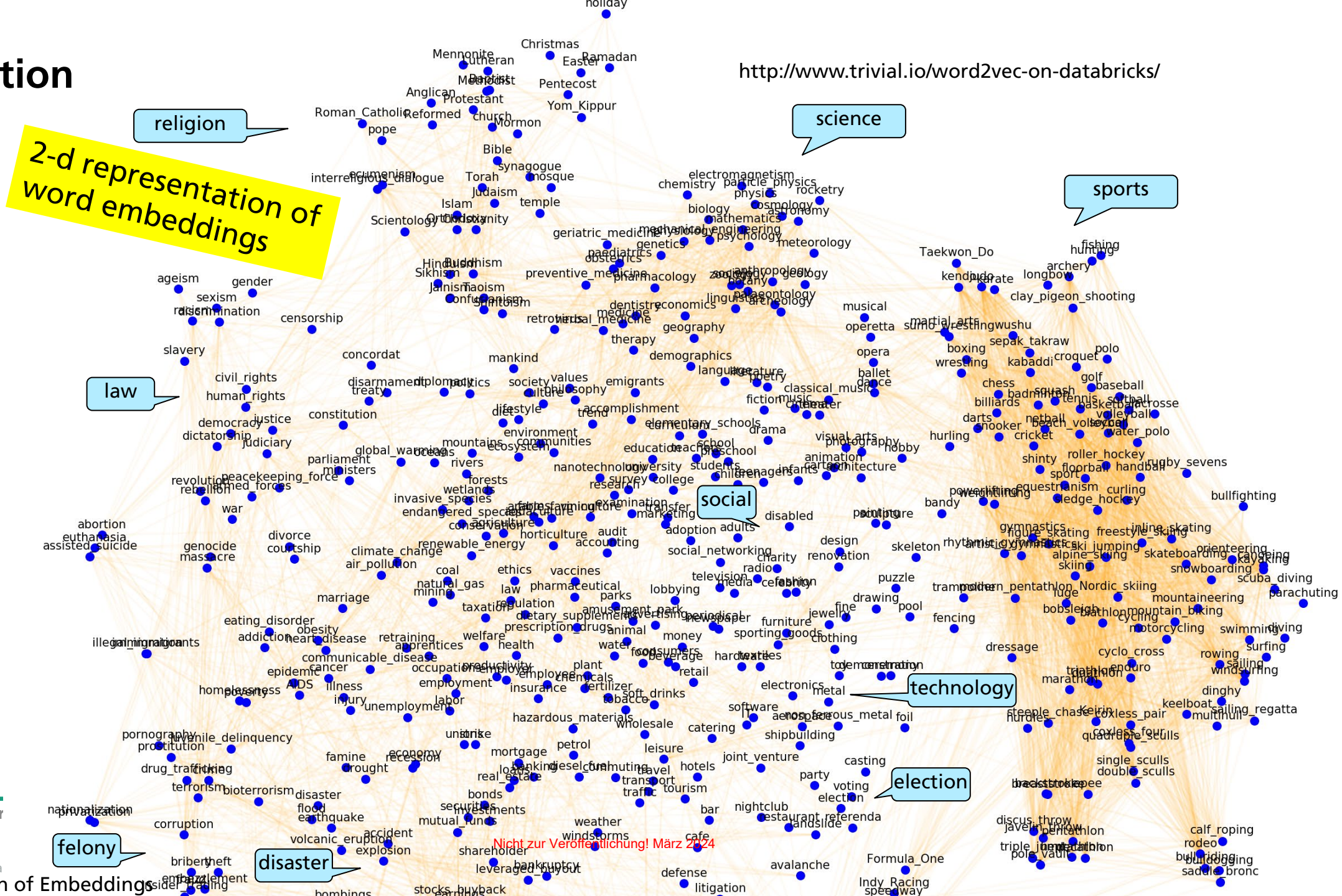
- get embedding of the word "proton"
- compute distance to embeddings of all words in the vocabulary
- select the closest embeddings
- print corresponding words and distances

```
pd.DataFrame(model_wrapper.most_similar("proton",topn=20))
```

	0	1
0	neutrons	0.869389
1	protons	0.864780
2	deuterium	0.863210
3	positron	0.862876
4	decays	0.837231
5	ions	0.834752
6	muon	0.834699
7	hydrogen	0.833226
8	neutrino	0.832478
9	acceptor	0.830248
10	antiproton	0.829165
11	tritium	0.827864
12	positrons	0.824720
13	nuclei	0.821067
14	electron	0.821055
15	pion	0.820621
16	photon	0.820317
17	enol	0.819259

Evaluation

http://www.trivial.io/word2vec-on-databricks/

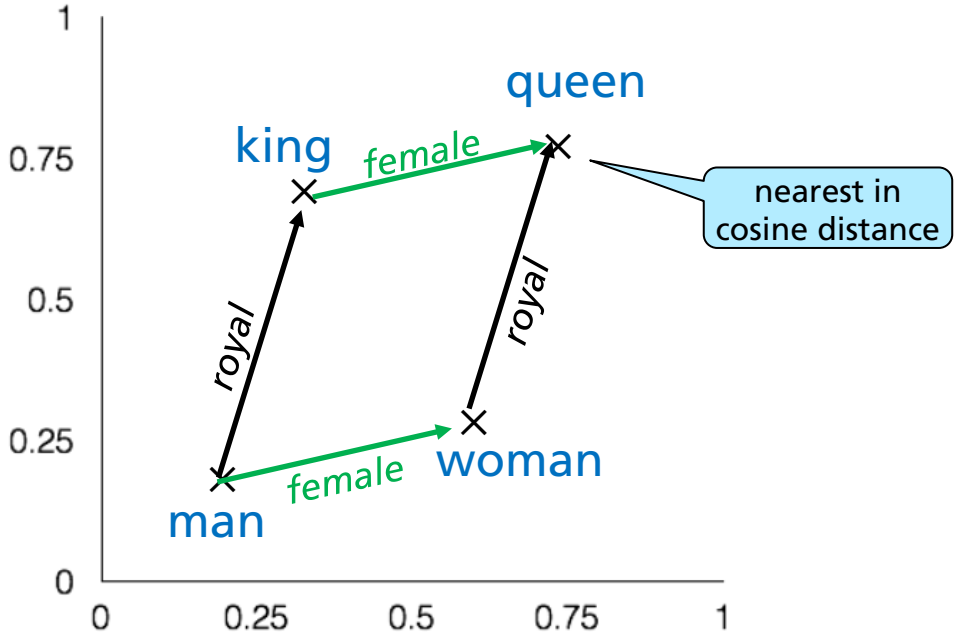


G. 2d-Representation of Embeddings

Embeddings Capture Relations: Embedding Difference

Test for linear relationships [Mikolov, Yih & Zweig 2013]

	Word	Embedding
+	king	[0.30, 0.70]
-	man	[0.20, 0.20]
=	*royal*	[0.10, 0.50]
+	woman	[0.60, 0.30]
=	queen	[0.70, 0.30]



- Relations ~ embedding difference
- Interactive analogy evaluation

```
Enter option:    usa dollar uk    gbp
```

nearest in cosine distance

G. Embedding difference

Nächste Veranstaltung! März 2024

Semantic and Syntactic Evaluation: Analogies

- A set of "questions" and "answers": "Athens Greece Baghdad" ➔ "Iraq"

  Answer: $\underset{w \in Voc}{\operatorname{argmin}} \| emb(Greece) - emb(Athens) + emb(Baghdad) - emb(w) \|^2$

- more than 19000 questions:
  - semantic (~ meaning)

    Athens Greece Bagdad Iraq,
    Algeria dinar Japan yen,
    Chicago Illinois Dallas Texas,
    boy girl dad mom,
    Albania Albanian Chile Chilean,
    amazing amazingly calm calmly,
    aware unaware clear unclear,

  - syntactic (~ grammatical)

    bad worse big bigger,
    bad worst big biggest,
    code coding dance dancing,
    dancing danced feeding fed,
    banana bananas bird birds,
    decrease decreases go goes
    city city's bank bank's
    see, saw, return, returned
    see, sees, return, returns

Comparison of Different Embedding Approaches

- Comprehensive evaluation by [Levy et al. 2015]
  - The performance depends strongly on **hyperparameter** optimizations.
  - Modifications may be transferred to other approaches: similar gains

Method	analogy
pointwise mutual info	.553
SVD	.554
skipgram	.676
GloVe	.569

Table 3 of [Levy et al. 2015]

