Deep Learning

Sequence to Sequence Learning

Tuesday 10th December

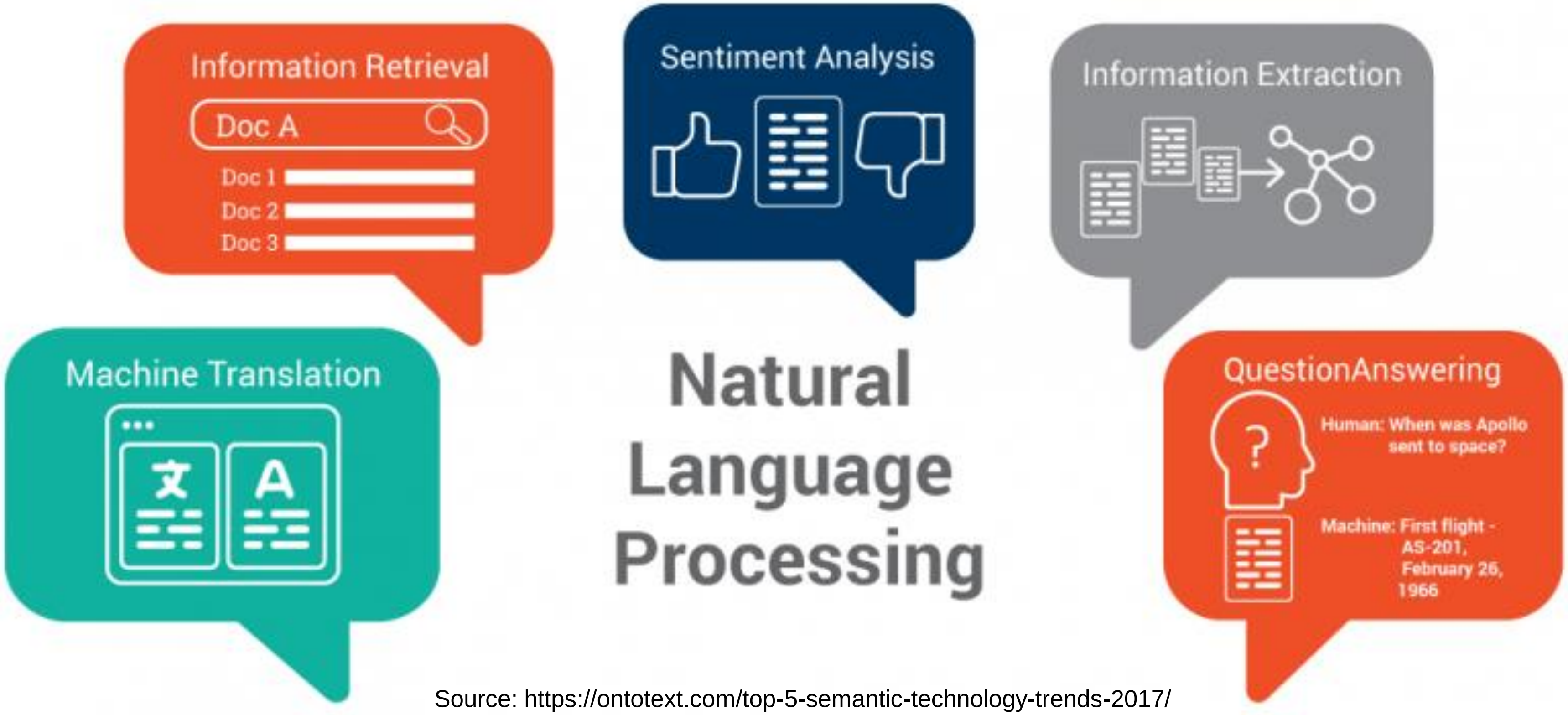
Dr. Nicholas Cummins

Lukas Stappen MSc, Dr. Ziping Zhao

Overview

- Natural Language Processing
- Sequence to Sequence Learning
- Attention Mechanisms
- Connectionist Temporal Classification

Natural Language Processing



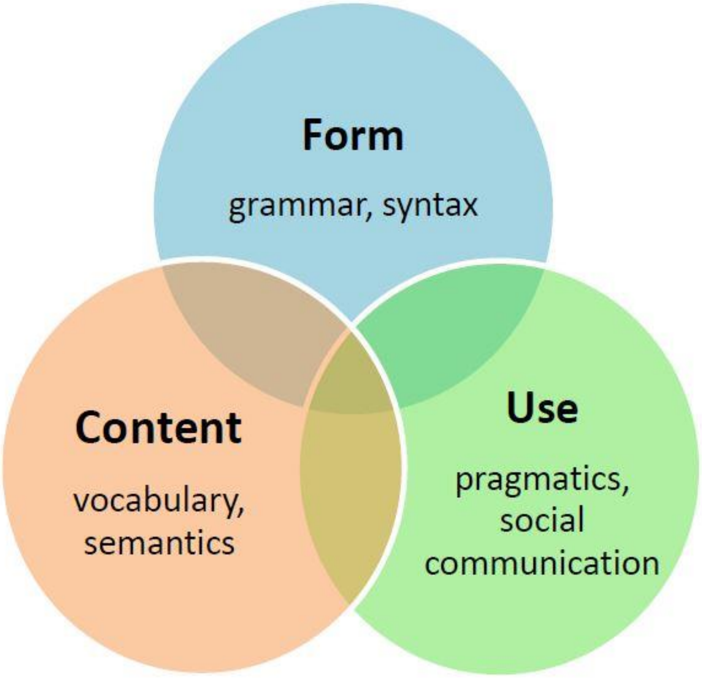
Source: https://ontotext.com/top-5-semantic-technology-trends-2017/

Definitions

- **Communication**
  - Any exchange of meaning between a sender and a receiver.
    - Intentional – e.g., inform your friend about your course schedule.
    - Unintentional – e.g., your friend interprets your facial expressions that indicate that you don't like your schedule.
- **Language**
  - A standard set of symbols (sounds or letters) and the underlying knowledge about how to meaningfully combine these symbols in order to convey a message

Any (meaningful) sentence requires an interaction of these three components of language

- **Content**
  - The structure of a language
  - The component of language that relates to meaning
- **Form**
  - The meaning of a language
  - Conventions for organising word structure and word order
- **Use**
  - Goals and social aspects of a language
  - Choosing between different combinations of words and sentences



Source: http://www.gameplanhq.com.au

What is Natural Language Processing?

Definition: Program a computer to process, analyse and understand "natural" human language.

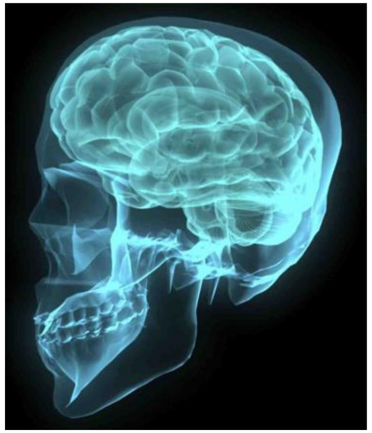
Fields: Computer Science, Linguistic, Artificial Intelligence

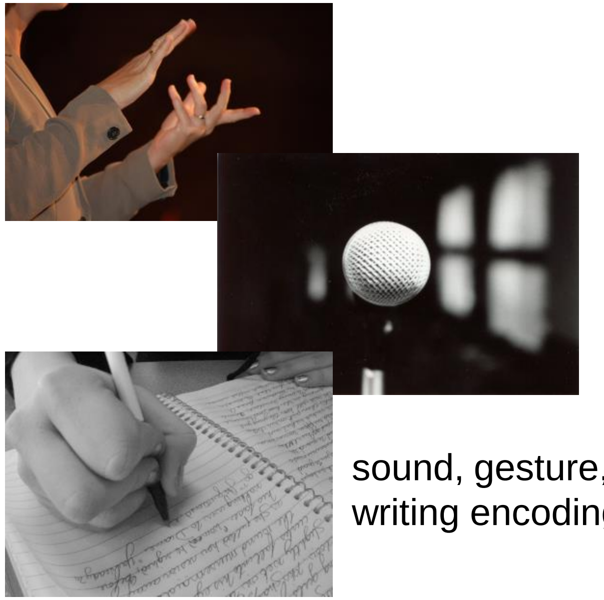
Applications: Automatic understanding of language enables the automatic execution of very complex tasks:

- Interaction between humans and machines
- Language translation
- Sentiment and entity classification
- Question answering (personal assistants e.g. Alexa)
- *and many more*

Inconsistency in human language signals

- Human language is a symbolic/categorical signaling system
- However, signal generation (brain) is continuous and communication has various encodings.


continuous activations

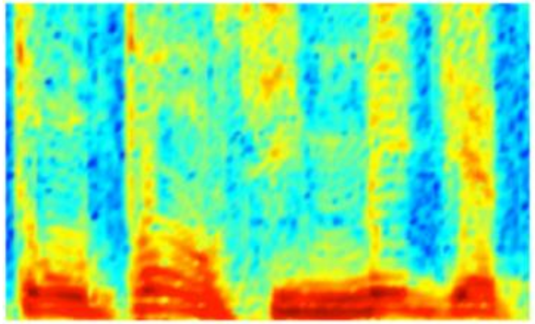

sound, gesture, writing encodings

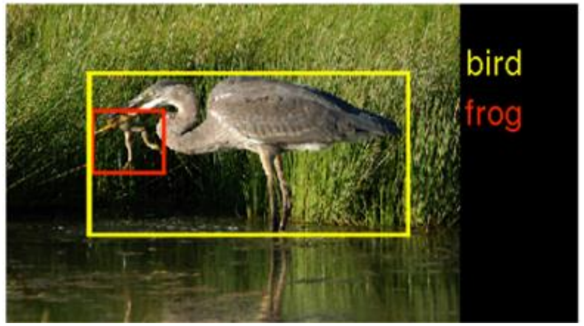

all can represent the same symbol

Inconsistency in human language signals

Image and audio processing systems work with **raw, rich and high-dimensional data** e.g.:

- Audio: Mel Frequency Cepstral Coefficients, Dense Audio Spectogram
- Image: Raw Pixel-Intensities





Illustrations [1]

Natural language processing systems work with high-level (meaningful), discrete symbols whereby the atomic symbols are **high-dimensional, sparse** and (often) **arbitrary**.

Source: http://www.fun-with-words.com/ambiguous_headlines

Example News Headlines

Teacher strikes idle kids

Stolen painting found by tree

Panda mating fails; veterinarian takes over

Court to try shooting defendant

Iraqi head seeks arms

Australian English

Carrying on like a pork chop

Blowing the froth off a few

Few roos loose in the top paddock

Fair shake of the sauce bottle

A head like a dropped pie

Straight to the pool room

Tickets on yourself

[2]

Why is human language hard to learn (for computers)?

- **Language** usage is **manifold and ambiguous**
- "**Understanding**" language means using not only linguistic knowledge but also world, contextual, **situational knowledge**
- **One symbol** can be represented by **many different encodings**
- The symbolic space of vocabulary is huge
  - Hard to search efficiently, **very sparse**

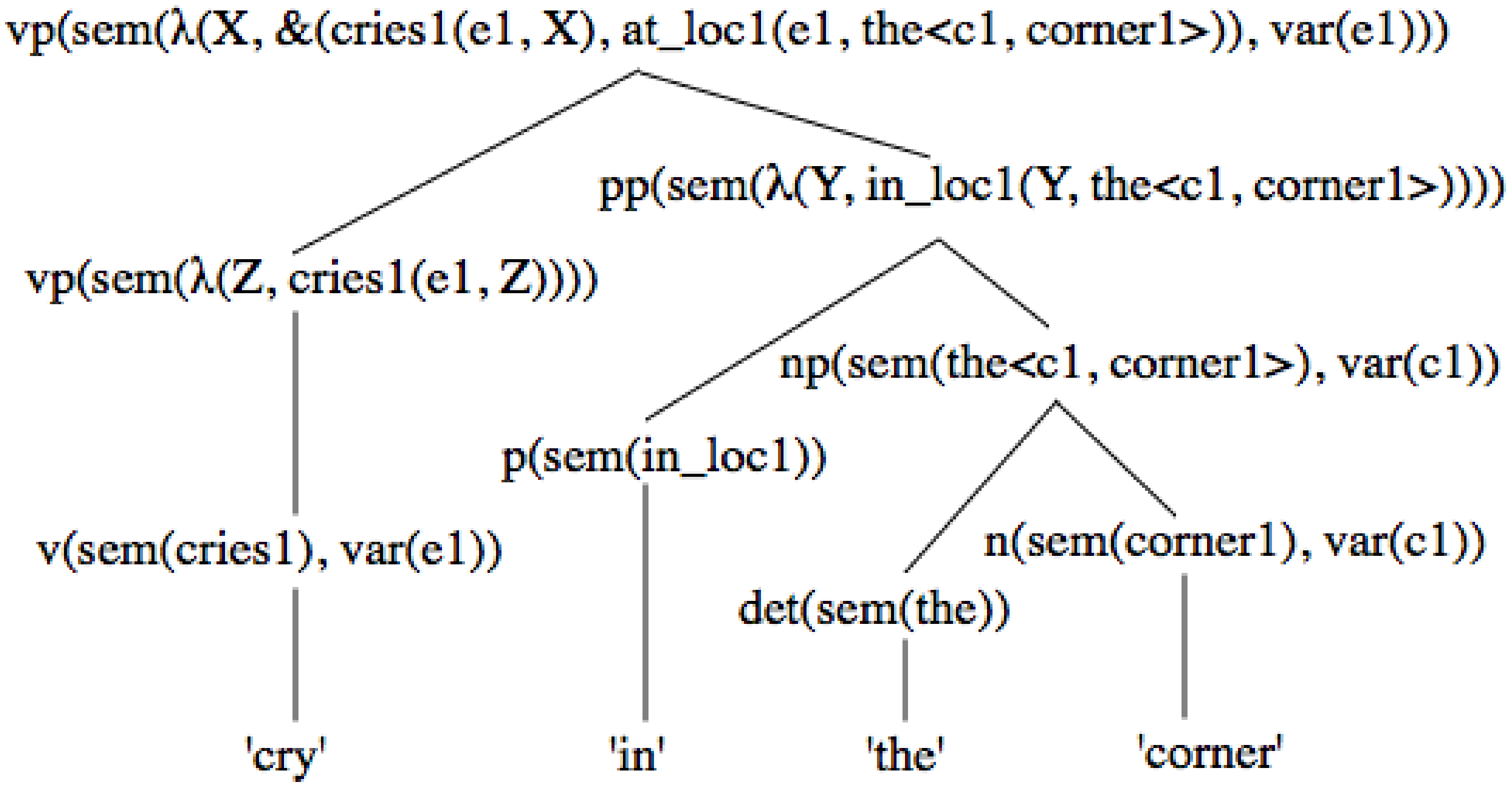
Discrete Language Models

Discrete Language models are a conventional approach to “learn” human language. For example:

- **Word meaning**: Using the morphology of words, prefix - stem - suffix e.g. un - educate - (e)d
- **Context of occurrence**: Bag-of-words approaches (e.g. Bayes Theorem), ignore word order or handcrafted prior probabilities of word weights
- **Semantic understanding**: Lamda calculus of handcrafted functions and (grammatical) composition operations results in a parse tree of a sentence, “positive” & “negative” word lists predict sentiment
- **Handle facts**: Regular expressions, rules

Parsing using logic

- e.g. lambda functions for syntactic analysis of a sentence.



[3]

Discrete Language Models

First attempts to vectorise words

- No. of words in dictionary = size of symbolic vector
- Zero vector for every word in the vocabulary
- “1” for the particular symbol e.g. “lecture” number 7 of the dictionary: [0 0 0 0 0 0 1 0 0 0]

Symbolic Vectors

→ are very big and sparse using one-hot encoding
→ have no meanings and relationships, thus, no notion of similarity e.g. dot product “tutorial”: [0 1 0 0 0 0 0 0 0 0]T · “lecture”: [0 0 0 0 0 0 1 0 0 0] = 0

Discrete Language Models

"I am a deep learning ninja"

Disadvantages using discrete approaches:

- Much human labor required
- Missing nuances of language
- Hard to keep up to date
- Subjective
- Ignore fuzziness of language

How can we tackle these challenges?

Continuous Space Language Models

Solution: Use continuous representations of words.

Ingredients:

1. **Distributional Hypothesis**

Linguistic items with similar distributions have similar meanings.

2. **Vector Space Model**

Words embedded in a continuous vector space…
... (+ Distributional Hypothesis) are semantically embedded nearby each other.

Continuous word embedding:
Do not just represent a word by figures - represent the meaning of a word!

Continuous Space Language Models

By **understanding the company a word keeps**, we can understand the **meaning of the word** and **how to represent** it. Thus, the dynamic context in which a word appears is decisive.

Tobacco smoke, for example, causes 90% of lung cancer.

https://en.wikipedia.org/wiki/Tobacco_smoking

Denotation by context similarity

Tobacco smoke can combine with other carcinogens present within the environment in order to produce elevated degrees of lung cancer.

https://en.wikipedia.org/wiki/Cancer

German scientists identified a link between smoking and lung cancer in the late 1920s, leading to the first anti-smoking campaign in modern history, albeit one truncated by the collapse of Nazi Germany at the end of World War

Continuous Space Language Models

By **understanding the company a word keeps**, we can understand the **meaning of the word** and **how to represent** it. Thus, the dynamic context in which a word appears is decisive.

In British academic parlance, a tutorial is a small class of one, or only a few students, in which the tutor, a lecturer, or other academic staff member, gives individual attention to the students.[1]

Denotation by context similarity

The practice in the medieval university was for the instructor to read from an original source to a class of students who took notes on the lecture. The reading from original sources evolved into the reading of glosses on an original and then more generally to lecture notes.