- **analogy** dataset: 19544 syntactic and semantic relations

- **Skipgram** with negative sampling has good overall performance
  fastest to train & least disk space
- recommendations
  - use many negative samples
  - use specific smoothing factor for random example generation

Unsupervised Learning

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Problems with Word2Vec

- the word **bank** has several senses (homonym)
  - They pulled the canoe up on the bank
  - He cashed a check at the bank.
  - The plane went into a steep bank.
- Word2Vec ignores word order: *man kills lion* vs. *lion kills man*
  ~hunting ~accident
- Word2Vec determines embedding only from a small neighborhood
  ignores meaning difference induced by far-away words
  I need some money, which I could get from my parents or from the bank

Requirements

- Need different embeddings for each word in a specific context: **context-sensitive**
- take into account long-range relations between words
- exactly take into account the order of words

"Canoeing in Algonquin Park" by Christopher Blizzard / CC BY-SA 2.0

"Wells Fargo Bank" by JeepersMedia / CC BY 2.0

"A couple of brown airplanes at an air show" by Horia Varlan / CC BY 2.0

Use Token Instead of Words: Byte Pair Encoding

[Sennrich et al. 2016]

- Problem: there exists millions of words ➔ huge parameter matrices

Algorithm

- first select **all characters** as token ➔ vocabulary
- join the most frequent token pair to form a new token
- repeat until the vocabulary has the prescribed size

Properties

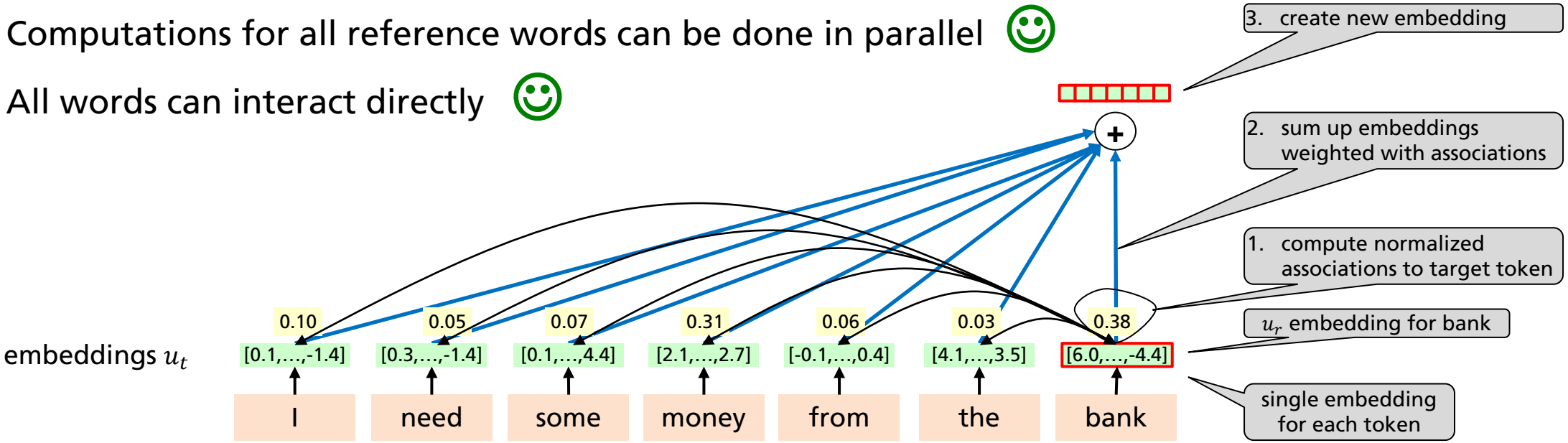
- frequent words are selected as tokens
- **can represent arbitrary words** by parts
- token vocabulary is adapted to training set

- Similar algos: WordPiece, SentencePart

```
e i n e _ r o s e _ i s t _ e i n e _ r o s e _
   vocabulary: e i n o r s t _

ei n e _ r o s e _ i s t _ ei n e _ r o s e _
   vocabulary: e ei i n o r s t _

ei ne _ r o s e _ i s t _ ei ne _ r o s e _
   vocabulary: e ei i n ne o r s t _

ei ne _ ro s e _ i s t _ ei ne _ ro s e _
   vocabulary: e ei i n ne o r ro s t _

ei ne _ ro se _ i s t _ ei ne _ ro se _
   vocabulary: e ei i n ne ne_ o r ro s se t _

ei ne_ ro se _ i s t _ ei ne_ ro se _
   vocabulary: e ei i n ne ne_ o r ro s se t _

eine_ ro se _ i s t _ eine_ ro se _
   vocabulary: e ei eine_ i n ne ne_ o r ro s se se_ t _
```

Alternative: Evaluate Association Between Embeddings

- compute **self-attention** between a reference word x_r and all sequence words x_t
  - access and incorporate information from all sequence words ➔ parametrized **association**
  - similar to a query from x_r to all sequence words x_t
  - repeat for all reference words
- Aggregate embeddings weighted by associations ➔ new embedding
- Computations for all reference words can be done in parallel ☺
- All words can interact directly ☺



Self-Attention: Details of Computations

- In the lowest layer: generate embeddings u_t for each token w_t
- Repeat for all reference embeddings $u_r, r = 1, \ldots, T$
  - Compute an **association** between a target embedding u_r and all sequence embeddings u_t
    1. **Query**: Transform embedding f target token with matrix $\boldsymbol{Q}$: $q_r = Q * u_r$
    2. **Keys**: Transform all sequence embeddings with matrix $\boldsymbol{K}$: $k_t = K * u_t$, $t = 1, \ldots, T$
    3. Compute scalar products: $\left(q_r' k_1/\sqrt{d}, \ldots, q_r' k_T/\sqrt{d}\right)$
       d is length of embeddings
  - Normalize to **probability vector**: $\left(\alpha_{r,1}, \ldots, \alpha_{r,T}\right) = softmax(q_r' k_1/\sqrt{d}, \ldots, q_r' k_T/\sqrt{d})$
  - Compute new embedding
    1. **Values**: Transform sequence embeddings with matrix $\boldsymbol{V}$: $v_t = V * u_t$
    2. Generate **new embedding** of target as weighted average: $h_r = (\alpha_{r,1} v_1 + \cdots + \alpha_{r,1} v_T)$
- The matrices Q, K, V are **parameters**. They are adapted during training.

Self-Attention: Evaluate Association of Words by Scalar Product

- Attention operates on queries, keys, and values. $u_t \in \Re^d$, $t \in \{1, \dots, T\}$ are the input embeddings.
  - **queries** $q_1, \dots, q_T$. $\quad q_t = \boldsymbol{Q} * u_t,\ q_t \in \Re^d$
  - **keys** $k_1, \dots, k_T$. $\quad k_t = \boldsymbol{K} * u_t,\ k_t \in \Re^d$
  - **values** $v_1, \dots, v_T$. $\quad v_t = \boldsymbol{V} * u_t,\ v_t \in \Re^d$

  $Q, K, V \in \Re^{d \times d}$ are parameter matrices

- Self-attention computations for all tokens $t = 1, \dots, T$ and reference token $r \in \{1, \dots, T\}$:

$$s_{r,t} = \frac{q_r' k_t}{\sqrt{d}}$$

Compute key-query „**association**“

$$\alpha_{r,t} = \frac{\exp(s_{r,t})}{\sum_m \exp(s_{r,m})}$$

Compute attention weights from „associations“ by softmax

$$h_r = \sum_t \alpha_{r,t} v_t$$

Compute outputs as weighted sum of values

new embedding for r-th token

- Repeat this: new embeddings $h_1, \dots, h_T$ for all tokens of the sequence

Self-Attention

Self-Attention for „bank“

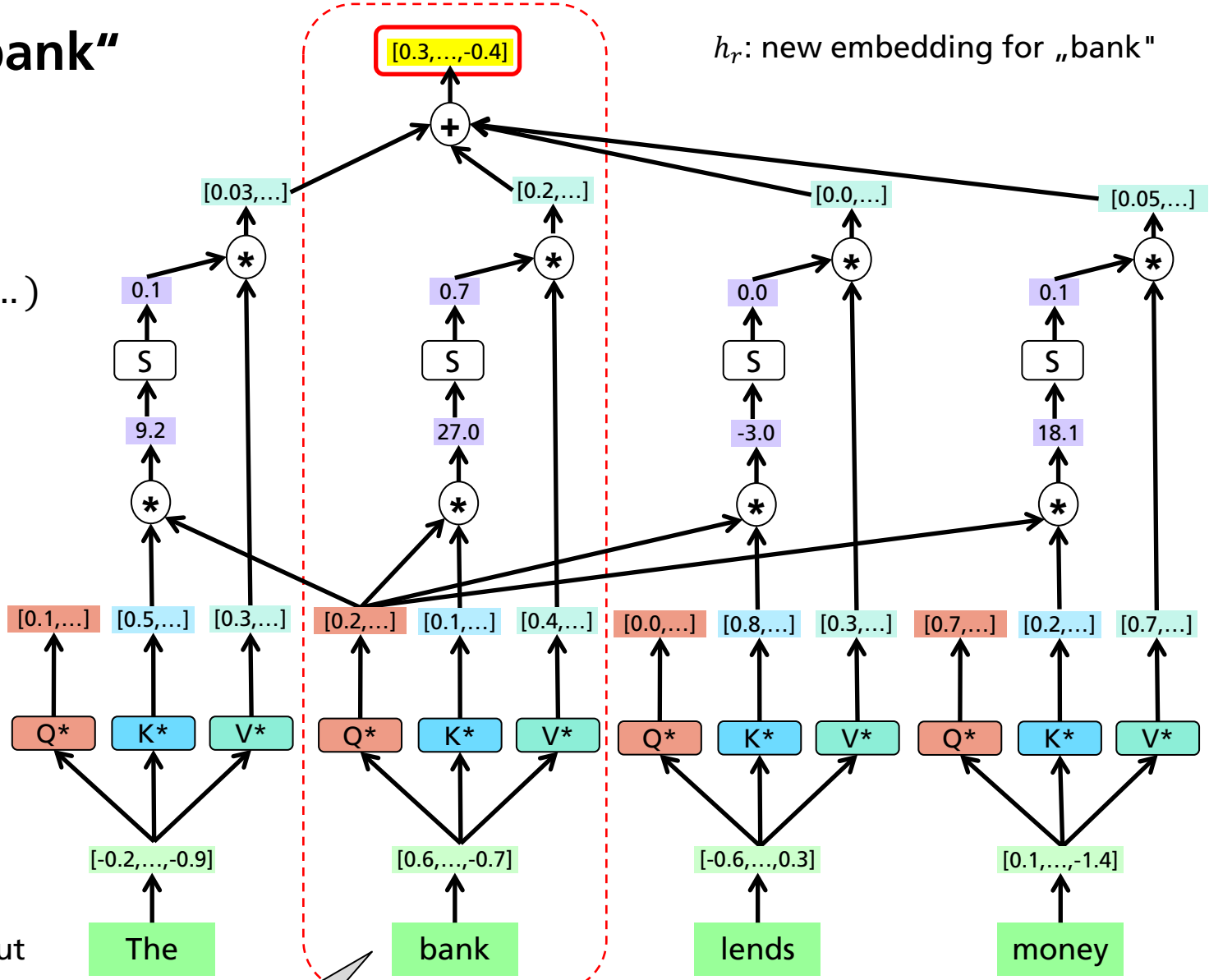
h_r: new embedding for „bank"

weighted average $$h_r = \sum_t \alpha_{r,t} * v_t$$

normalized association $$\alpha_{r,t} = \text{softmax}(s_{r,1}, \dots)$$

associations to r-th token „bank“ $$s_{r,t} = \frac{q'_r * k_t}{\sqrt{d}}$$

$$v_t = V * u_t$$
$$k_t = K * u_t$$
$$q_t = Q * u_t$$



Q, K, V are parameter matrices

[Vaswani et al. 2017]

r-th token: target token

Self-Attention

weighted average

$$z_t = \sum_j \alpha_{t,j} * v_j$$

normalized association

$$\alpha_{t,r} = \text{softmax}(s_{t,1}, \dots)$$

associations to r-th token „lends“

$$s_{t,r} = \frac{q_t' * k_r}{\sqrt{d}}$$

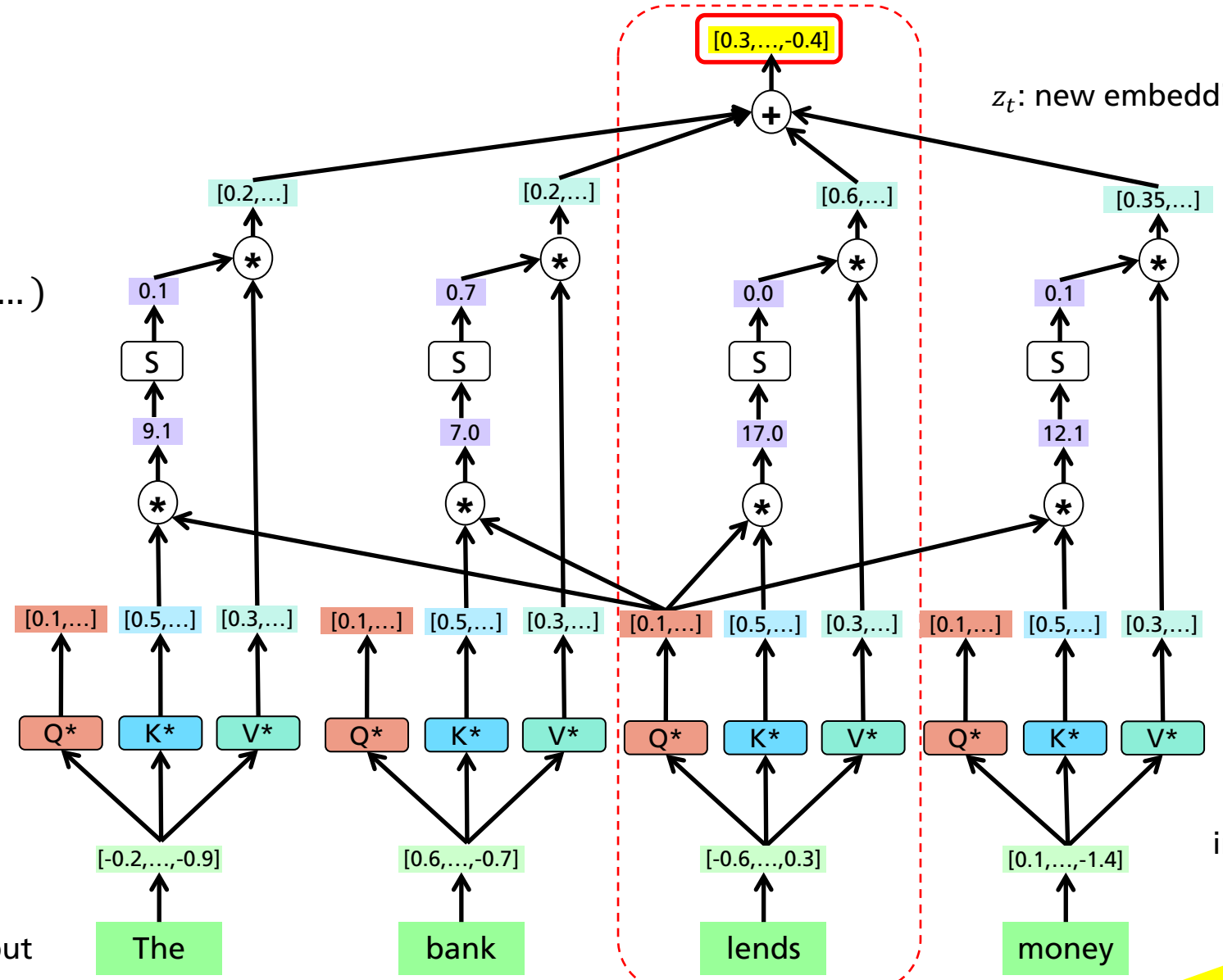
$$v_t = V * u_t$$

$$k_t = K * u_t$$

$$q_t = Q * u_t$$

Embeddings u

z_t: new embedding for „lends"



Q, K, V are parameters

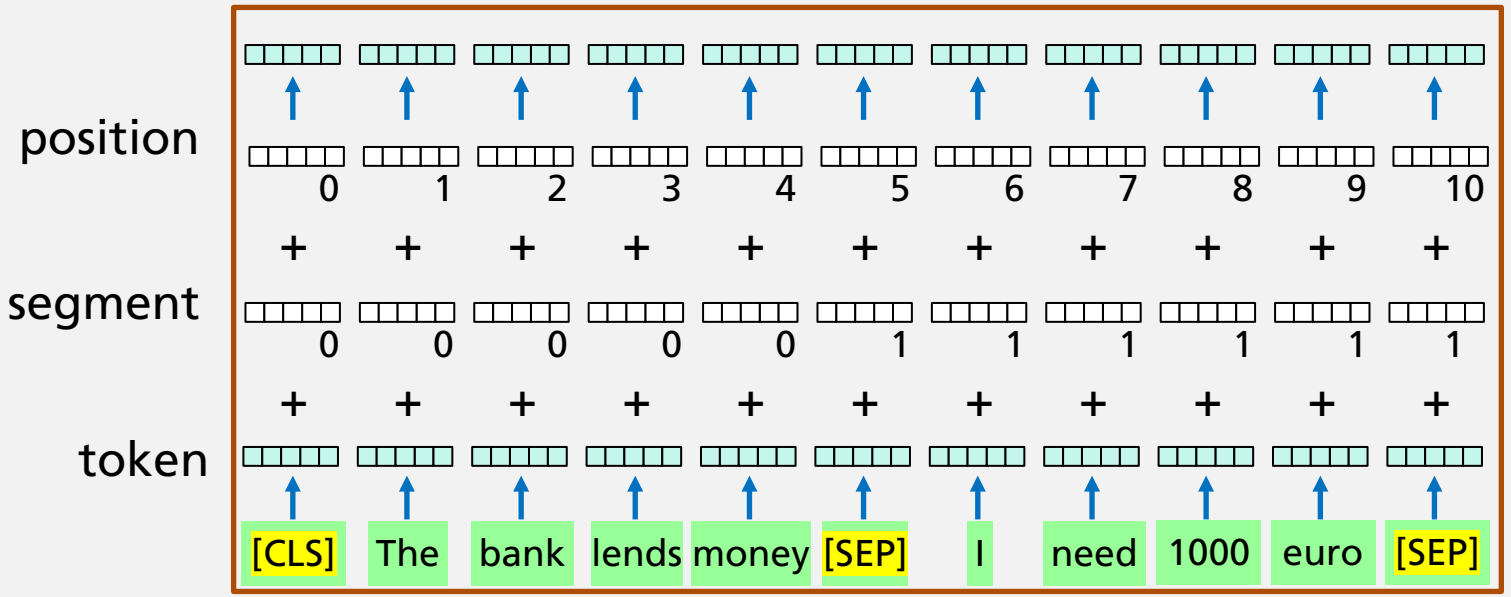
input embeddings are parameters

[Vaswani et al. 2017]

Repeat for all inputs

Transformer: Training Details

- Additional tokens: [CLS] Beginning of text, [SEP] separate two sentences / paragraphs
- Segment: marks two sentences / parts of the input (optional)
- **Position embeddings**: Need information on position of a token in the text
  ➔ use trainable position embedding, add to token embeddings
- Input embeddings: Sum of 3 embeddings



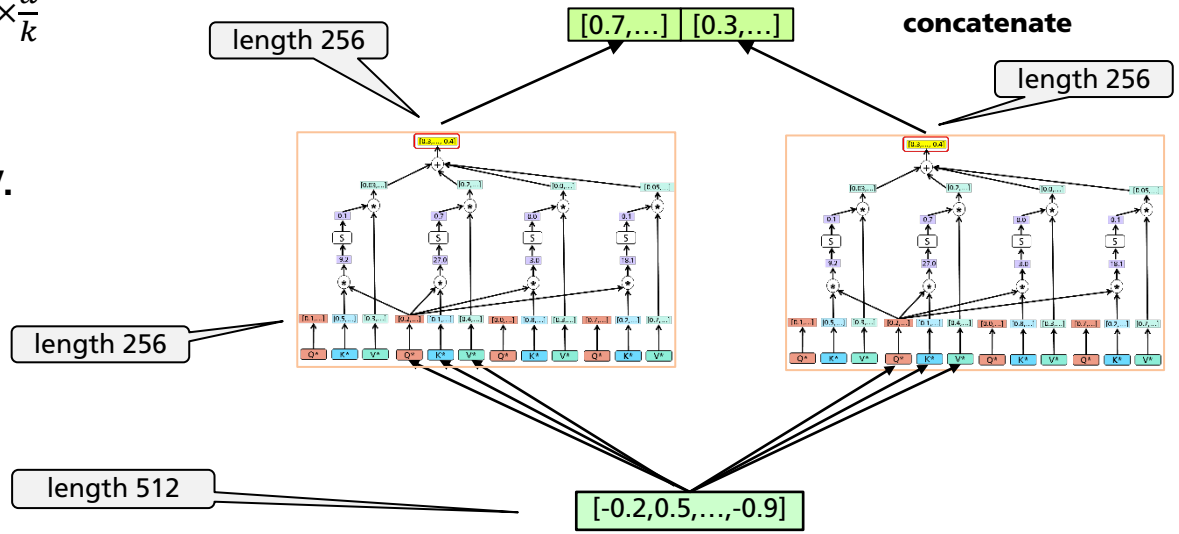
G. Position embedding

Self-Attention: Compute Different Attentions in Parallel

- **Attention between different aspects of embeddings**
  - **Key**: $k_t = K * u_t$ **Query**: $q_t = Q * u_t$ **Value**: $v_t = V * u_t$
  - K, Q, V allow different aspects of the u_t vectors to be used/emphasized in each of the three roles.

- **Multihead self-attention:** use k different parallel attentions in one layer with different matrices $K_i, Q_i, \mathrm{V_i}$ ➔ cover different types of similarities
  - Concatenate resulting vectors to new "embedding"
  - Reduce dimensions accordingly: $K_i, Q_i, \mathrm{V_i} \in \Re^{d \times \frac{d}{k}}$
  - Each head "looks" at **different things**, and constructs key, query, value vectors differently.
  - Same amount of computation as single head self attention!

if $d = 512$ and $k = 2$ then $\frac{d}{k} = 256$



G. Multihead Self-Attention

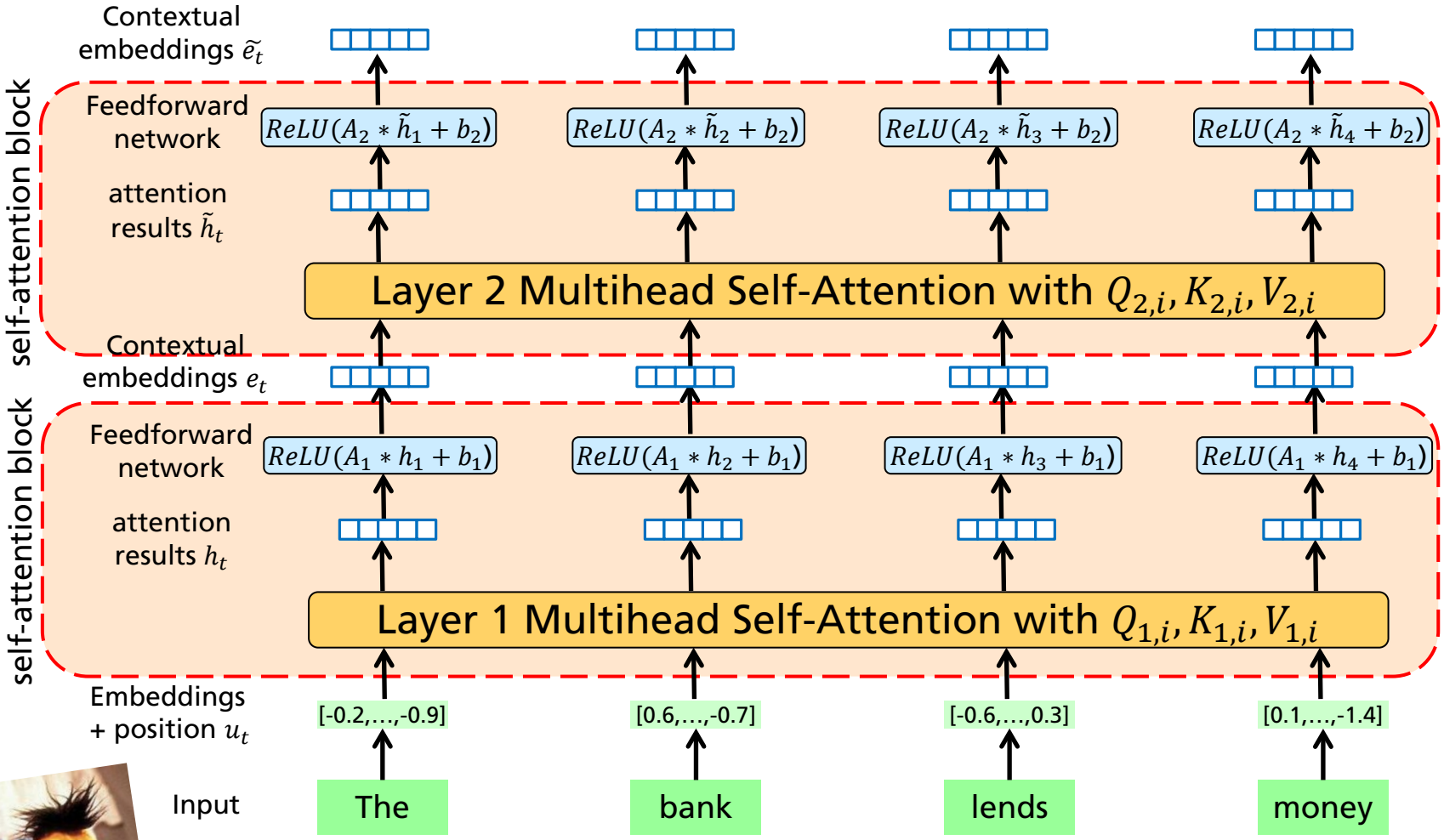
Stacking Self-Attention Layers

[Devlin et al. 2018]