[23,24]

Continuous Space Language Models

Two different ways to leverage these principles:

Count-based methods

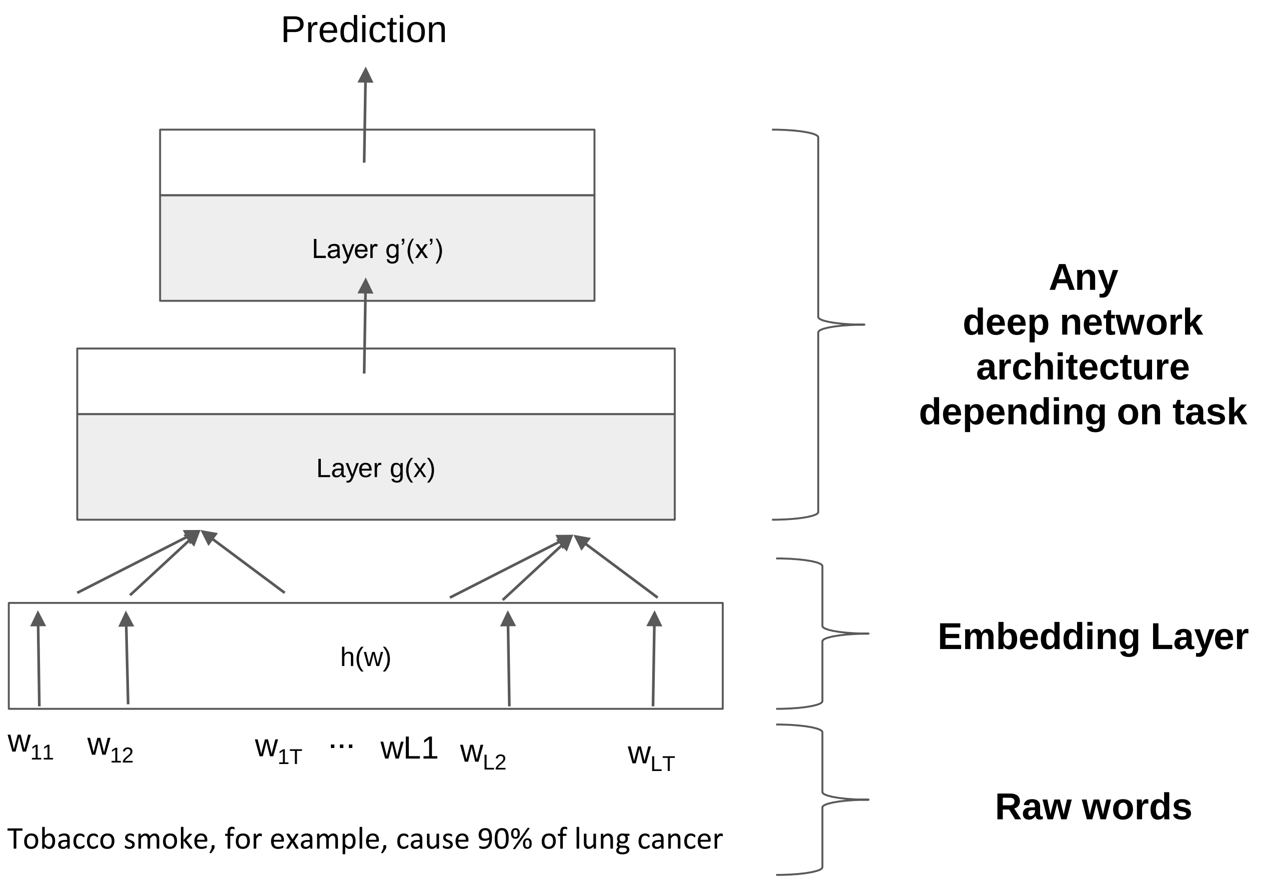
- In a large text corpus, statistics are computed of how often some word co-occurs with its neighbor words and mapped down to a dense vector for each word e.g. Latent Semantic Analysis.

Predictive methods

- Directly predict a word from its neighbors in terms of learned small, dense embedding vectors e.g. Neural Probabilistic Language Models.

DEEP LEARNING

Bottom-up NLP with Deep Learning Architecture, from word to prediction

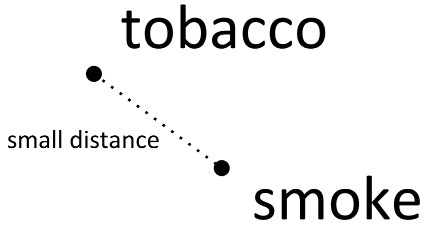


What we want:

- A dense vector = most of the values in the vector are non zero
- Good at predicting other words that appear in the context of this words
- Each of the other words are also represented by a vector (recursive)
- Using similarity measures (e.g dot product) between these vectors

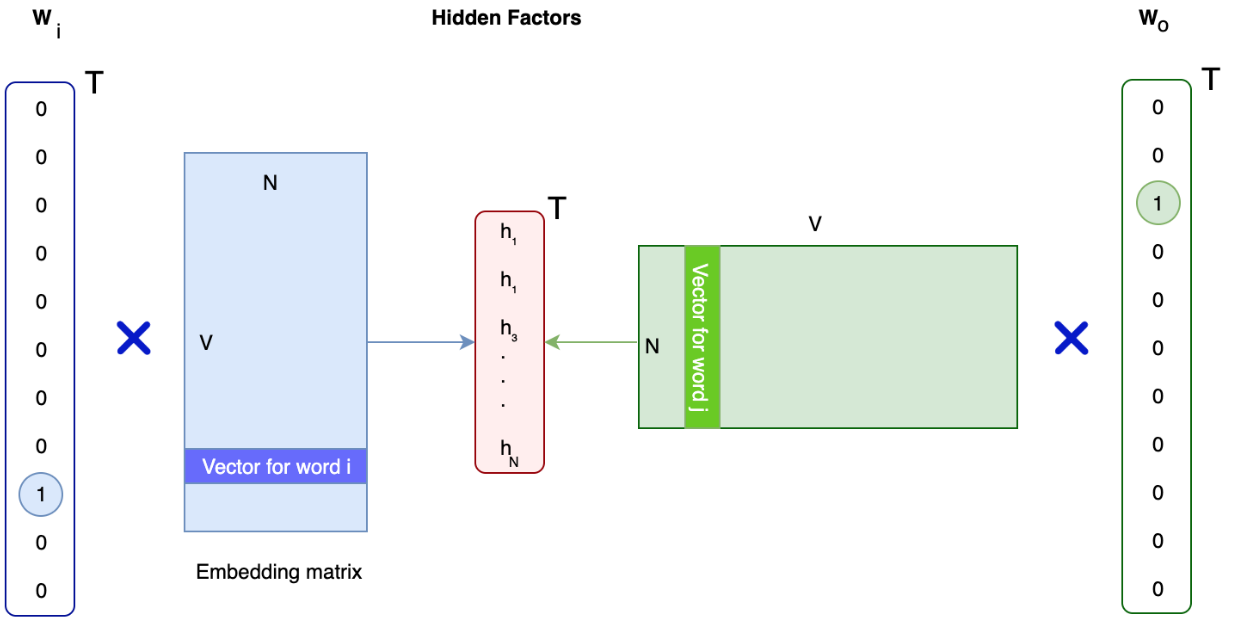
$$\text{cancer} = \begin{pmatrix} -0.2399 \\ 0.0243 \\ 0.2041 \\ 0.2185 \\ 0.4348 \\ 0.0056 \\ 0.0958 \end{pmatrix}$$

Word embeddings are the foundation and **key to success** in any NLP model!



Dense Representation

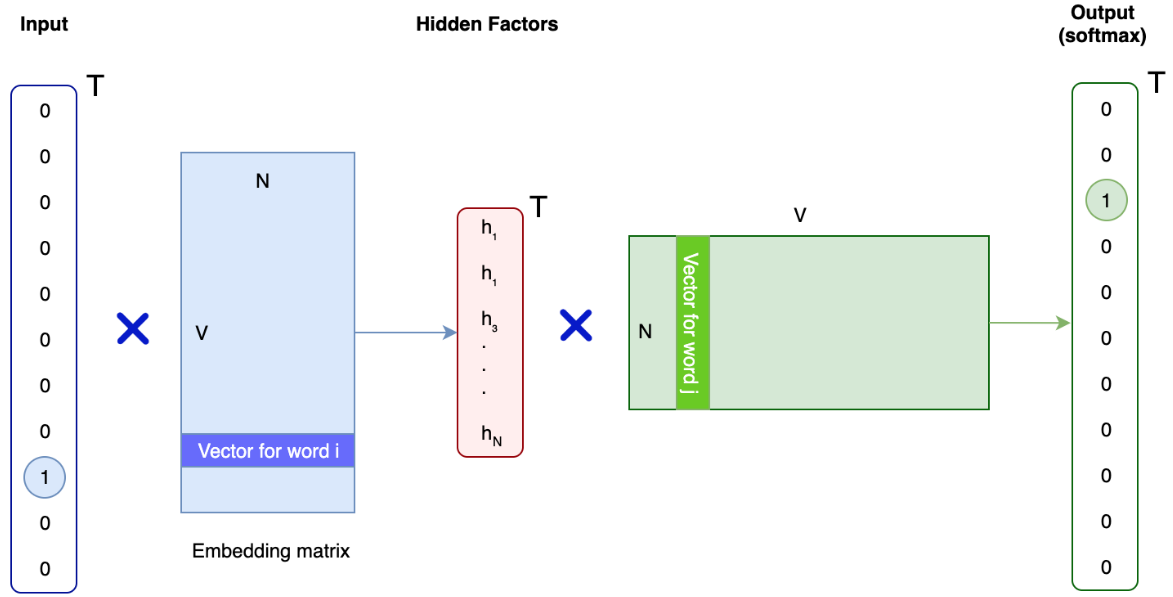
- Transform one-hot vector into a denser representation h
- h must have a lower dimensionality
- Similar words should be close to each other in the projected space



Universität Augsburg University

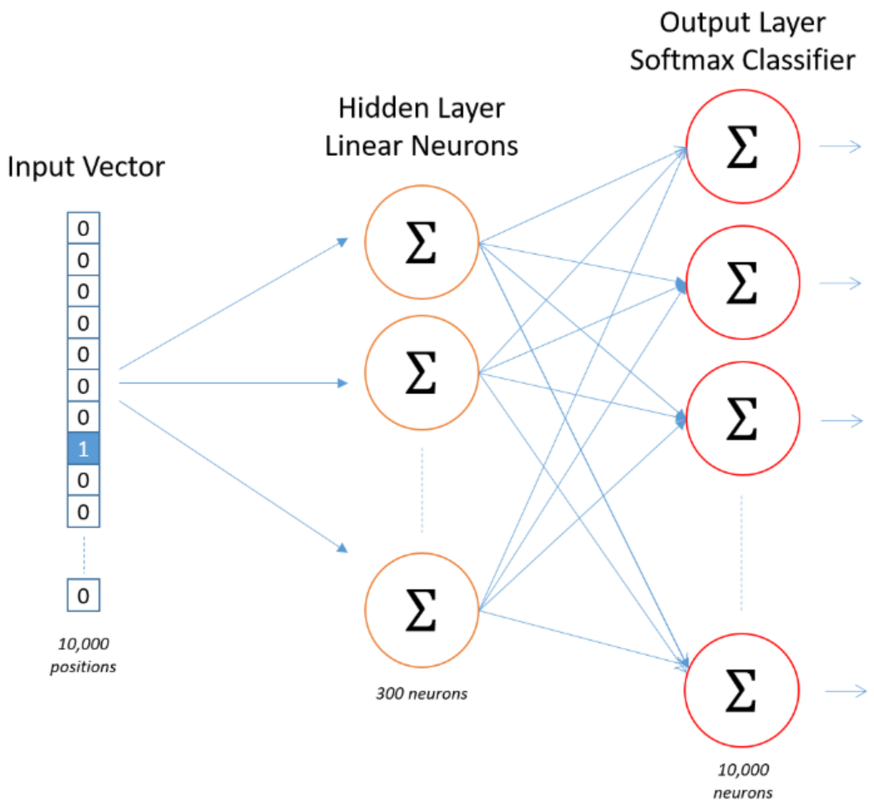
The simple approach

- Start with word w_i
- We first compute its vector representation with an embedding matrix
- We multiply it with another matrix to predict a similar word w_o
- Use a softmax function to calculate output probabilities



Network Architecture

- Training data
  - **Input:** a one-hot vector representing the input word
  - **Output:** a one-hot vector representing the output word
  - **Trainable parameters:** the weight matrix of the hidden layers – the rows of this matrix are our word vectors



Word Embedding: Word2Vec

Word2vec is a computationally-efficient predictive model for learning word embeddings from raw text.

Neighboring Words use the co-occurrence words within a context window (say within 2 words range) as our training data

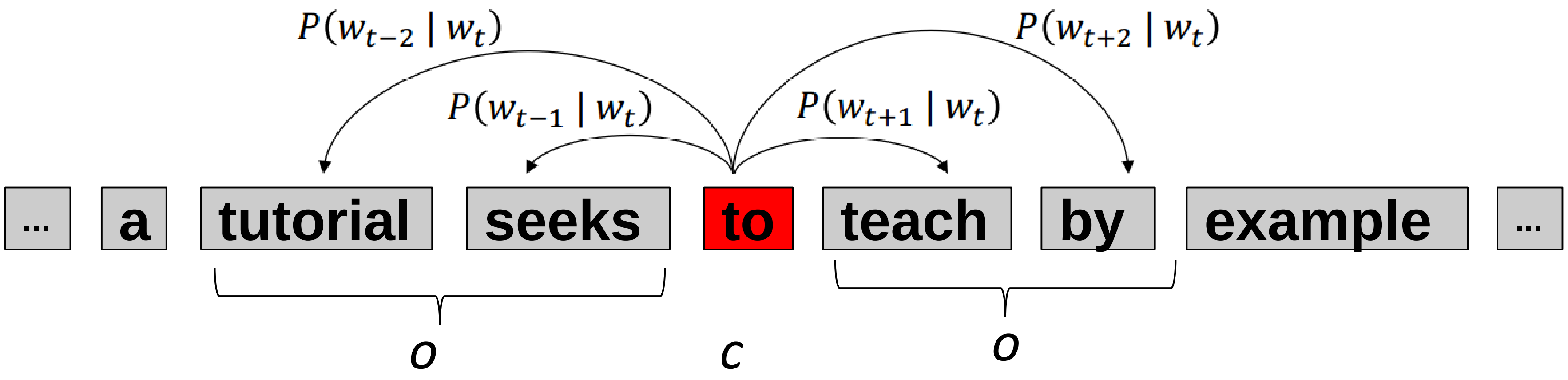
Word2Vec by Mikolov et al. (2013)[4]:

- Very large corpus of text (--> fixed vocabulary)
- Every word is represented by a vector

Example next page

Example:

- context window size m = 2
- compute $P(w_{t+j} \mid w_t)$



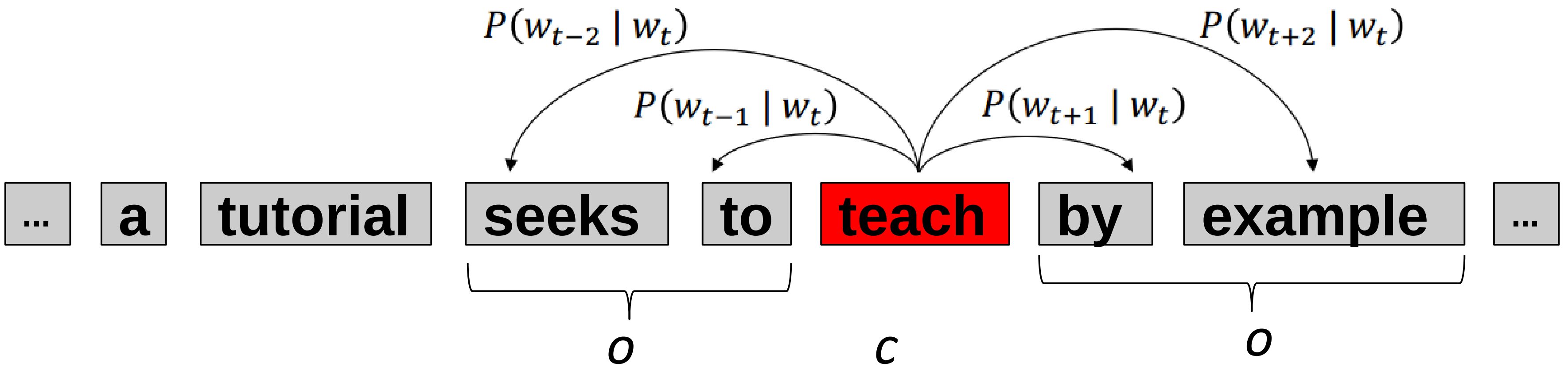
c = center word

o = context ("outside") words

Universität Augsburg University

Example:

- context window size m = 2
- compute $P(w_{t+j} \mid w_t)$



c = center word

o = context (“outside”) words

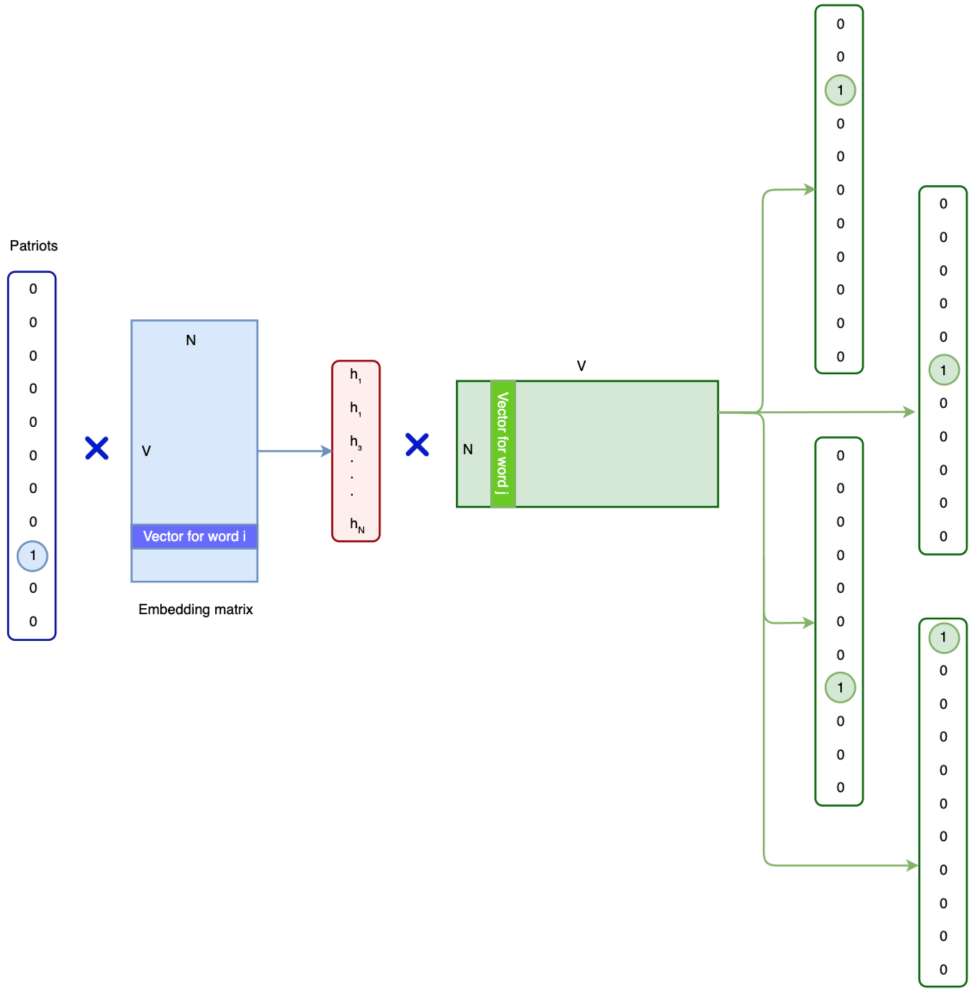
Skip-Gram model

- Model produces one prediction per possible neighbour
- **Objective function:** log-likelihood of the predicted words given the target word t

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t)$$

Training Words

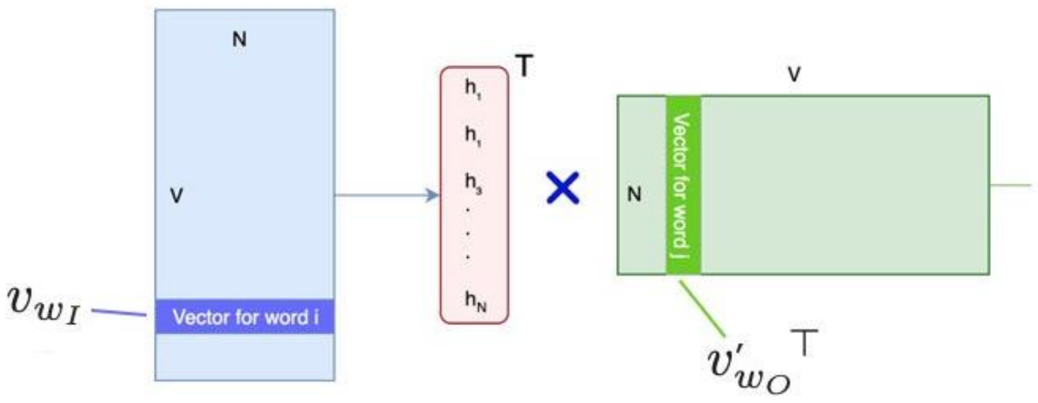
Surrounding words



Universität Augsburg University

Calculating $p(w_o|w_t)$

- Locate the corresponding row and column entries related to w_i and w_o in the corresponding matrix
- Use softmax function to compute conditional probability



Word similarity measure using dot product

$$p(w_o|w_t) = \frac{\exp({v'_{w_o}}^T v_{w_t})}{\sum_{w=1}^{W} \exp({v'_w}^T v_{w_t})}$$

Sum all possible similarities together to normalise into probability distribution

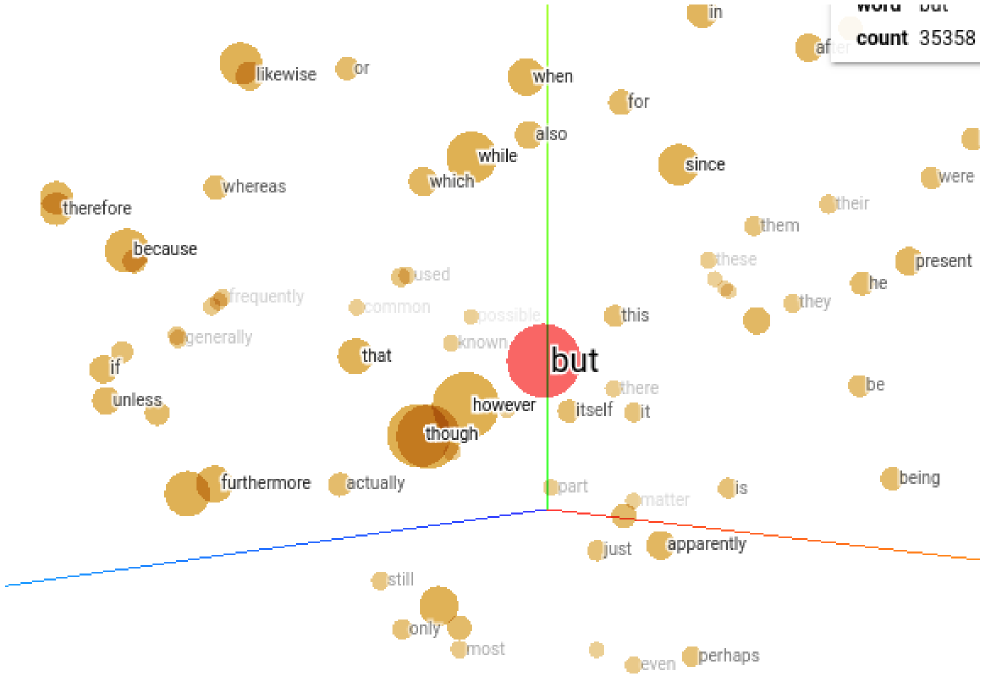
Word2Vec: Skip-Gram model

Algorithm:

1. Define a “word” window *m*, that includes left and right words as “context” words *o*
2. Go through each position *t* in the text, which has a center word *c* and context (“outside”) words *o*
3. Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa) $P(w_{t+j} \mid w_t)$
4. Keep adjusting the word vectors to maximize this probability

Word2vec: Visualisation of resulting word vectors

Distance measure to find “nearest” neighbours that have similar meaning of a word (e.g. euclidean, inner distance). Dimension reduction to 2D/3D by PCA & t-SNE.



$$\text{but} = \begin{pmatrix} 0.7399 \\ -0.2433 \\ 0.0431 \\ -0.1825 \\ -0.3468 \\ 0.5600 \\ \dots \end{pmatrix}$$

e.g. 200-dimensions

Nearest points in the original space:	
however	0.315
although	0.370
though	0.388
while	0.452
therefore	0.480
because	0.484
which	0.498

https://projector.tensorflow.org/

Word Embedding: Evaluation

Deep learning standard: Partitioning data and score test set does not work!

Other ways to measure performance of word embeddings:

Complex (and fun) **Word Vector Analogies**

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Very intuitive but no mathematical proof!

[4,5]

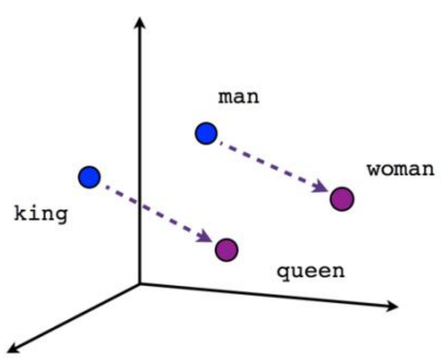
Word Embedding: Evaluation

Deep learning standard: Partitioning data and score test set does not work!

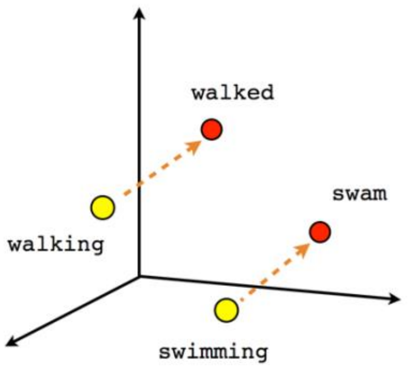
Other ways to measure performance of word embeddings:

1. **Word Vector Analogies:** Evaluate on specific subtasks that a product of human sense, for instance, evaluate intuitive semantic and syntactic relationships using cosine distance after addition/subtraction of vectors

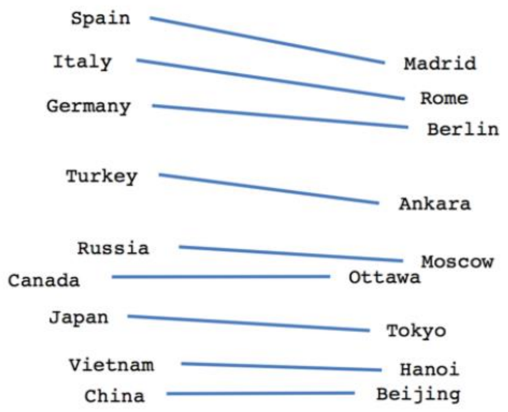
man→women as king → ???



Male-Female



Verb tense



Country-Capital

Word Embedding: Evaluation

Deep learning standard: Partitioning data and score test set does not work!
Otherwise we would lose the vocabulary of the validation and test corpus.

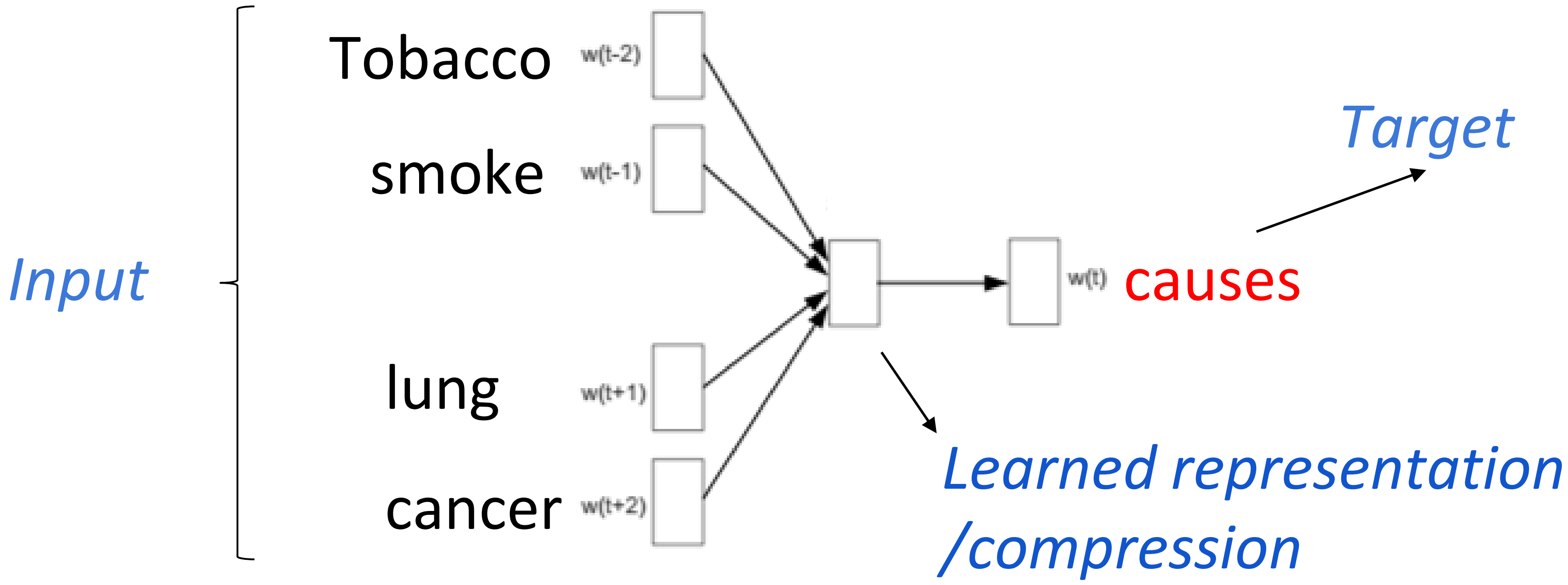
Other ways to measure performance of word embeddings:

1. **Intrinsic Word Vector Analogies:** Evaluate on specific subtasks that a product of human sense, for instance, evaluate intuitive semantic and syntactic relationships using cosine distance after addition/subtraction of vectors
2. **Extrinsic Evaluation** of word vectors: Real-world tasks e.g. named entity recognition (finding organisation, location etc.)
   More in next chapter after word embeddings!
3. **Computation costs**

Further Word2Vec models

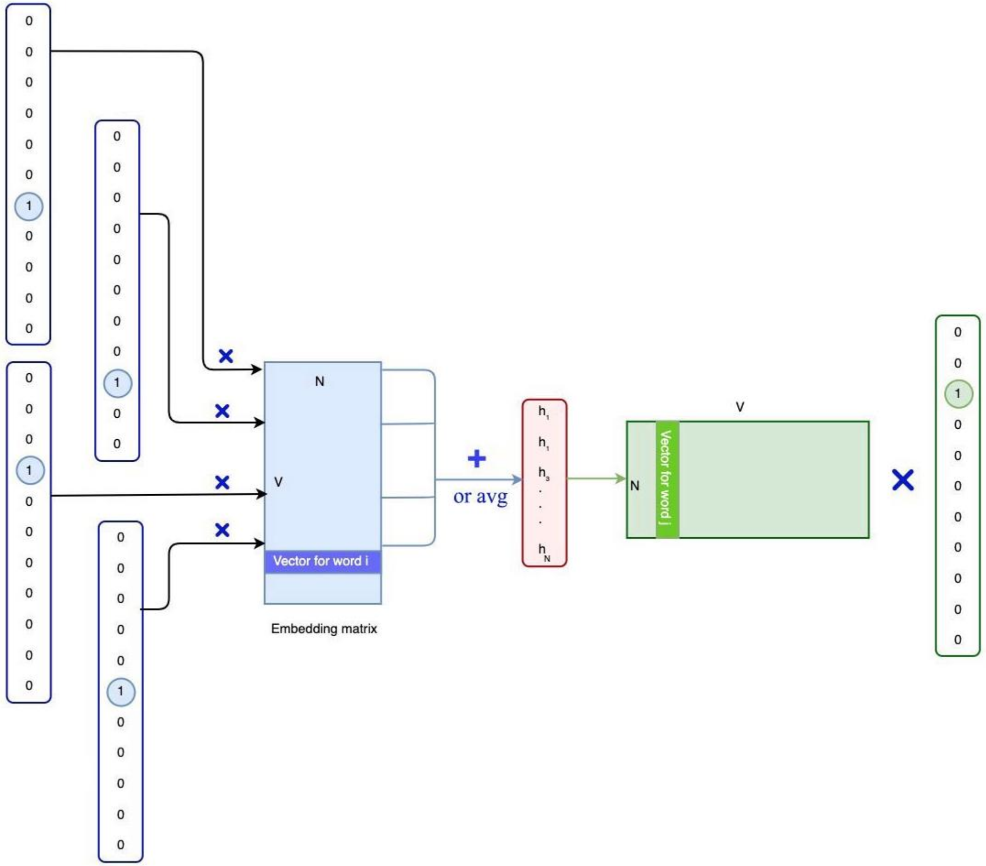
"Mirror" of Skip-gram: **Continuous Bag-of-Words model (CBOW)**

- predicts target words (e.g. "cause") from source context words ("Tobacco smoke … lung cancer")
- statistically CBOW smoothes over a lot of the distributional information



Continuous Bag-of-Words

- Given the context, we want to predict the target word instead
- Advantage: faster training over skip-gram – decrease in computational complexity
- Disadvantage: does not model infrequent words



Further Word Embedding models

Global Vectors for Word Representation (GloVe)

- Hybrid of count-based and predictive model
- Training is performed on aggregated global word-word co-occurrence statistics
- Fast training iterations because the number of non-zero matrix entries is small due to the precompute the global statistics of the entire corpus by the first network pass

Nearest word to frog

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

Word Embedding: Practical advice

How can we use word embedding to tackle a NLP problem?

1. **Leveraging off-the-shelf models**
   a. Three SOTA word embeddings: Word2Vec (by Google), GloVe (by Stanford), fastText (by Facebook) based on different corpuses e.g. Google News, Wikipedia, Mixed…
   b. Freeze weights of embedding layer and add new, trainable layer on top for specific problem

- **Note: retraining word vectors for a specific problem is hard:** Only words in training set are changing, similar words don't change → destroys syntactic & semantic relations

Word Embedding: Practical advice

How can we use word embedding to tackle a NLP problem?

3. **Building a domain specific word embeddings**
   (If many domain words not included in 1.)
   a. Use a spelling checker in advance
   b. Harverest big domain datasets, if possible, enrich your training data by using free available data e.g. Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab), Common Crawl (840B tokens, 2.2M vocab), Twitter (2B tweets, 27B tokens, 1.2M vocab)...

Keep in mind: Underlying word embeddings are often more important than tuning the actual model to achieve (very) good performance!

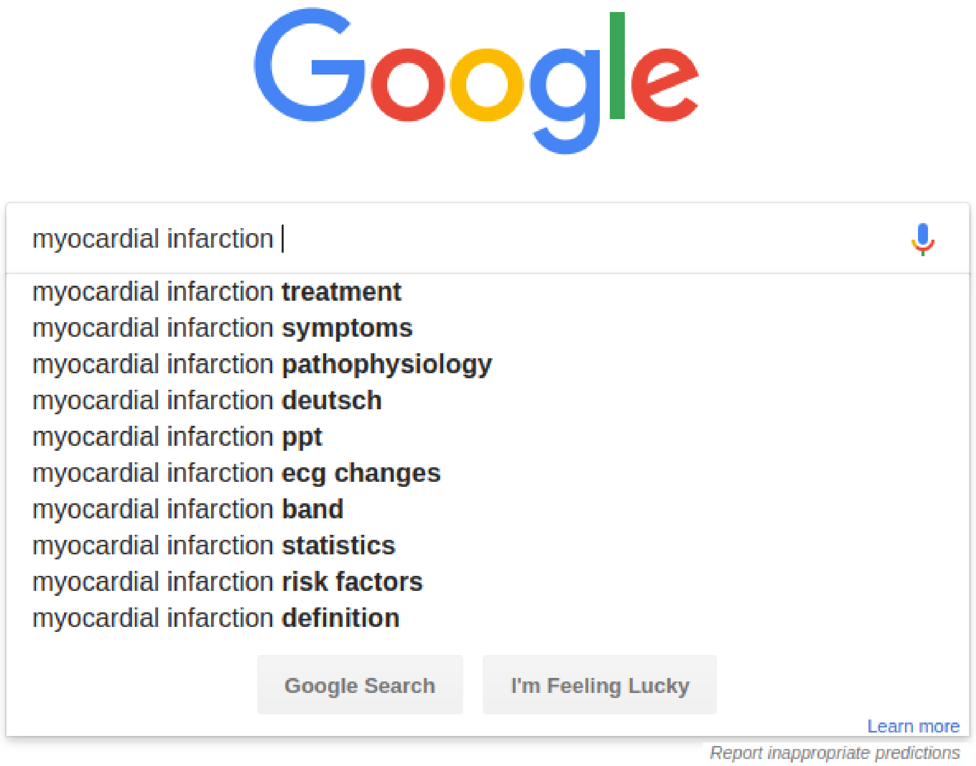
Summary of Word Embeddings

- Continuous Language models are superior over Discrete Language Models by treating a word as a **continuous, dense vector of word meaning** and not as a atomic symbol.
- The word **meaning** is derived by **surrounding context words of the word.**
- Word embedding is state-of-the-art in NLP and the **cornerstone** for almost every NLP task.
- Word embeddings are **evaluated** by **intrinsic** and **extrinsic tasks.**

Still work to do e.g. sarcasm.

Neural Language Models

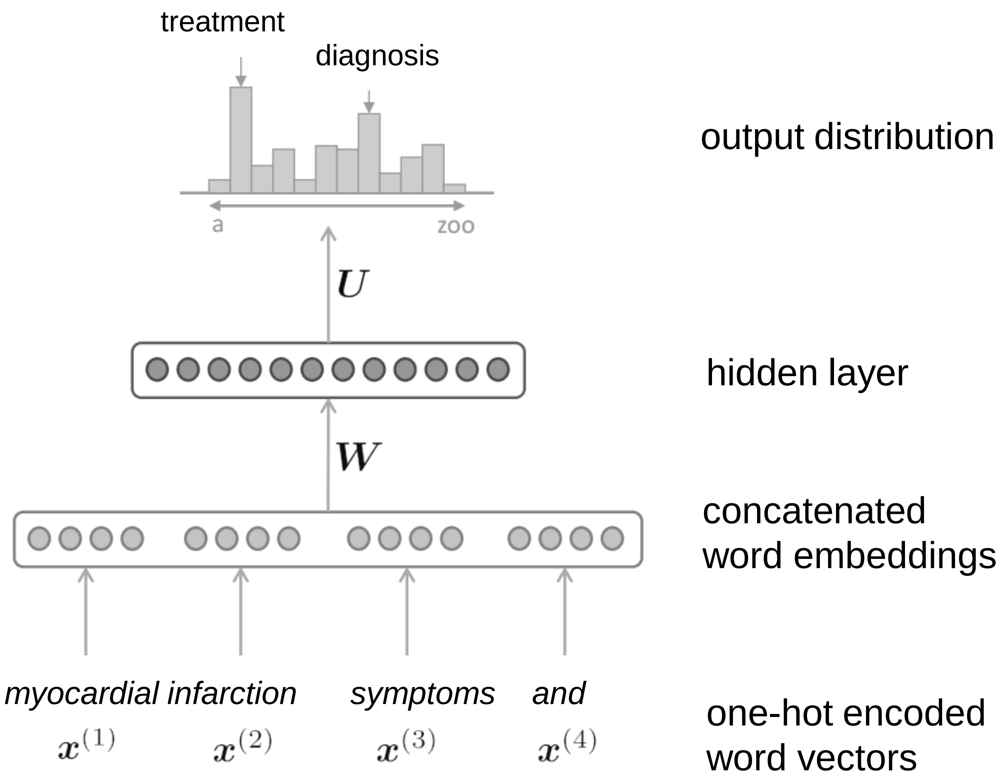
Classification: Predict the next word in a sentence.



daily usage by auto completion systems

Neural Language Models: Feedforward

Feedforward neural network to predict the next word in a sentence.
N-gram is a fixed-size, moving “window” of consecutive words.



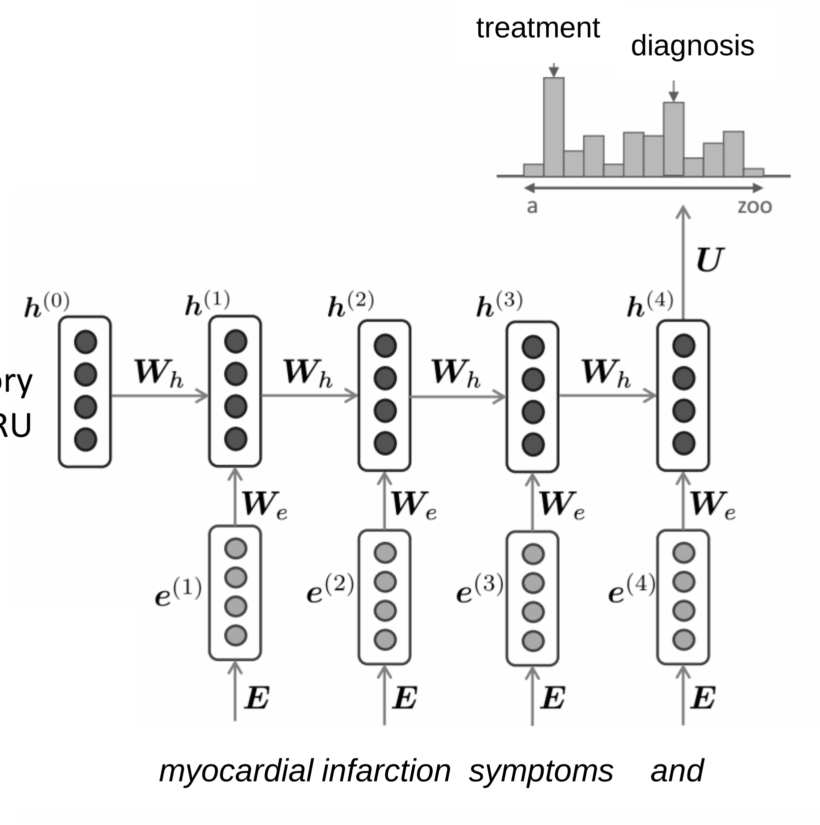
Disadvantages:
- only for small, fixed windows
- no weight sharing across window position

Illustration based on [7]

Neural Language Models: RNN

Recurrent Neural Network, for example, LSTM or GRU to predict the next word in a sentence. Trained on entire corpus optimised by stochastic gradient descent.

Includes memory (gates) for LSTM/GRU



output distribution

hidden layer

word embedding layer

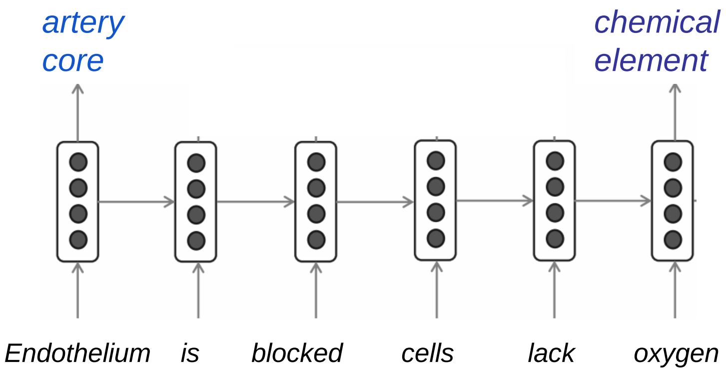
one-hot encoded word vectors

Advantages:
- considers the order of words
- process any length input without increase of model size
- share weights repeatedly across timesteps

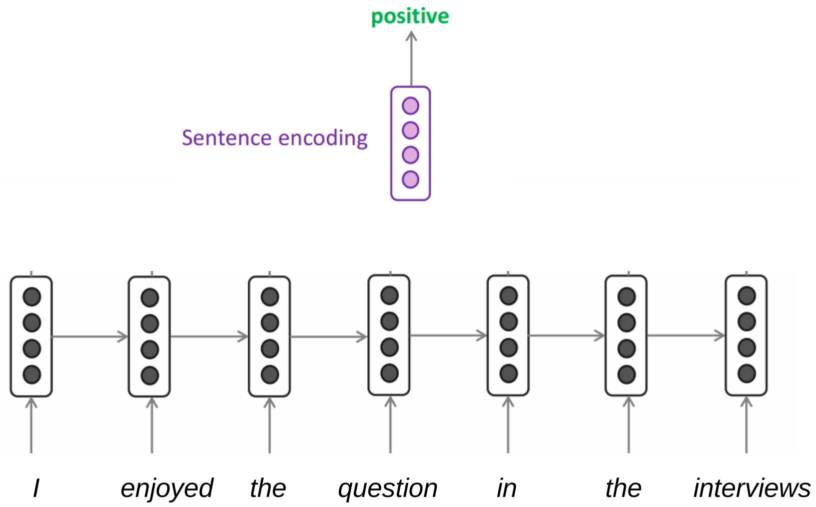
Illustration based on [7]

Recurrent Neural Network text classification examples:

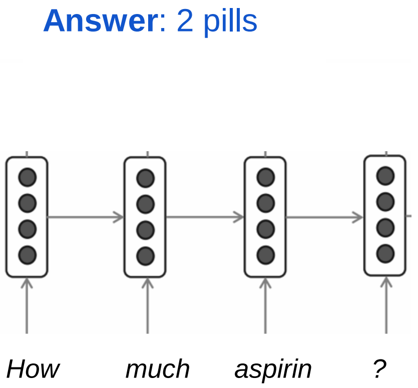
Named Entity Recognition



Sentiment classification



Question answering



Encoder RNN: Encodes sentence or question representation! **How?**

Illustration based on [7]

Convolutional Neural Networks (CNN) for have been used to compute multiple vectors for every possible phrase in parallel. Pure deep learning approach ignoring any linguistic knowledge.

1. Initialize with **pre-trained word vectors** (word2vec, d = 300)
2. **Concatenate** lengthwise the vectors of every **word** to a "sentence vector"
3. **1d-convolution** (windows of length): **Multiple window filter** sizes e.g. 3,4,5 create tri-grams, 4- and 5-grams of sentences
4. Capture only the most important activations by max- or average-**pooling**

Usage of Dropout is necessary, but still very **efficient and simple architecture**!

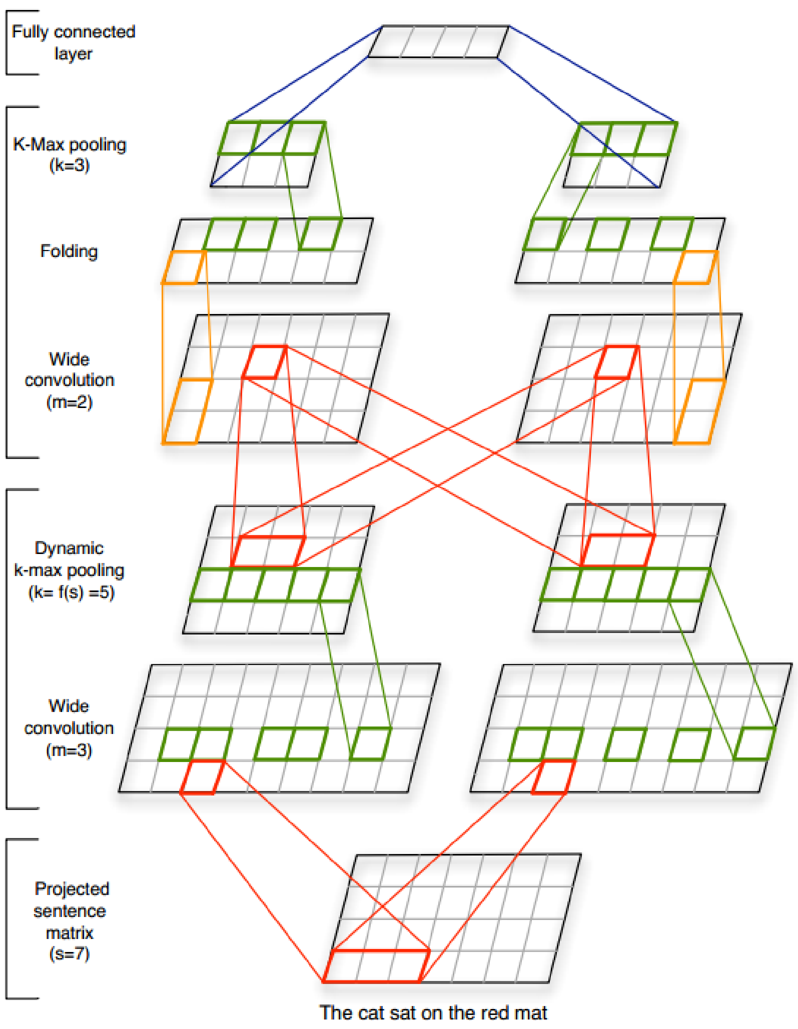


Illustration [16]

Summary Neural Language Models

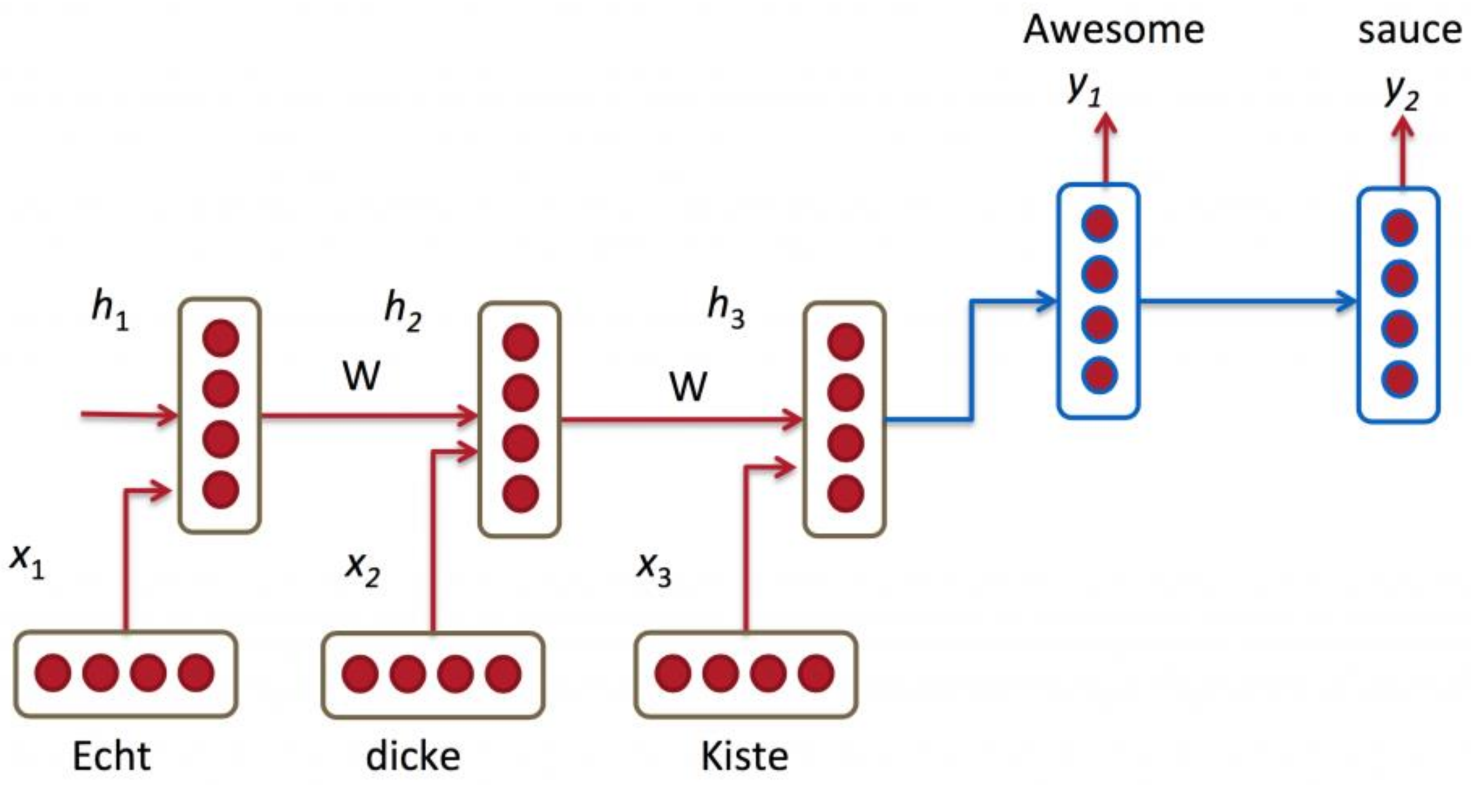
Recurrent Neural Networks:

- Cognitively plausible (reading from left to right, keeping a state)
- Not the best for text classification (n-gram)
- But great for sequence classification
- Very slow computation because not parallelizable