- No nonlinearities in self-attention:
  - ➔ add a feedforward network after each self-attention layer

$$ReLU(A * h + b)$$

- add new self-attention block with different $Q_{2,i}, K_{2,i}, V_{2,i}$ and A_2, b_2
  ➔ catch different aspect of embeddings
- efficient computation of attentions in a layer
- create final **contextual embeddings**

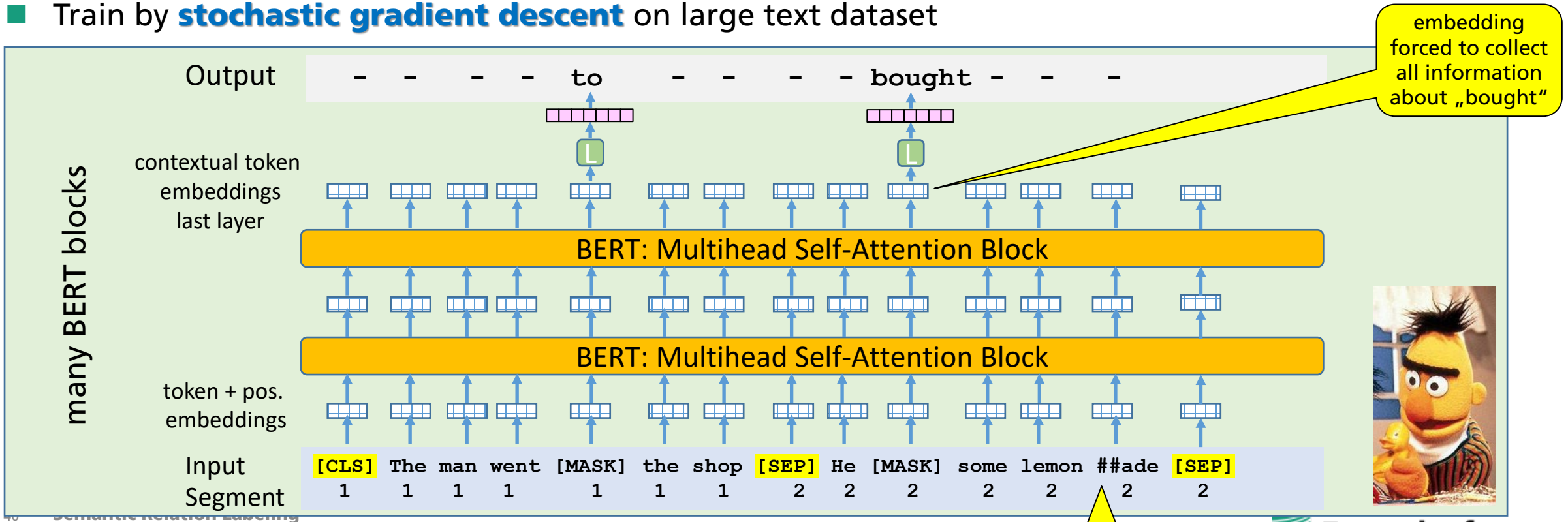


Training: Predict Masked Words

Changing input

- 80% of the time tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

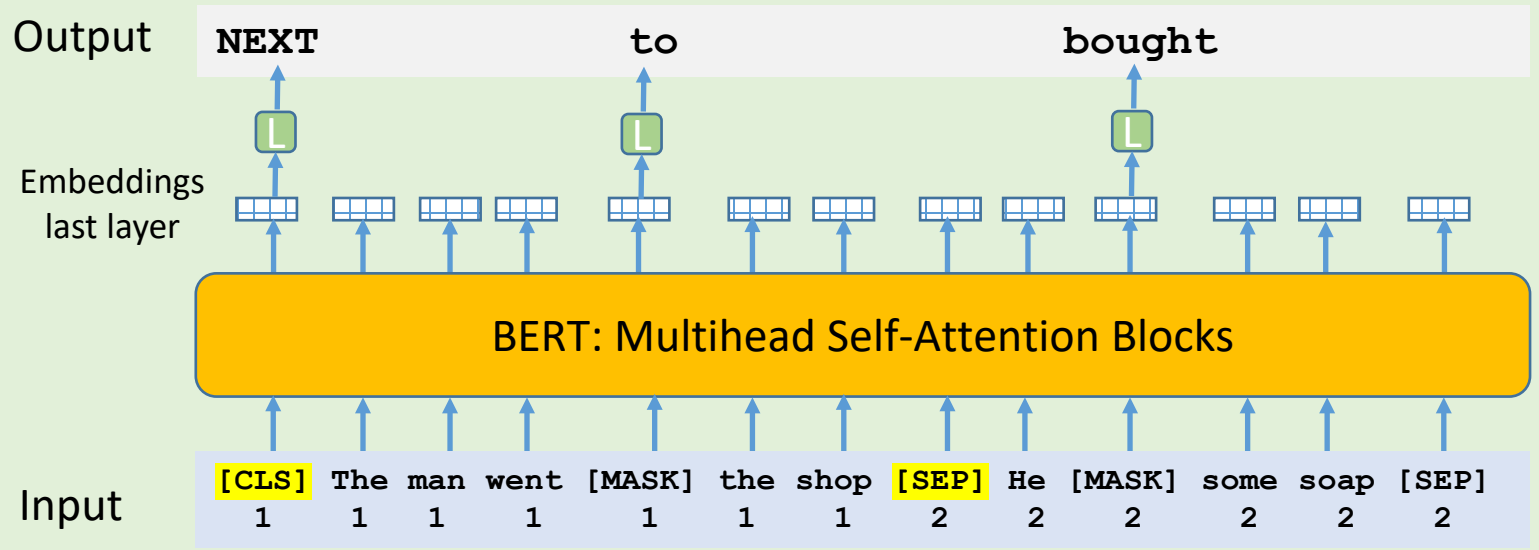
- Start token [CLS]. Finish first and second text by [SEP]
- Randomly replace 15% of words by [MASK]
- predict the **masked tokens** by logistic regression L using last layer embeddings
  Masked Language Modelling
- Train by **stochastic gradient descent** on large text dataset