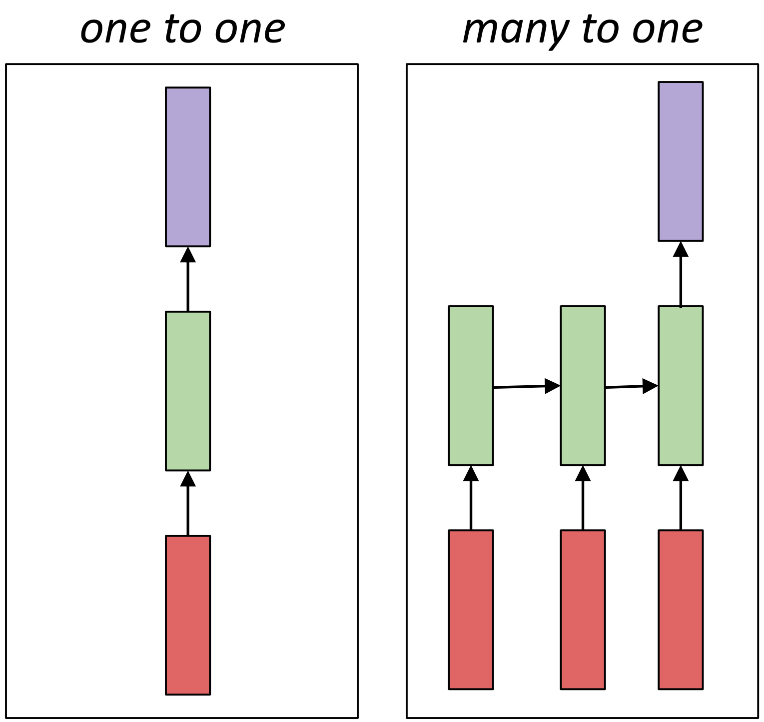
Convolutional Neural Networks:

- Adapted architecture, original for vision purposes
- Very good for text classification
- Need zero padding for shorter phrases
- Hard to interpret on text
- Very efficient and versatile because parallelizable on GPUs

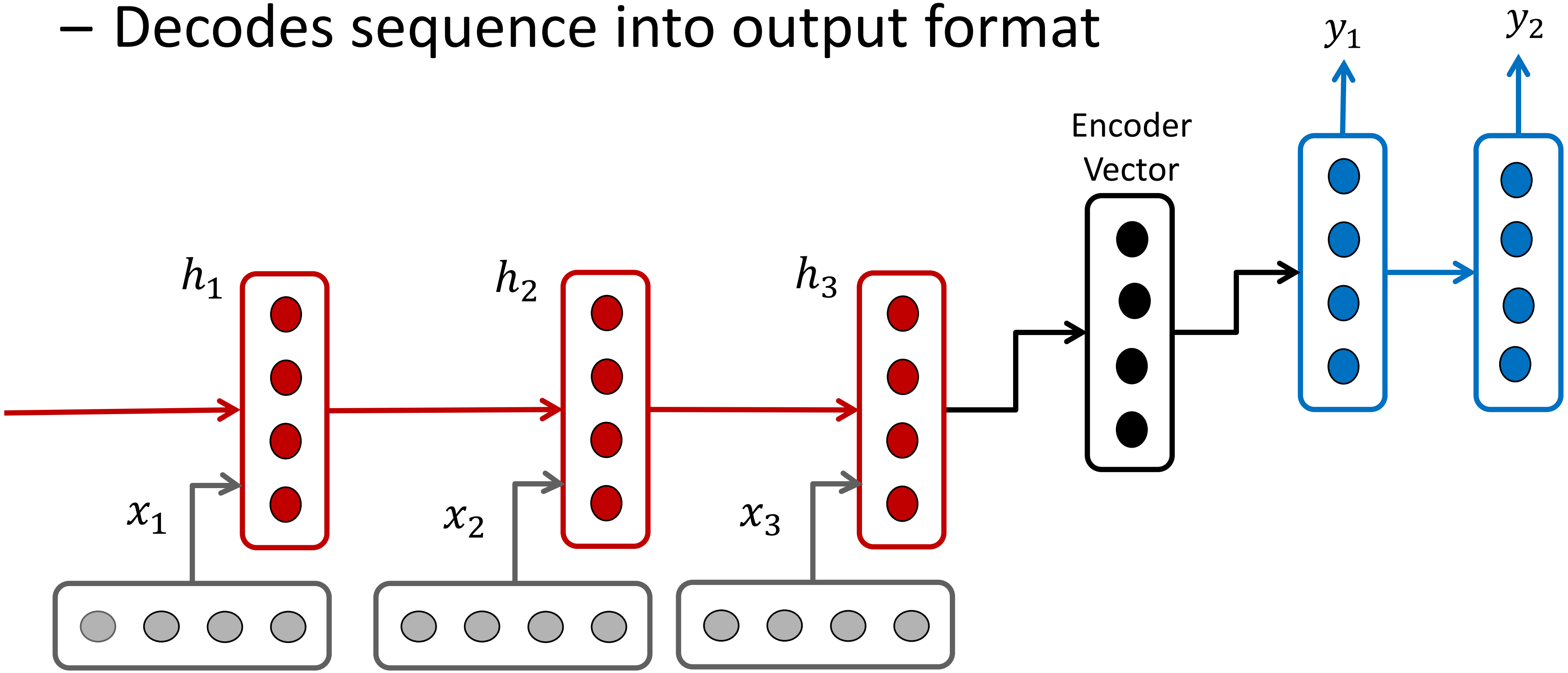
Sequence to Sequence Learning



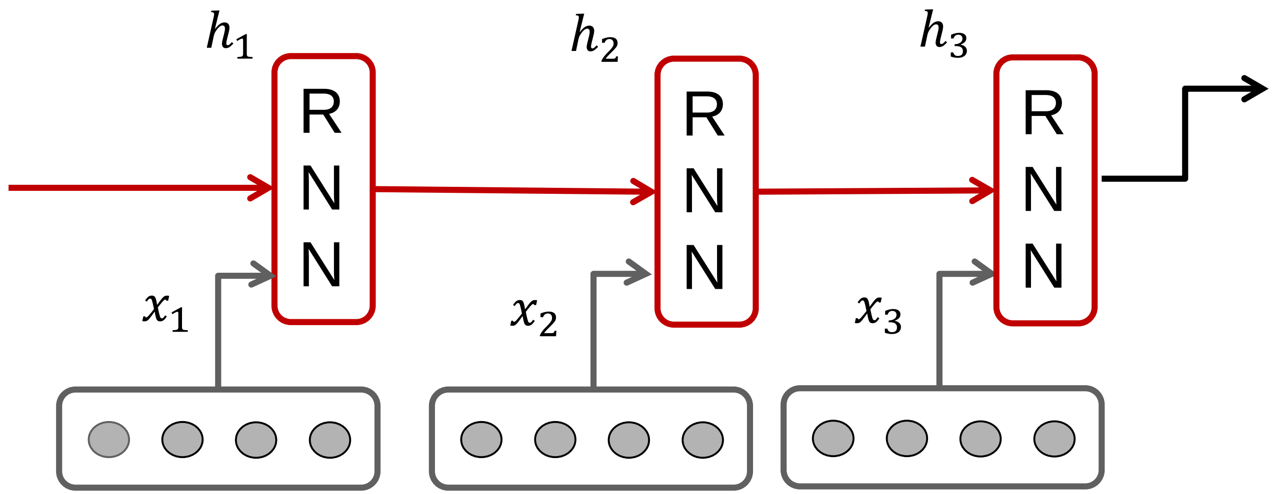
- **Processing sequential inputs/outputs**
  - Requires information/knowledge from **previous inputs**
  - RNN’s we have learnt to date take a current RNN state and input and produces a subsequent RNN state that encodes the sequence so far
  - What happens when the output is also a sequence?



- **Sequence to sequence model**
  - Encoding the source sequence into a vector
  - Decodes sequence into output format



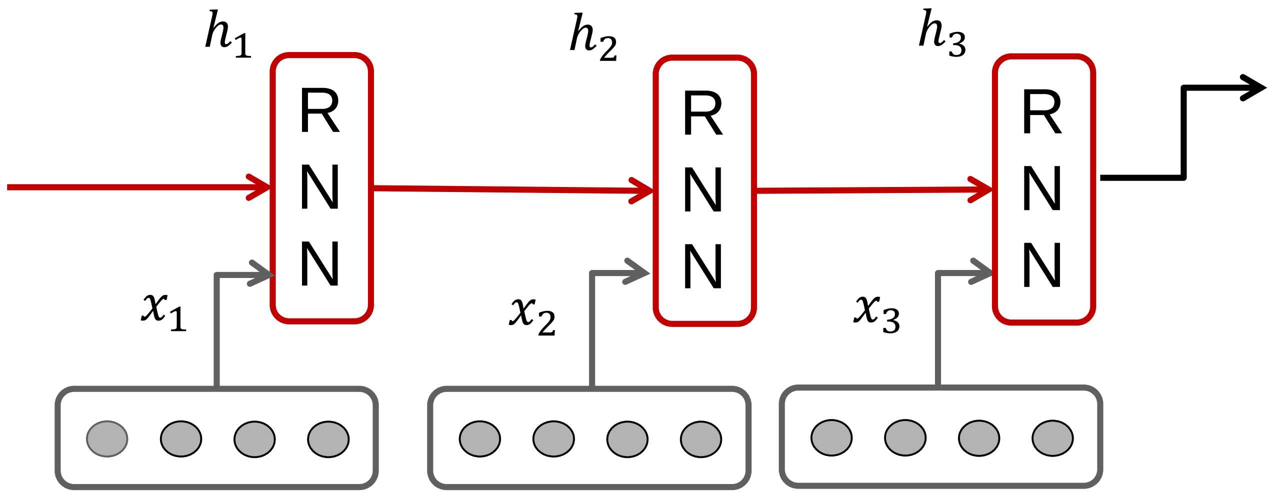
- **Encoder**
  - A stack of several recurrent units (LSTM or GRU)
    - Each accepts a single element of the input sequence
    - Collects information for that element and propagates it forward



- **Encoder**
  - Hidden states, h_t, computed using

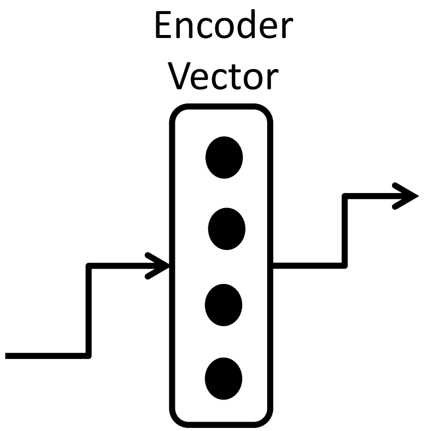
$$h_t = f\left(W^{hh} h_{t-1} + W^{hx} x_t\right)$$

  - Ordinary RNN, in which we apply weights to the previous hidden state and new input vector



Universität Augsburg University

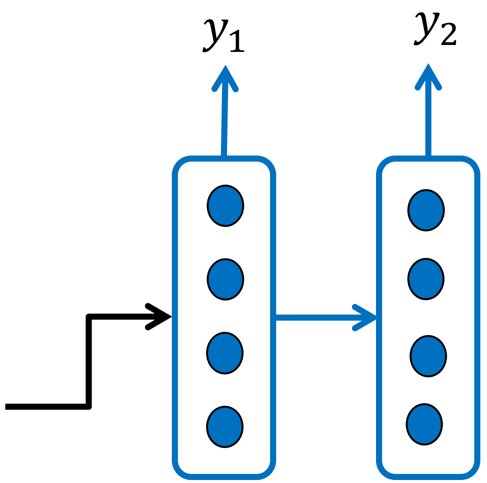
- **Encoder Vector**
  - This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
  - This vector aims to capture the information for all input elements in order to help the decoder make accurate predictions.
  - It acts as the initial hidden state of the decoder part of the model.



- **Decoder**
  - A stack of several recurrent units where each predicts an output y_t at a time step t
  - Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state
  - Hidden states, h_t, computed using:

$$h_t = f\left(W^{hh} h_{t-1}\right)$$

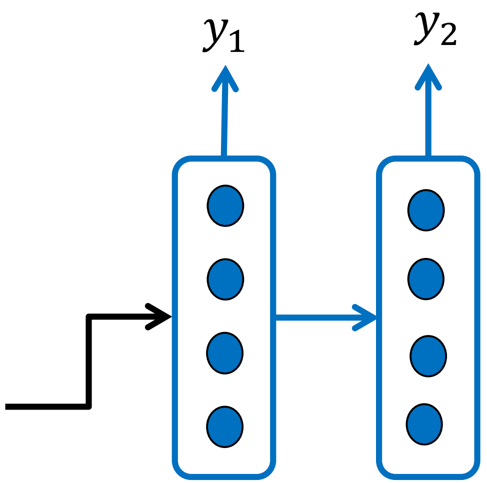
    - Uses previous hidden state to compute the next



- **Decoder**
  - Output, y_t, computed using:

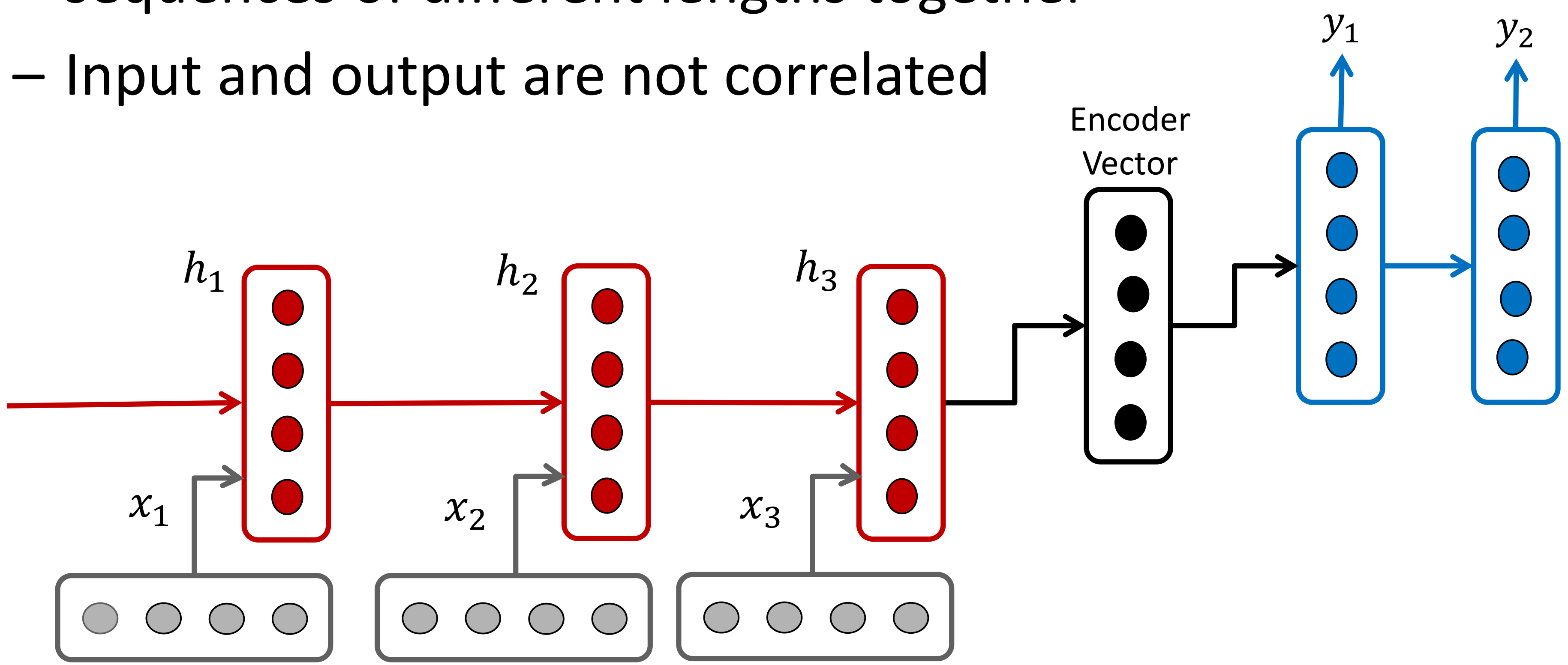
$$y_t = softmax(W^{yh} h_t)$$

  - Softmax function normalises the vector of scores given by the dot product $W^{yh} h_t$ into a probability distribution



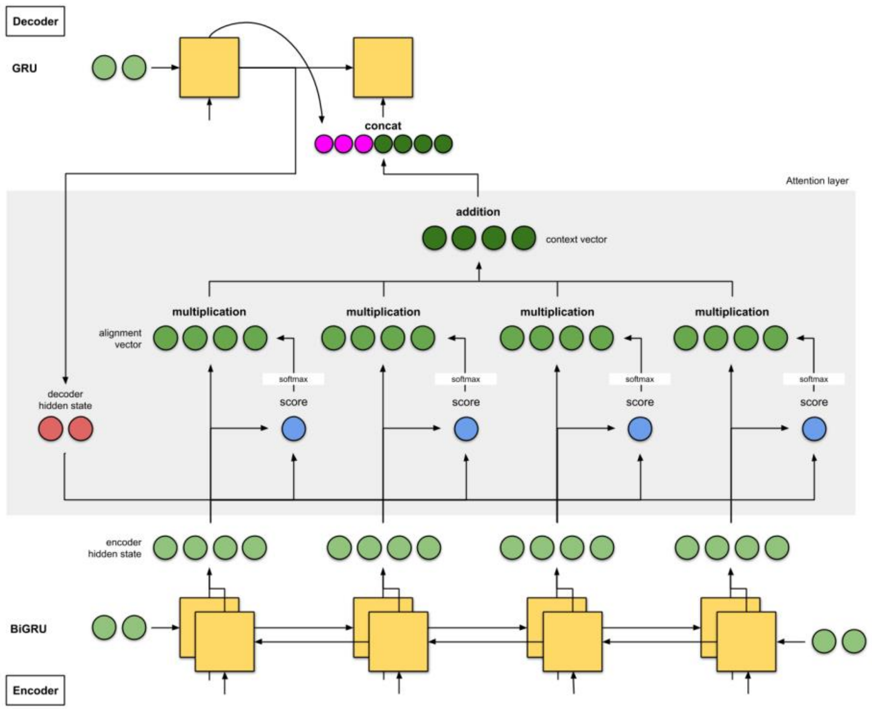
Universität Augsburg University

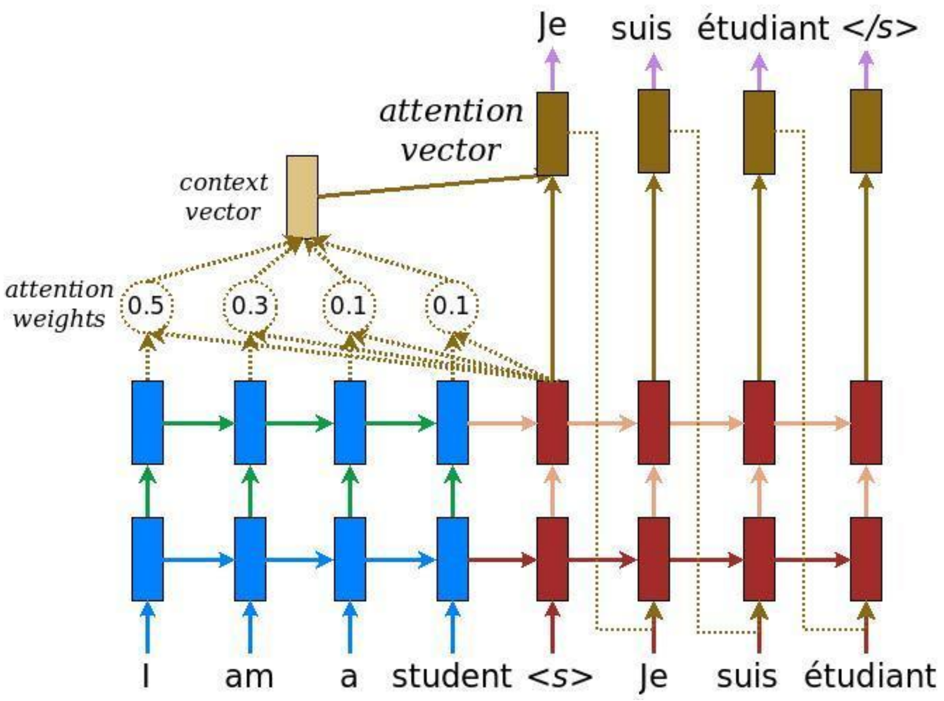
- **Sequence to sequence model**
  - Advantage of this model is that it can map sequences of different lengths together
  - Input and output are not correlated



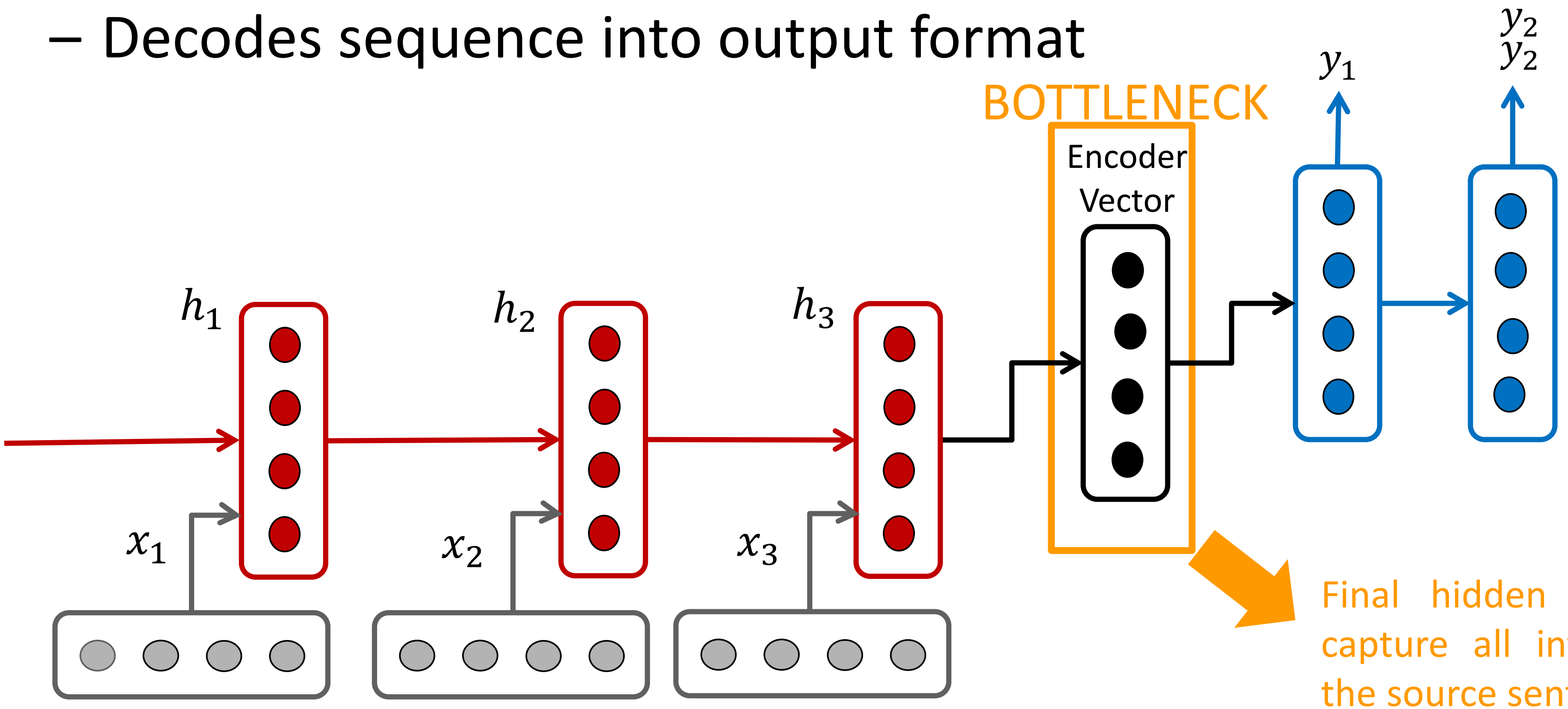
- **How is it trained?**
  - Decoder forced to generate correct sequences
  - It is penalised for assigning a sequence with low probability
  - Losses are calculated for each output token
  - Losses are summed across the output sequence
  - New parameters then calculated using gradient descent
  - Typical loss function is cross-entropy
    - At each output the network produces an probability over all possible outputs
    - Cross-entropy penalises differences between distributions

Attention Mechanisms



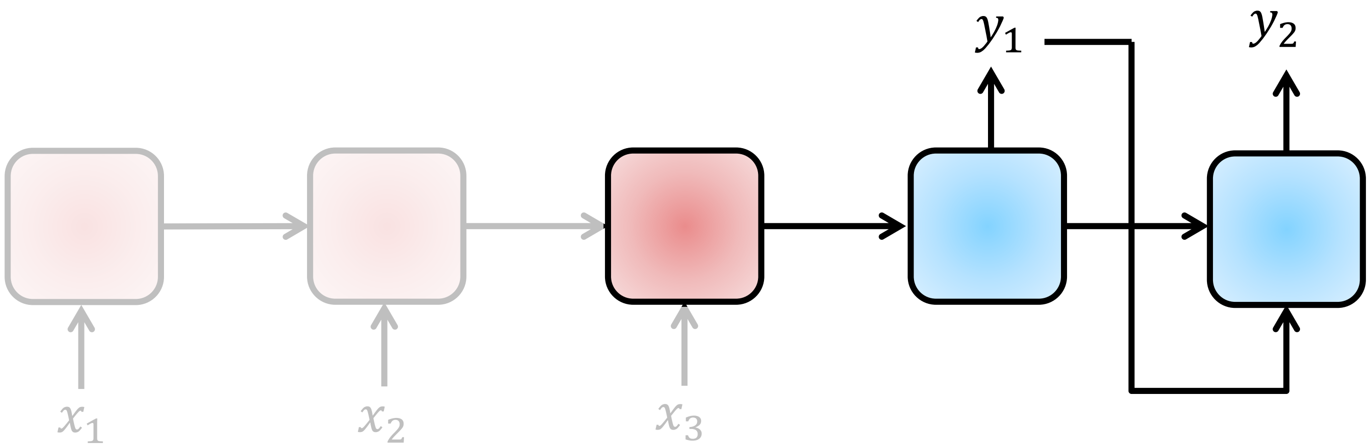


- **Sequence to sequence model**
  - Encoding the source sequence into a vector
  - Decodes sequence into output format



Final hidden state needs to capture all information about the source sentence

- **Sequence modelling bottleneck**
  - Conventional techniques discard all the intermediate encoder states and use only its final state to initialise the decoder
    - This technique works good for smaller sequences
  - This a vector becomes a bottleneck as sequence length increases
    - Difficult to summarise long sequences into a single vector

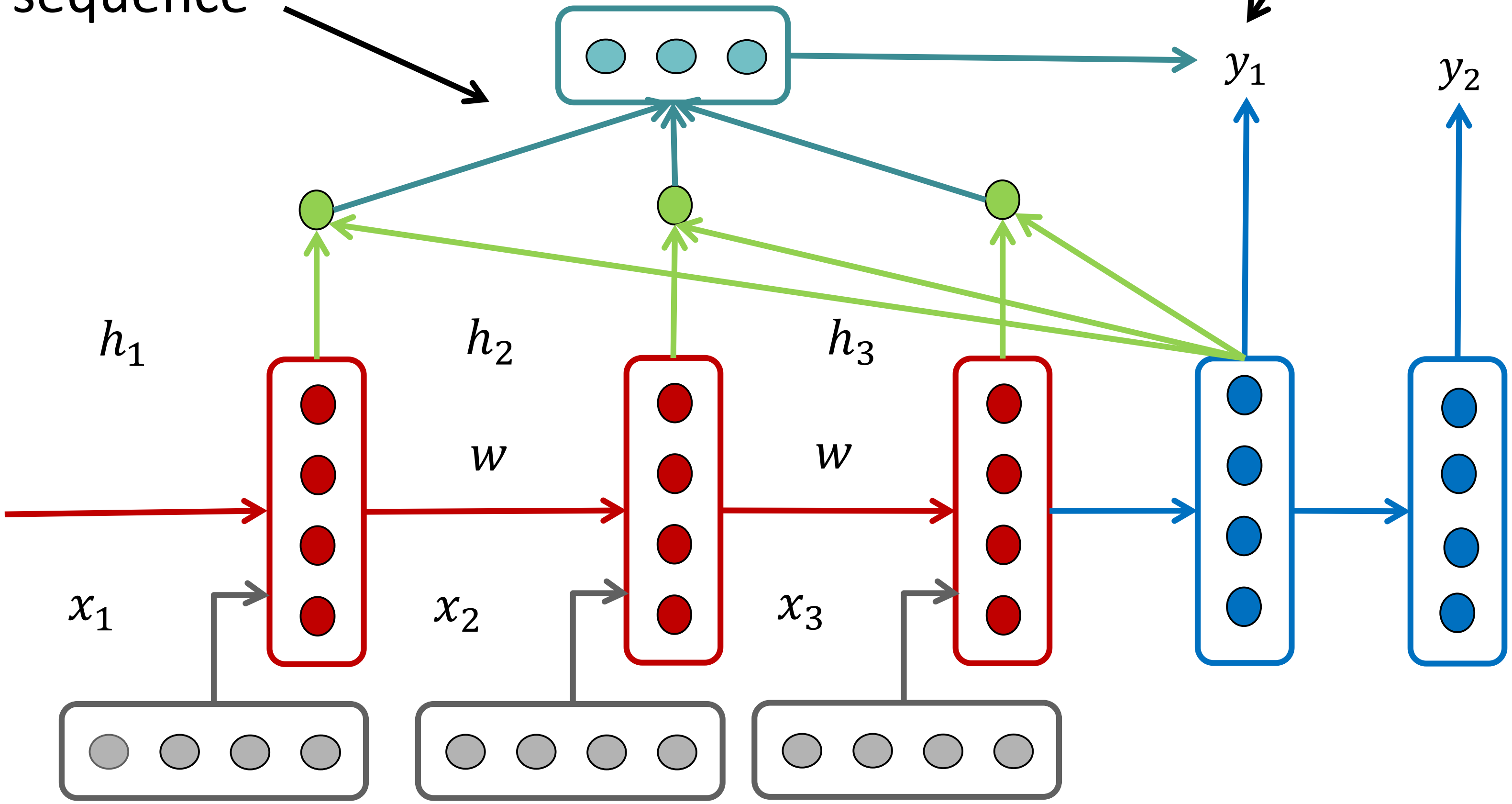


- **Attention Mechanisms: Core Idea**
  - **Do not discard intermediate encoder states**, instead utilise all states in order to construct an new context vector
    - Probability distribution mapping each input to the output state that the decoder wants to generate
  - Use this context vector when decoding the output sequence
  - This means the decoder captures global information rather than solely making inferences based on a single hidden state
  - During training the new network learns which inputs are important for the task, hence the name attention

Attention Mechanisms

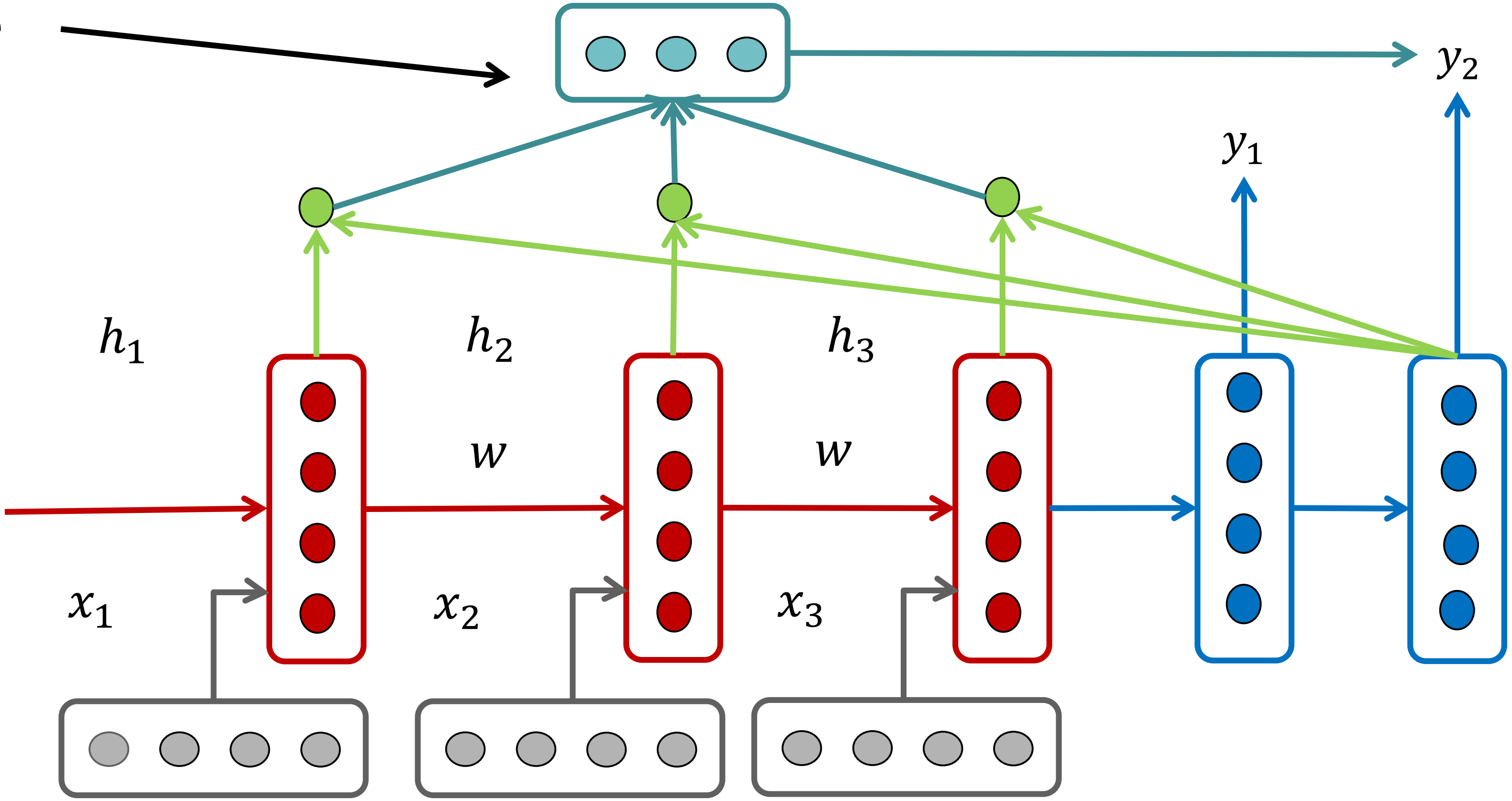
At each step of the decoder, focus on a particular part of the source sequence