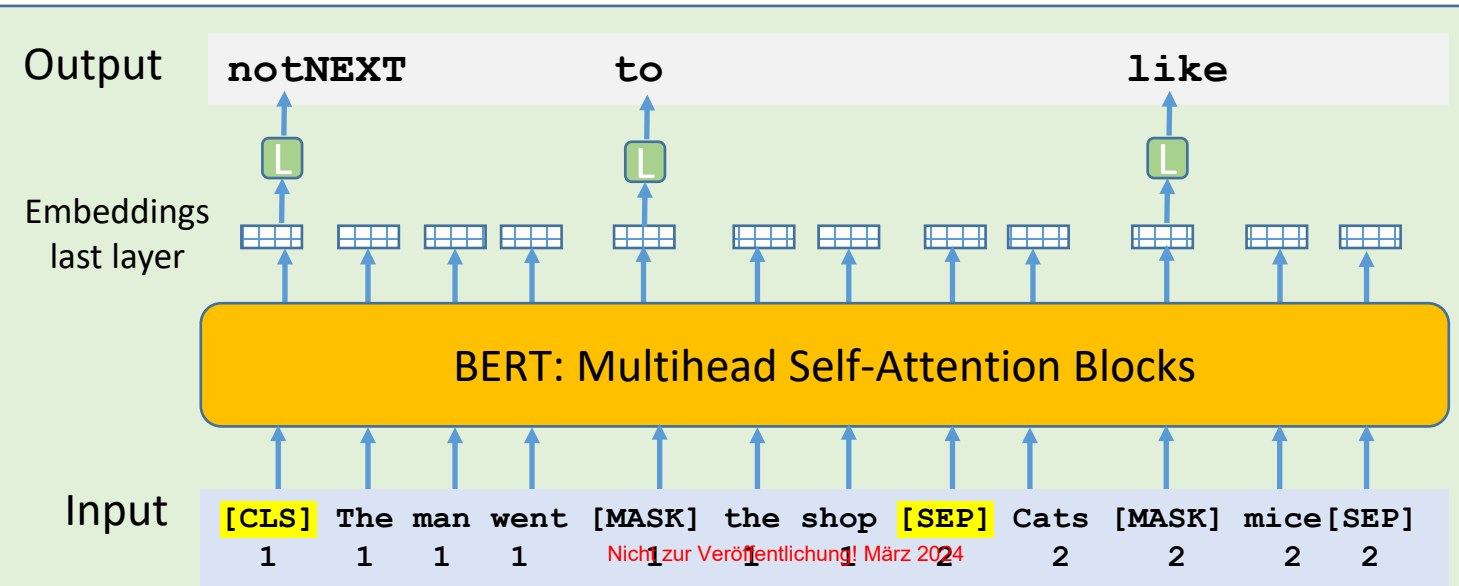


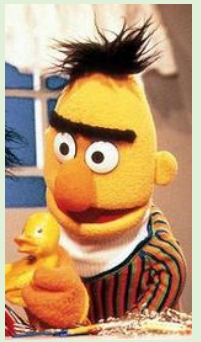
G. Masked Language modelling G. BERT architecture

Bert and Ernie in the 1978 TV special *Christmas Eve on Sesame Street* (a screenshot). Copyright owner is the Children's Television Workshop. Cropped / fair use

Pretraining: Next Sentence Detection





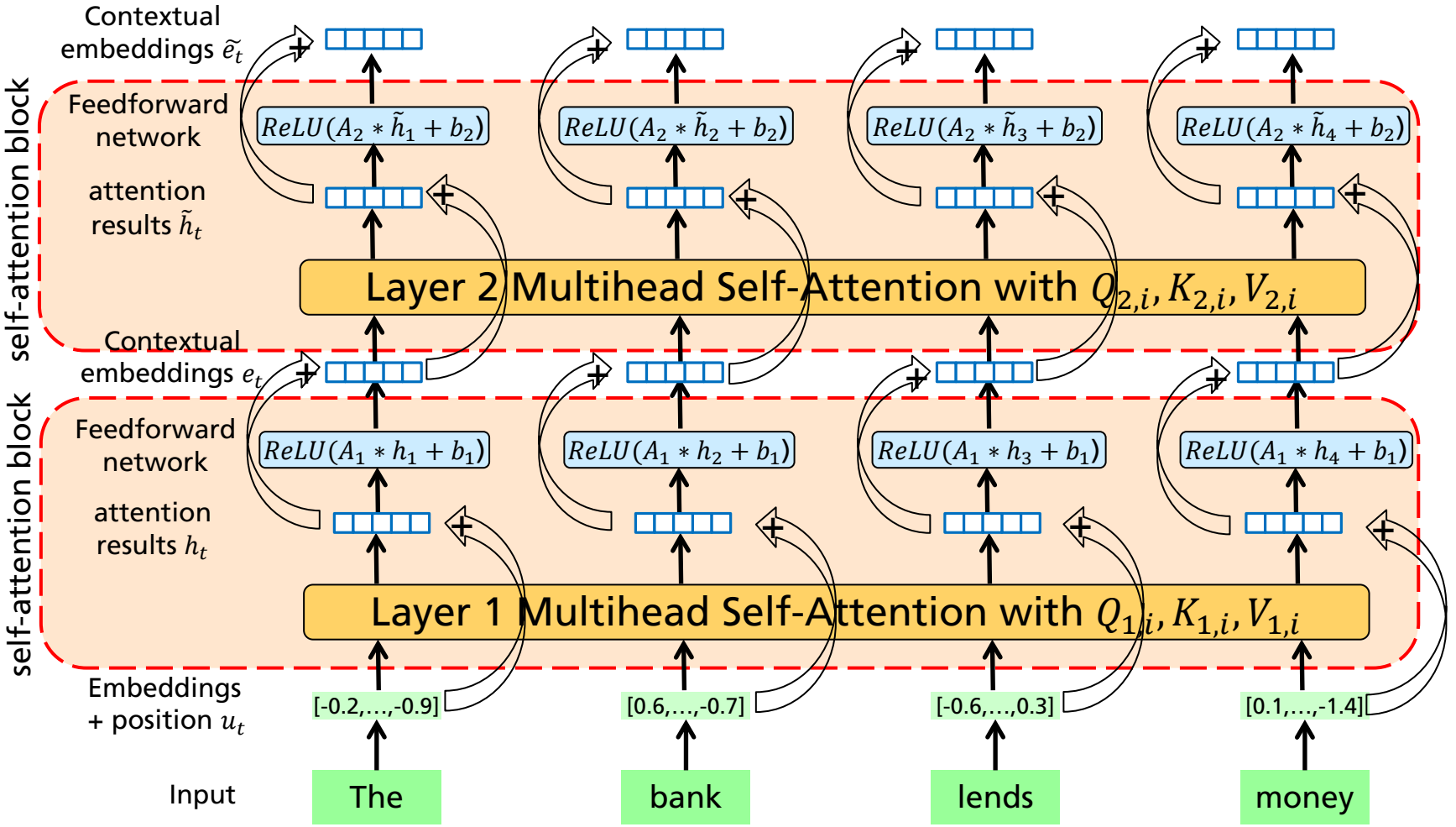


Bert and Ernie in the 1978 TV special *Christmas Eve on Sesame Street* (a screenshot). Copyright owner is the Children's Television Workshop. Cropped / fair use

Improving Optimization: Residual Connections

[Devlin et al. 2018]

- **BERT Architecture**
  12 layers
- Optimization with many layers very difficult
  ➔ gradients disappear
- At start:
  network output
  very small
- **Residual Connections**:
  add input to output
  ➔ identity connection to last layer
  ➔ gradients do not vanish
- Later: network corrects indentity mapping