Concatenate context vector with decoder hidden state before producing output



Universität Augsburg University

This mapping is repeated at each output sequence

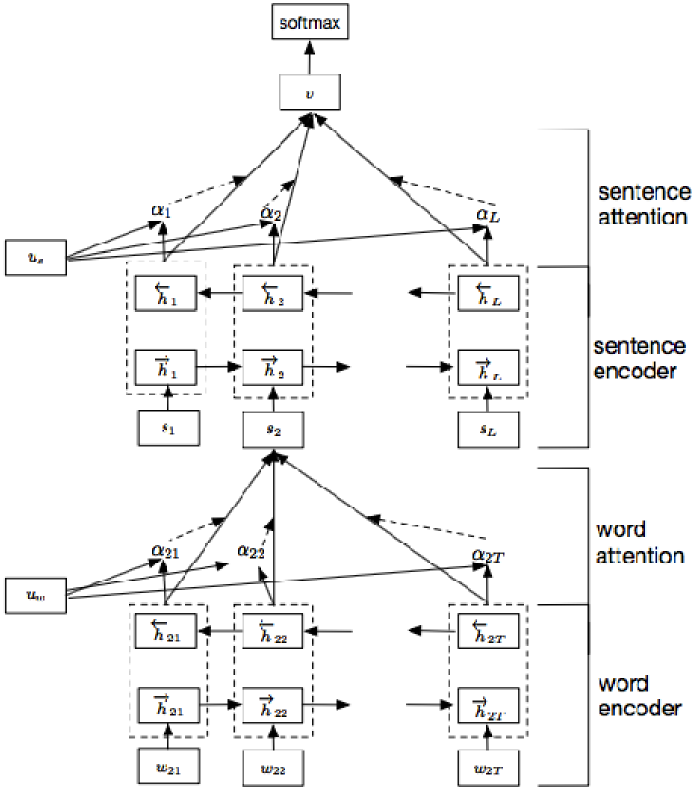


Model learns mappings between different parts of the input sequence and corresponding parts of the output sequence

- **Core Steps of basic Attention Modules**
  - We have encoder hidden states $h_1, \dots, h_n \in \mathbb{R}^h$
  - At timestep , we have decoder hidden state $s_t \in \mathbb{R}^h$
  - We get the attention scores for this step: $e^t = [s_t^T h_1, \dots, s_t^T h_n] \in \mathbb{R}^h$
  - We use a softmax activation to get the attention distribution for this step $\alpha^t = \mathrm{softmax}(e^t) \in \mathbb{R}^h$
  - We use α^t to take a weighted sum of the encoder hidden states to get the attention output $c_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$
  - Finally we concatenate the attention output with the decoder hidden state: $[c_t; s_t] \in \mathbb{R}^{2h}$

Hierarchal Attention

- **Classify a document made up of many sentences**
  1. Apply RNN on word level
  2. Use attention models to extract words that are important to the meaning of each sentence
  3. Aggregate theses representation to form a sentence vector
  4. Apply the same procedure on derived sentence vectors to generate a vector capturing the meaning of the document
  5. Use this vector for final classification

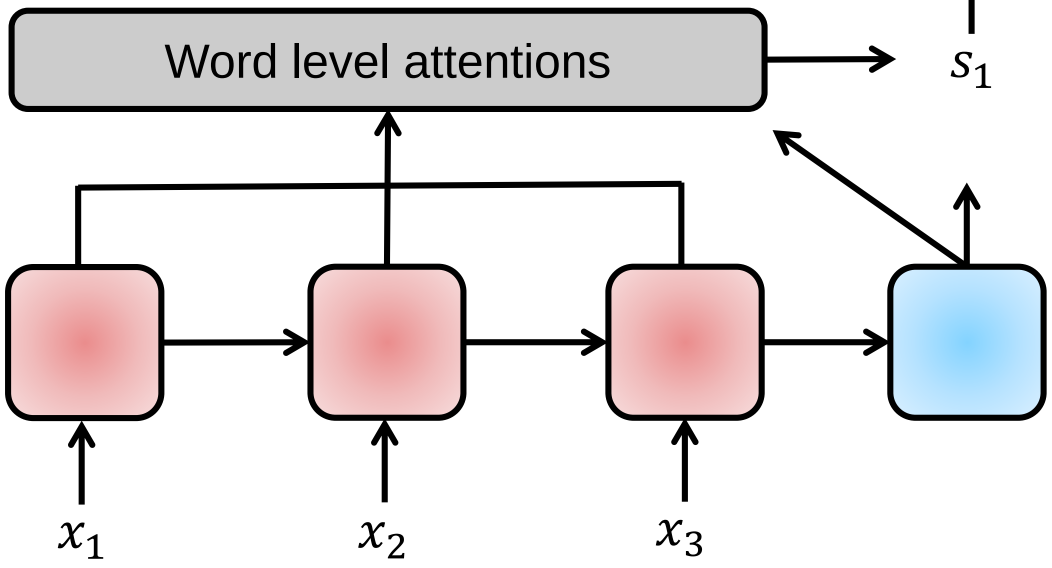


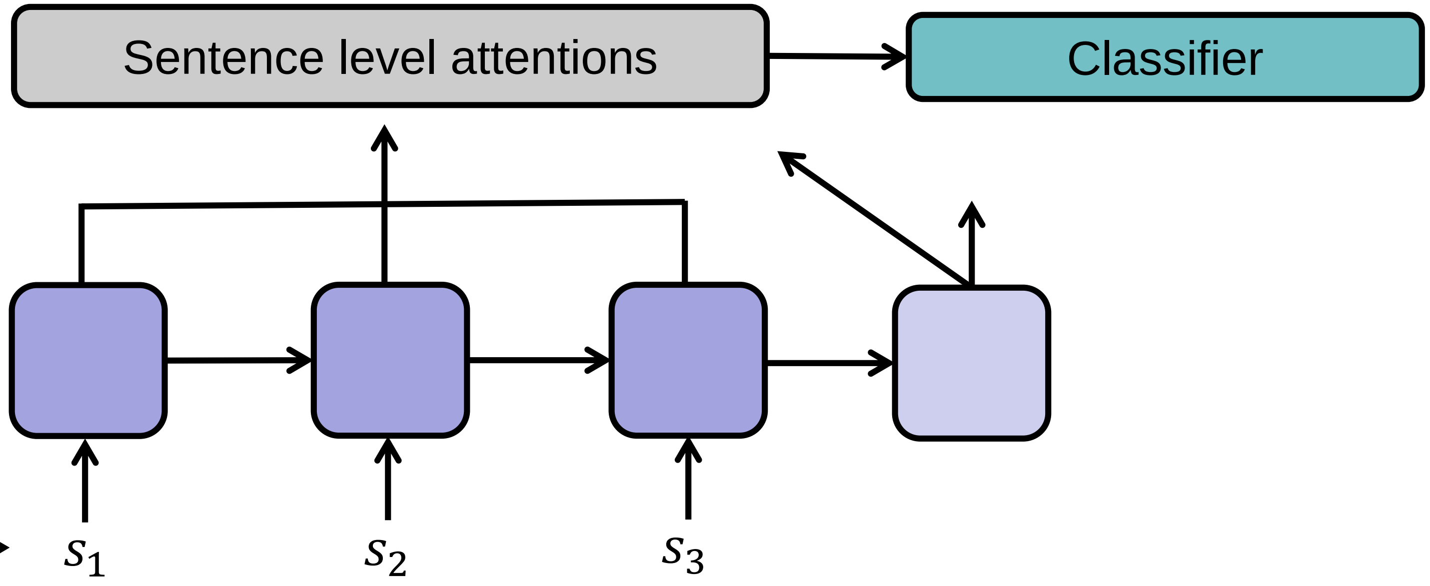
Yang et al., "Hierarchical attention networks for document classification," in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016, pp. 1480–1489, ACL.

Hierarchal Attention

Word encoder + context attention

- The input layer reads in one full sentence a time, in the form of word representations
- The attention mechanism selects the most important words

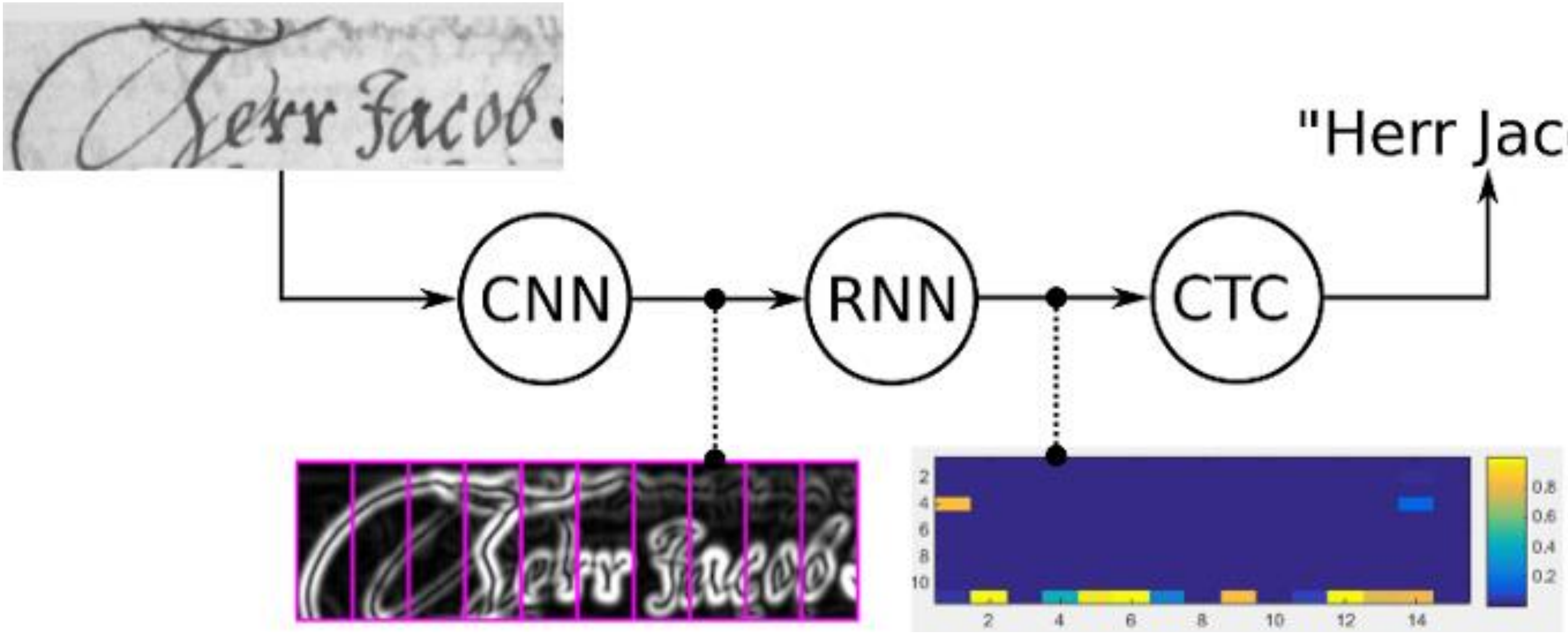




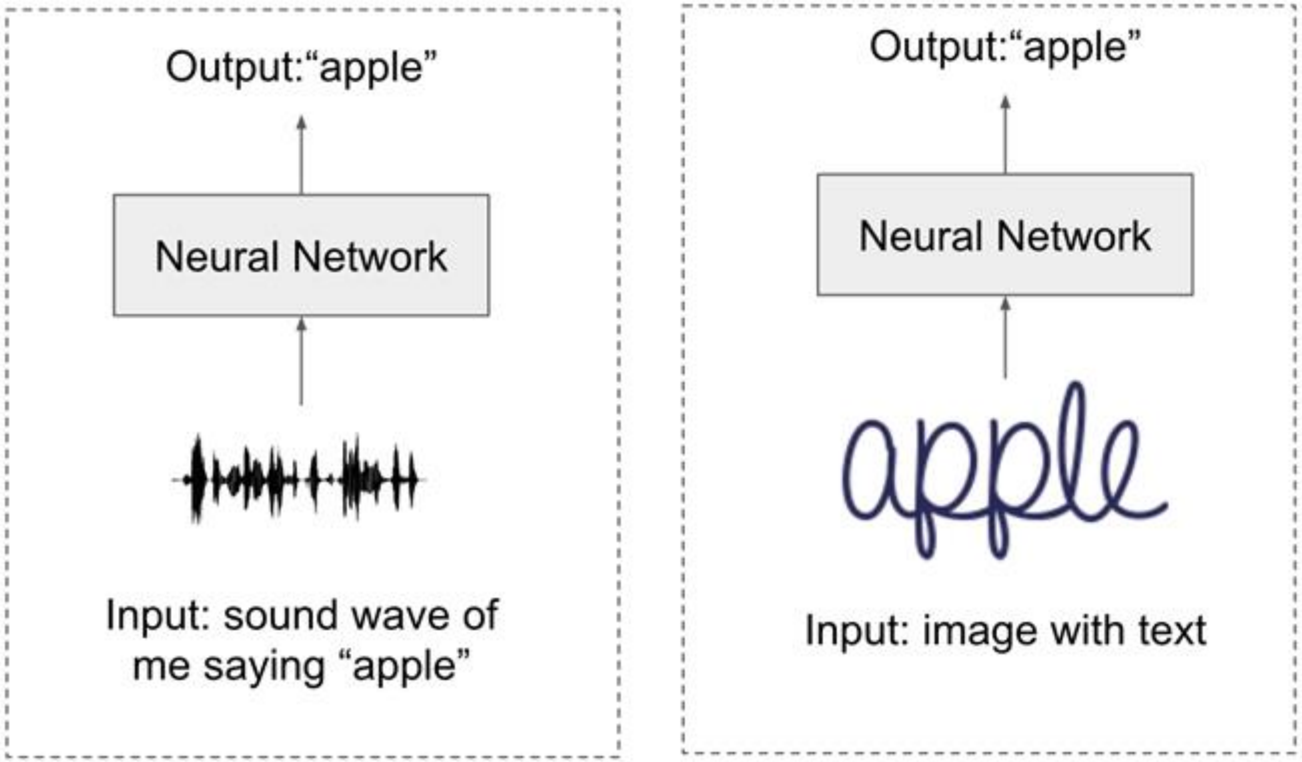
Sentence Encoder + context attention

- Reads in the sentence representation output
- The attention mechanism selects the most important sentences
- The output is a document representation which should capture the meaning of the entire document

Connectionist Temporal Classification

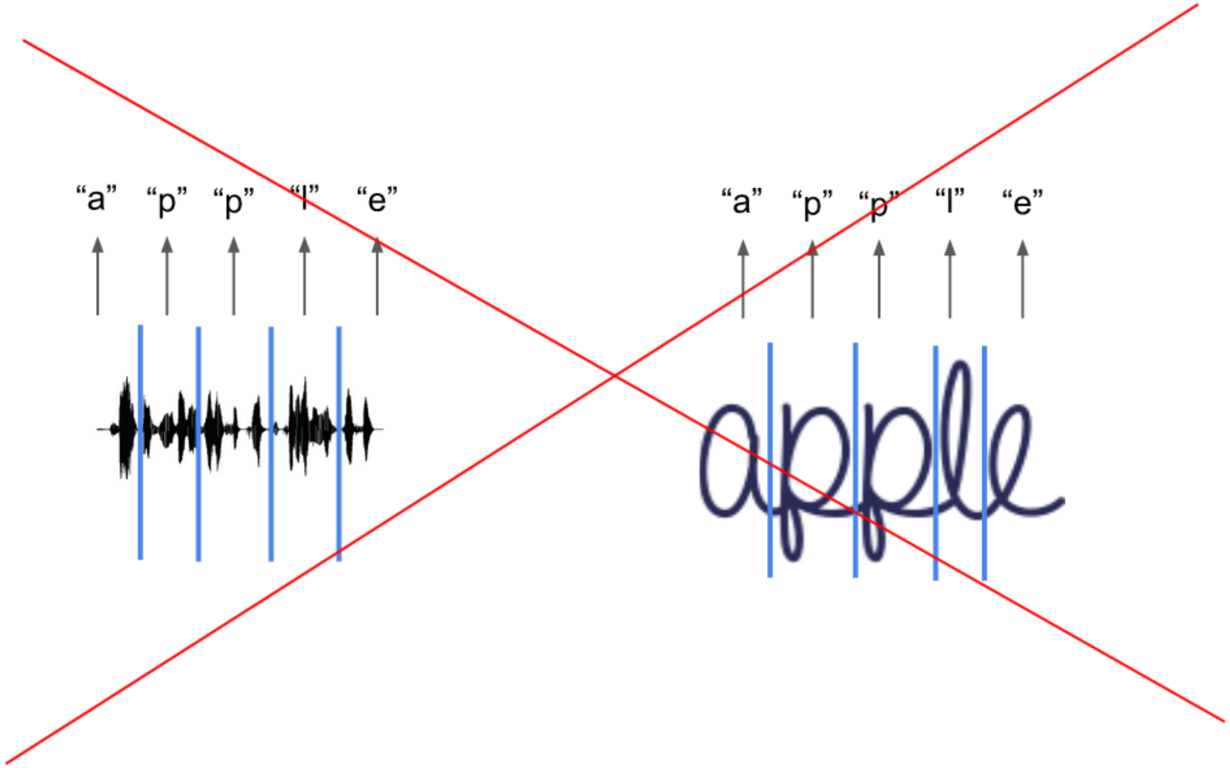


- **Performing sequence learning on unsegmented data**



Motivation of CTC

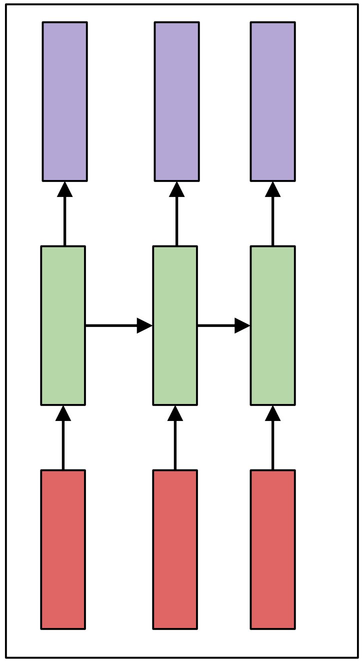
Labelling unsegmented sequence data. i.e. training data is not pre-segmented.