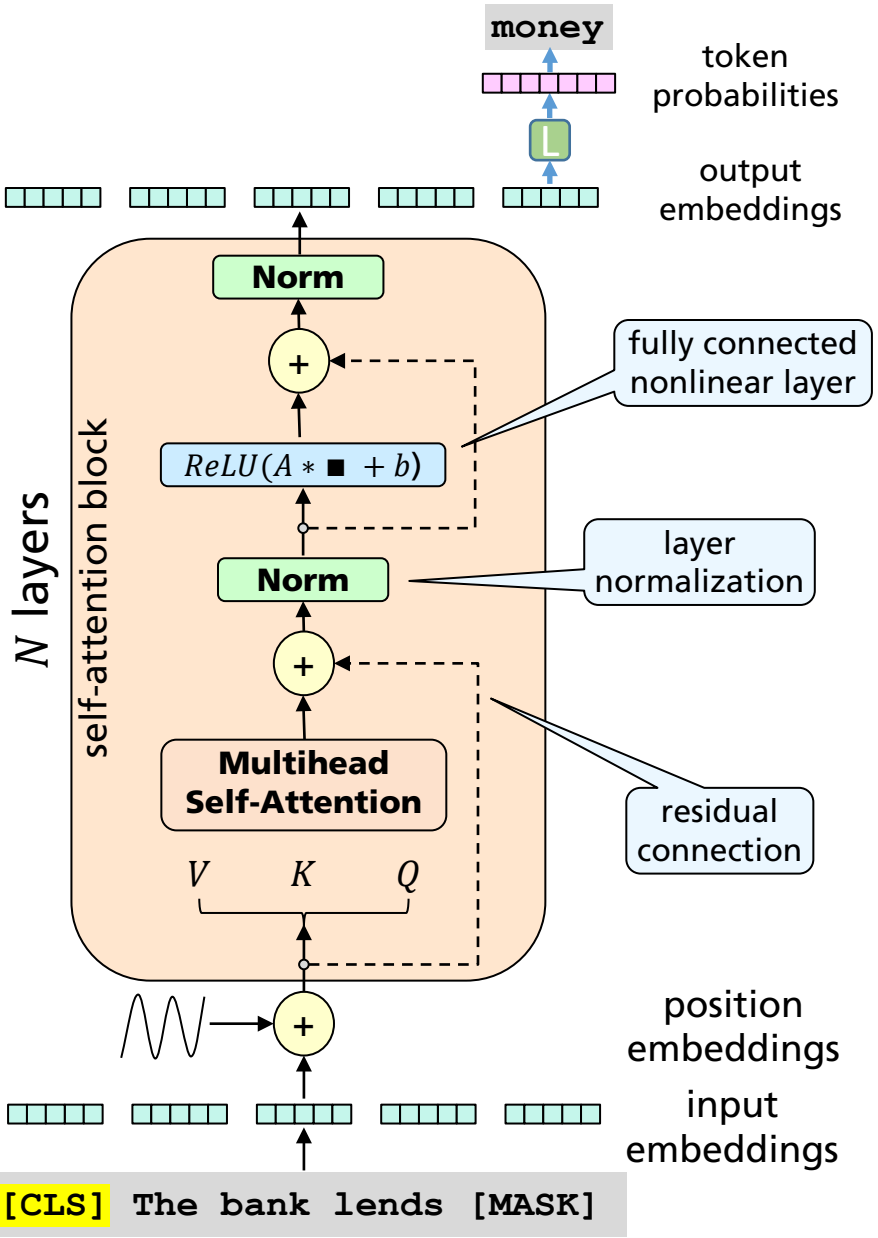


Self-Attention: Tricks to improve Optimization

- **Architecture** contains 12 layers, each with 12 self-attention heads [Devlin et al. 2018]
- **Residual Connection**: create a „bypass“ parallel to attention block: easier to train
- **Layer Normalization**: Reduce variation in hidden vector values by normalizing to unit mean and standard deviation within each layer. [Ba et al., 2016]
  - Well-behaved gradients, regularization

Training: backpropagation of derivatives

- for masked tokens
- for next sentence prediction
- modify parameters V, K, Q, A, b in different heads and layers



G. Attention regularization

BERT: Why these prediction tasks?

- Masked word prediction
  - embeddings have to capture meaning of potential word at this position
  - can use information from **left** and **right** words similar to bidirectional LSTM
  - can use information from words **far away**

- **What can we learn** from reconstructing the input sentence?

The cat sat on ___ mat.	➔ syntactic information: here may be the, my , …
Berlin is the capital of ___.	➔ content information: geographic relations
I went to the ocean to see fish, seals, and ___.	➔ other type of sea life
Peter touched the kettle over the fire, and it was ___.	➔ common sense knowledge

- Website to explore the predictions of BERT 

BERT: Pretrained models

- need a large corpus for pretraining
  - BooksCorpus 800M words
  - English Wikipedia 2500M words

[Devlin et al. 2018]

	# layers	# heads	# attentions	hidden size	# param	training time
BERT base	12	12	144	768	110M	4 days on 4 TPUs
BERT large	24	16	384	1024	340M	4 days on 16 TPUs

- BERT embeddings
  - are extremely good to encode meaning of tokens
  - will be discussed in the next later in depth

04-b

Semantic Similarity of Words & Documents

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

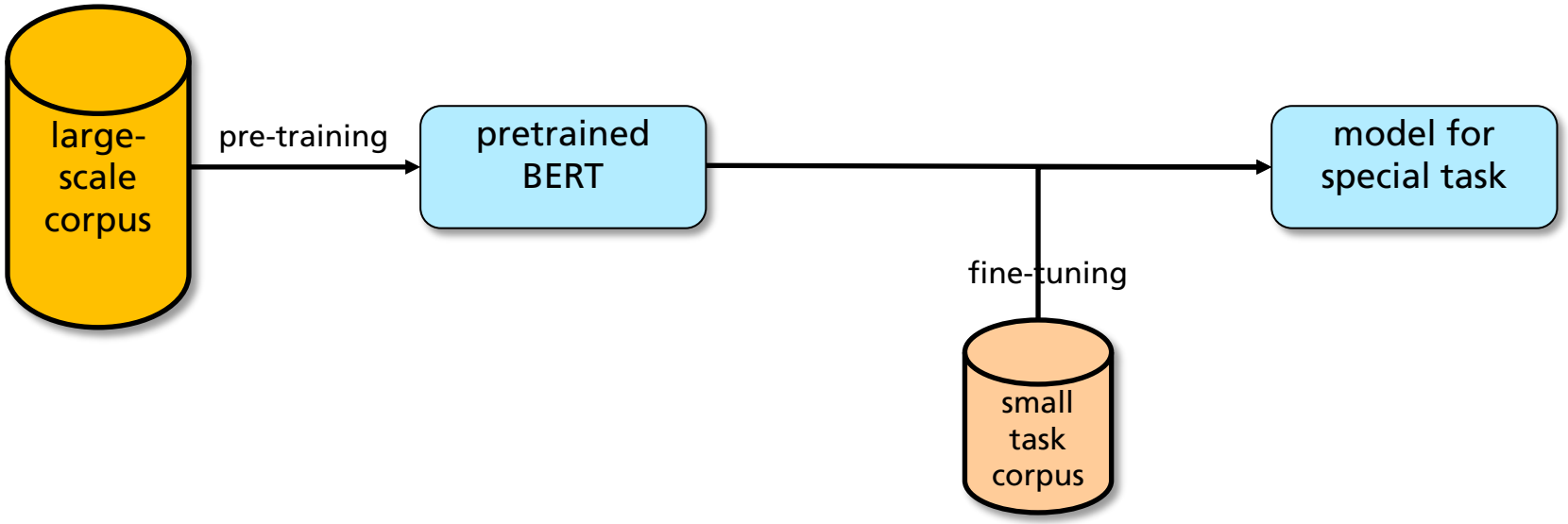
New Approach to Learning: Transfer Learning

- Train a language model on a large corpus without annotations

  ➔ Get context sensitive embeddings: know the **structure of language**

- train for some special task with new logistic regression

  ➔ Transfer learning



- knowledge learned from first task is **re-used** in a related task
  - if model is large enough, knowledge from first task is **not** destroyed (catastrophic forgetting)

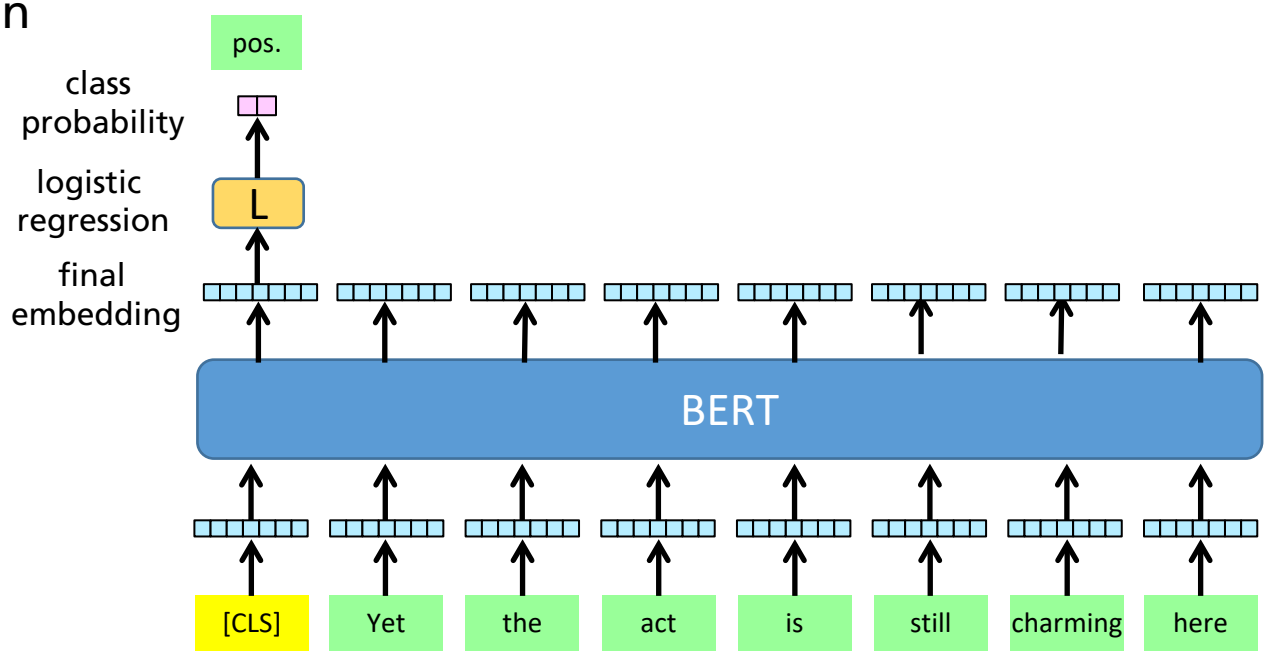
G. transfer learning

Sentiment Analysis Tasks

- Simplest task: **Sentiment classification**
  - Is the attitude **of this text** positive, negative, or neutral?
  - Different classes of sentiments: *Like, love, hate, value, desire,* etc.
    ➔ use document classification methods / LSTMs
- IMDb sentiment classification benchmark 📎
  - 50000 reviews from IMDB
  - Only highly polarizing reviews are considered.

BERT: Fine tuning

- Pretrained models knows the structure of language
  - syntax: how words may form a sentence
  - semantics: how to express a fact or relation
- Adapt this model to a **new** task
  - Example: classify opinion in a sentence
  - predict by [CLS] embedding: positive, negative, neutral



- Yet the act is still charming here.
  positive
- This isn't a new idea.
  negative

- IMDB data: BERT large finetuned: 95.8% accuracy
  - Current SotA: 97.4 Test F1

much better accuracy than for conventional classifiers

Fine Tuning Tasks for Text Mining for Information Extraction

- **Named Entity Recognition**
  - Mark all proper names (or entities) in a text together with their type
  - $[\text{Joe Biden}]_{person}$ went to $[\text{New York}]_{location}$
- **Sentiment Analysis**
  - Detect and classify the sentiment of a text as negative, neutral, positive
  - The lobster soup was excellent ➔ positive
- **Grammatical correctness**
  - Classify if a text is grammatically correct
  - My **school have** strict teachers ➔ incorrect
- **Entailment**
  - Determine if text B is a consequence of text A.
  - A: The bar was closed. B: You could not buy a drink in the bar. ➔ entails

Information Extraction

Extract facts from a text: entities, names, relations, properties, objects, etc.

details in text mining course

Example: Use BERT for Named Entity Recognition

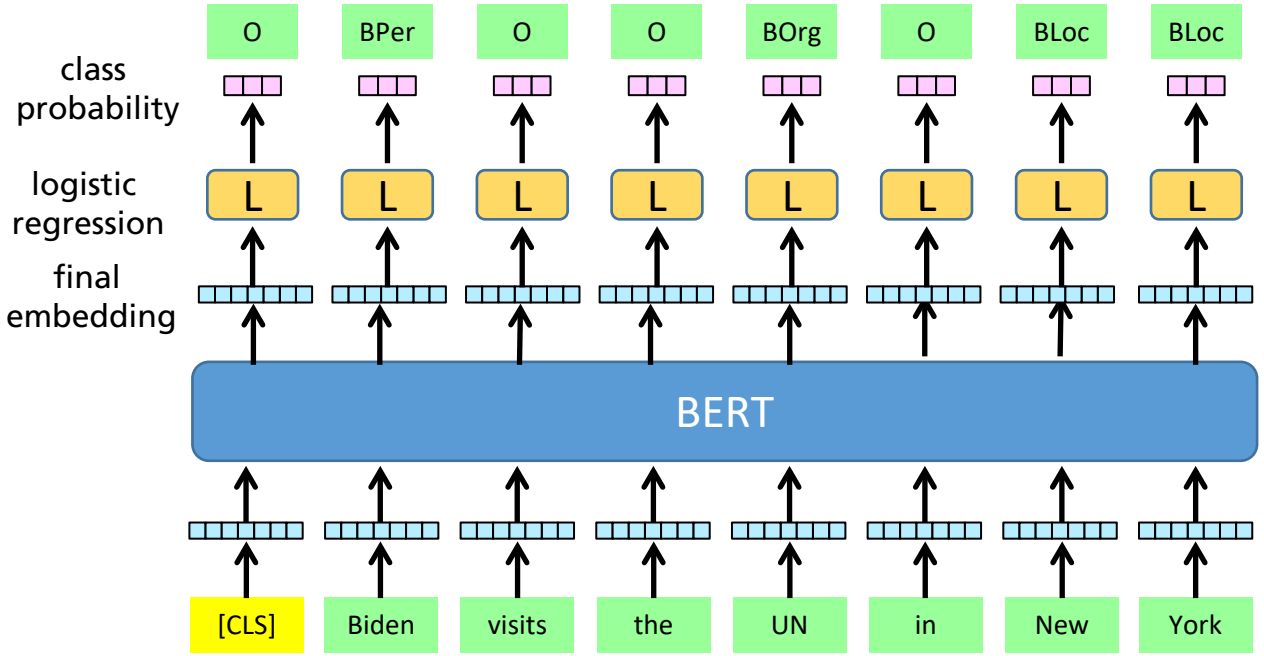
- The input is a sequence of tokens from a sentence.
- Instead of predicting the masked tokens➔ predict a label of each token
- Different types: Person (PER), location (LOC), organization (ORG), mixed (MISC), other (O)

Finetuning

- Use annotated data CoNLL 2003
- Indicate beginning of each entity by „B“
  ➔ 9 classes

Results

- Prior best model: 92.6% F1
- BERT large: 92.8% F1



G. Named entity recognition G. Sentiment analysis G. Check grammatical correctness G. Check Entailment

G. Fine tuning tasks for text mining G. Information extraction

GLUE Benchmark for Natura Language Understanding

- Test a system's ability to **understand and reason** about English texts
- Tasks solvable by most college-educated English speakers.
- No domain-specific knowledge required, e.g. medical notes or scientific papers.

Task	Description	Example
CoLA	Is the sentence grammatical or ungrammatical?	This building is than that one. ➔ ungrammatical
SST-2	Is the movie positive, negative, or neutral?	The movie is funny, smart, visually inventive, and most of all, alive. ➔ positive
MRPC	Is the sentence B a paraphrase of sentence A?	A: Today, Taiwan reported 35 new infections. B: Taiwan announced another 35 probable cases at noon. ➔ paraphrase
STS-B	How similar are sentences A and B?	A: Elephants are walking down a trail. B: A herd of elephants is walking down a trail. ➔ similar
QQP	Are the two questions similar?	A: How can I increase the speed of my Internet connection while using a VPN? B: How can Internet speed be increased by hacking through DNS? ➔ not similar
MNLI-mm	Does sentence A entail or contradict sentence B?	A: Tourist information offices can be very helpful. B: Tourist information offices are never of any help. ➔ contradiction
QNLI	Does sentence B contain the answer to the question in sentence A?	A: Which collection of minor poems are sometimes attributed to Virgil. B: A number of minor poems, collected in the Appendix Vergiliana, are often attributed to him. ➔ contains answer
RTE	Does sentence A entail sentence B?	A: Yunus launched the microcredit revolution, funding 50,000 beggars, whom Grameen Bank respectfully calls `Struggling Members.' B: Yunus supported more than 50,000 Struggling Members. ➔ entailed
WNLI	Sentence B replaces sentence A's pronoun with a noun - is this the correct noun?	A: Lily spoke to Donna, breaking her concentration. B: Lily spoke to Donna, breaking Lily's concentration. ➔ incorrect

BERT on Language Understanding Benchmarks

GLUE General Language Understanding Evaluation benchmark, is a collection of resources for training, evaluating, and analyzing natural language understanding systems.

- On **GLUE** BERT was able to lift SOTA on GLUE from 75.2% to **82.1%**
  - Remarkable success, although still below human performance of 87.1%.

- Current results on GLUE: 91.3% by a variant of BERT
  - Models far beyond human performance
  - Considered as solved
  - Replaced by SuperGLUE benchmark, BIG benchmark collection

? 04-c

Semantic Similarity of Words & Documents

Agenda

1. Introduction
2. Word Embeddings with Word2Vec
3. Implementation of Word2Vec
4. Evaluating Embeddings
5. Context-Sensitive Embeddings: BERT
6. Fine-tuning to Special Tasks
7. Summary

Summary

- **Embeddings** characterize word meanings: local neighborhood
  - Character n-gram embeddings: can cope with large vocabulary
  - Multithread implementation in Tensorflow, gensim, fastText
- BERT embeddings
  - **context-sensitive**: extremely good characterization of word meaning
  - models can be fine-tuned to specific classification tasks
- Can be used to …
  - understand very detailed nuances of language
  - show semantic content of a document
  - document search: used in all current **search engines**
  - basis for current methods for natural language understanding

Disclaimer

Responsible contact: **Dr. Gerhard Paaß**, Fraunhofer IAIS, Sankt Augustin
E-mail: gerhard.paass@iais.fraunhofer.de