We can not pre-segment input data because:

- It is too time consuming
- It is too expensive
- It is impossible in most cases

Output: “deep learning”

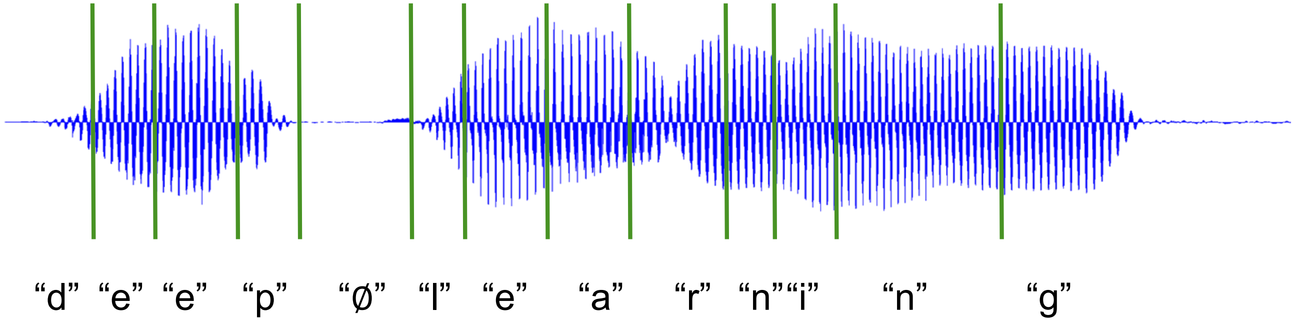


Input: audio waveform of “deep learning”

Example: Automatic speech recognition

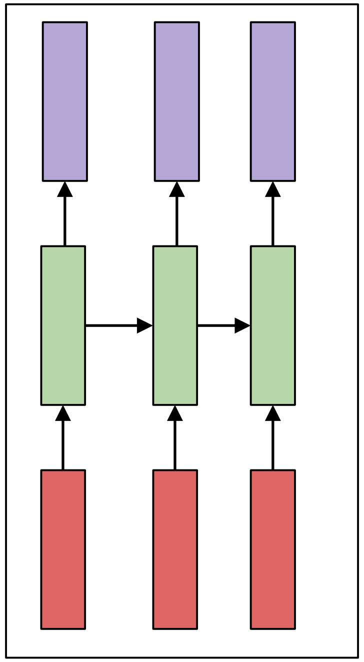
➔ label of each frame is required

demand alignment in preprocessing



Motivation of Connectionist Temporal Classification

Output: "deep learning"



Input: audio waveform of "deep learning"

CTC is applied to address this **temporal classification problem** without the need for **frame-level alignments**, and normally output sequence is much shorter than input sequence.

The CTC Model (Graves et. al., 2006)

- **At each step, the network can output a "blank" label or any character in the vocabulary L**
- Transform the network outputs into a conditional probability distribution **over label sequences**, which enables to compute the probability for each path:

$$Pr(p|X) = \prod_{t=1}^{T} y_t$$

- The total probability of any label sequence can be found by summing the probabilities of the different paths leading to it:

$$Pr(Y \mid X) = \sum_{p \in \phi(Y)} Pr(p \mid X)$$

Graves, Alex, et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." Proceedings of the 23rd International Conference on Machine learning, 2006.

Overview of CTC

CTC is a sequence-to-sequence learning technique

E.g., *input X* =(x_1, …, x_T) - a sequence of frame-level observations

↓ *of the length T*

CTC path P = (y_1, …, y_T) -a sequence of frame-level predictions

↓ *mapping : generating label sequence from the CTC path*

output Y =("d", "e", "e", "p") - a much shorter label sequence

$$L_{CTC} = -lnPr(Y \mid X)$$

- **A CTC path** **P** is a sequence of labels on frame-level
- Its likelihood can be decomposed into independent frames:

$$Pr(p|X) = \prod_{t=1}^{T} y_t$$

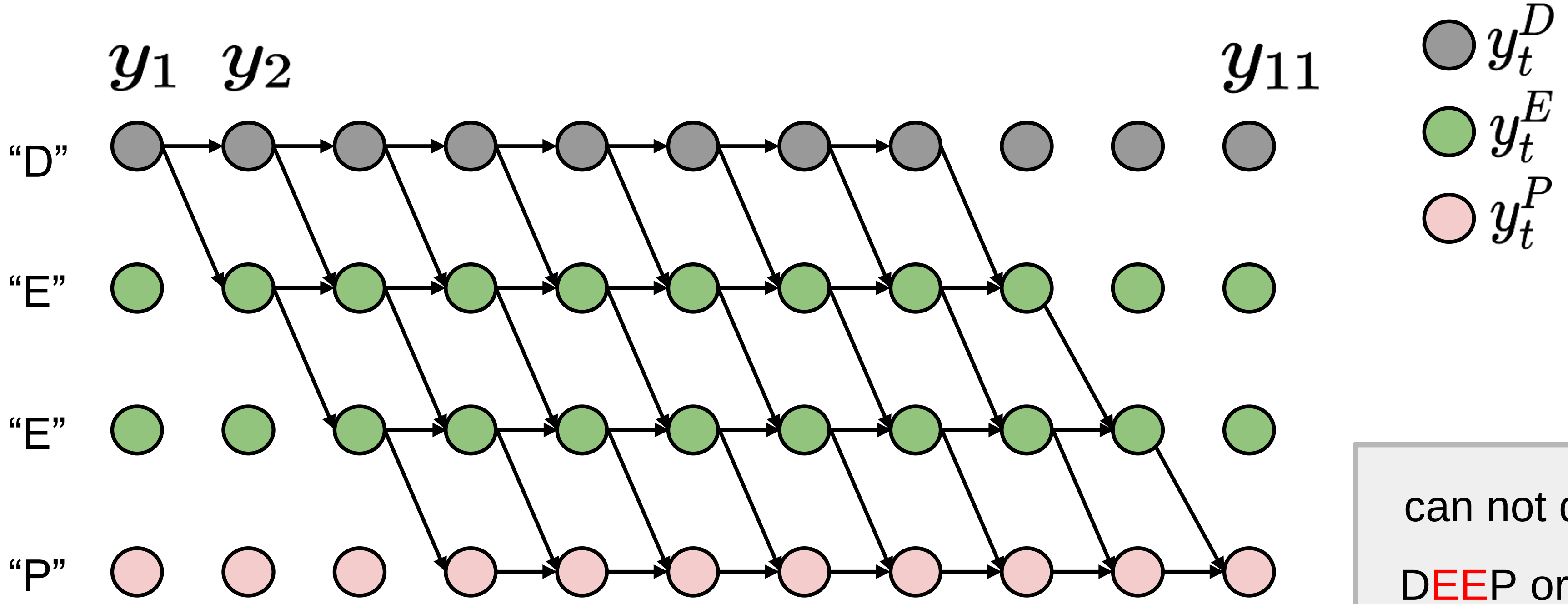
- It differs from labels as it
  - Introduces a special symbol - blank "Ø" - as an additional label, meaning no (actual labels) are assigned to the frame
  - Allows repetitions of non-blank labels

e.g.

D Ø Ø E Ø E ØØ PØ	→	DEEP
D D Ø E Ø E E Ø P P	→	DEEP
Ø D E E Ø E E E P P	→	DEEP
Ø D Ø E E E EØ PØ	→	DEP

a lot of CTC paths mapping to a *final* label "DEEP"

Example of possible paths for output "DEEP"

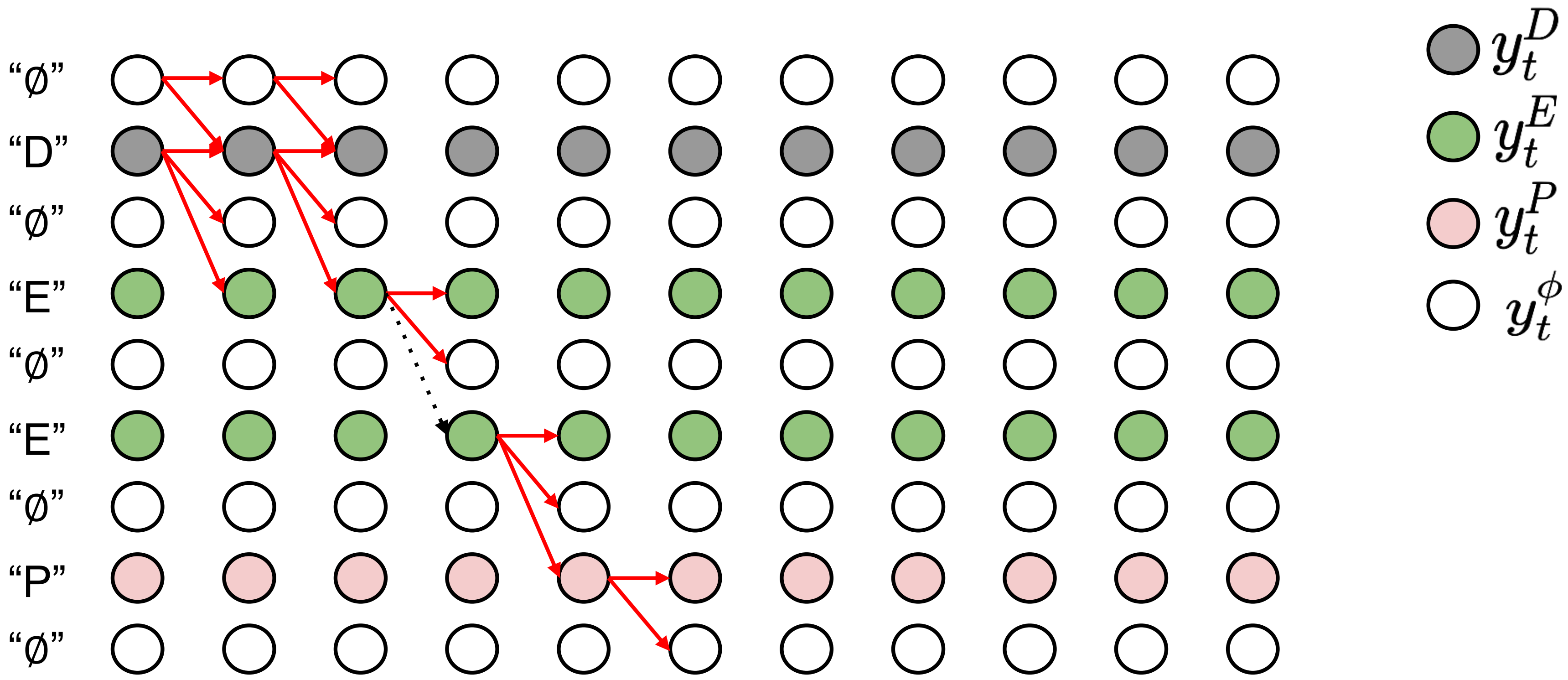


can not distinguish if it is

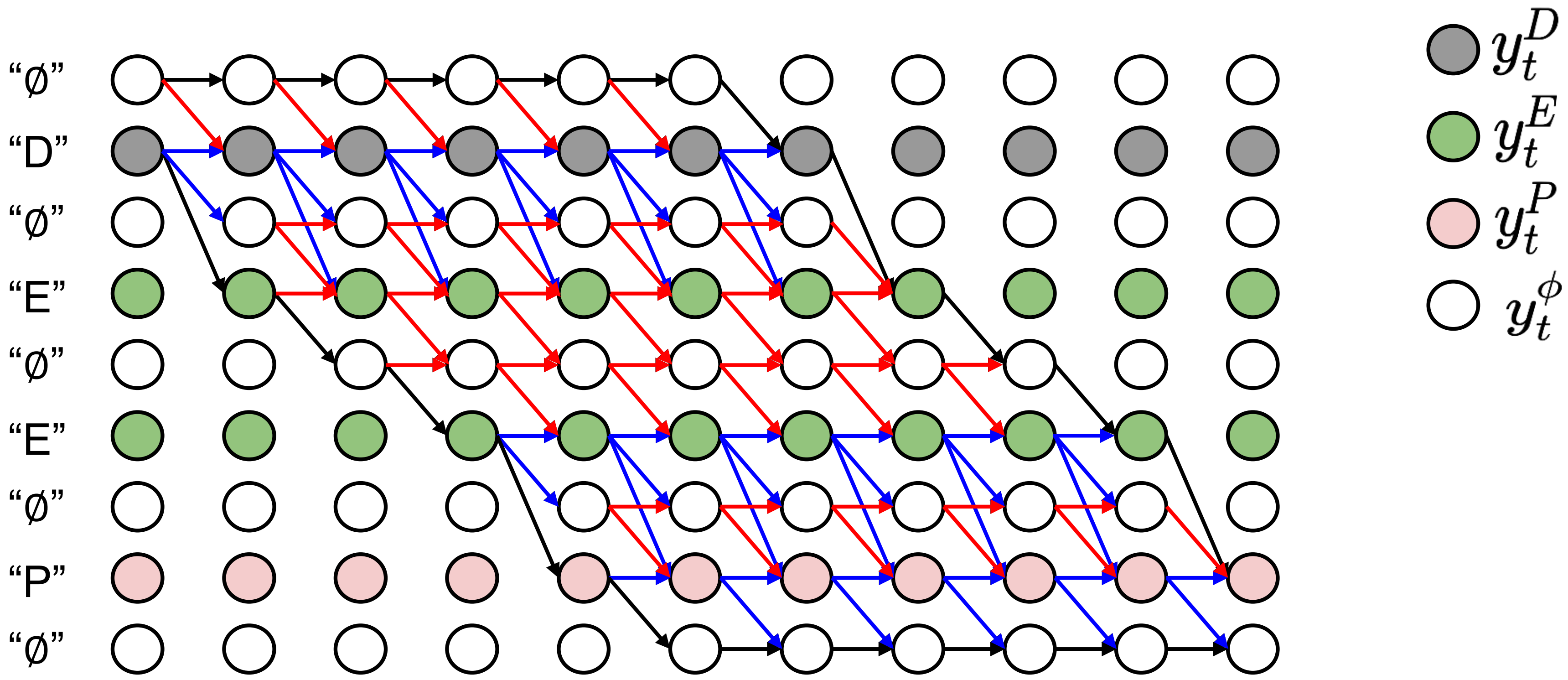
DEEP or DEP or DEEEP

Universität Augsburg University

Example of possible path for output “DEEP” with “∅”

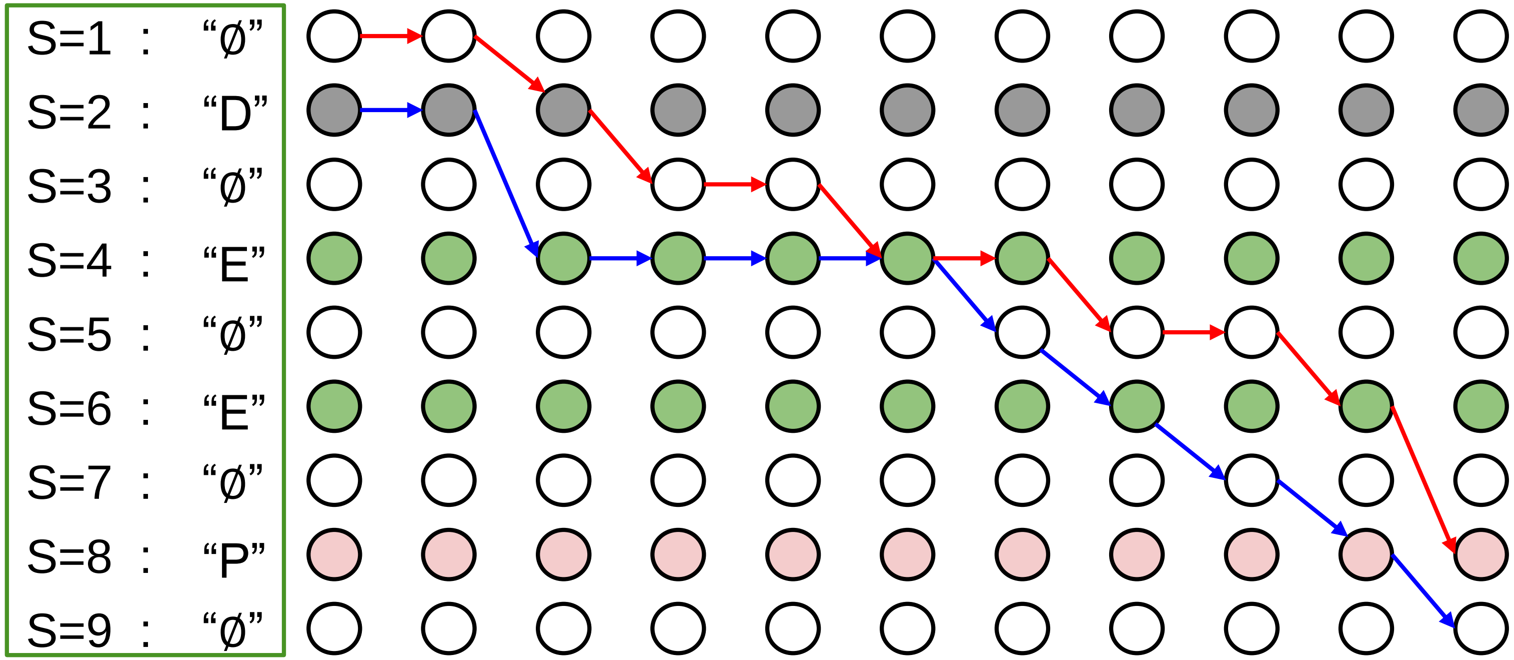


Example of possible paths for output "DEEP" with "∅"



Issue: Massive number of potential paths

Forward-Backward Algorithm



summed probability of **all** the CTC paths **ending** at time ***t*** with state ***s***

$$\boxed{\alpha_t(s)} \cdot y_t^s \cdot \boxed{\beta_t(s)}$$

summed probability of **all** the CTC paths **starting** at time ***t*** with state ***s***

$P_1 = y_1^{\emptyset} \cdot y_2^{\emptyset} \cdot y_3^{D} \cdot y_4^{\emptyset} \cdot y_5^{\emptyset} \cdot \boldsymbol{y_6^{E}} \cdot y_7^{E} \cdot y_8^{\emptyset} \cdot y_9^{\emptyset} \cdot y_{10}^{E} \cdot y_{11}^{P}$

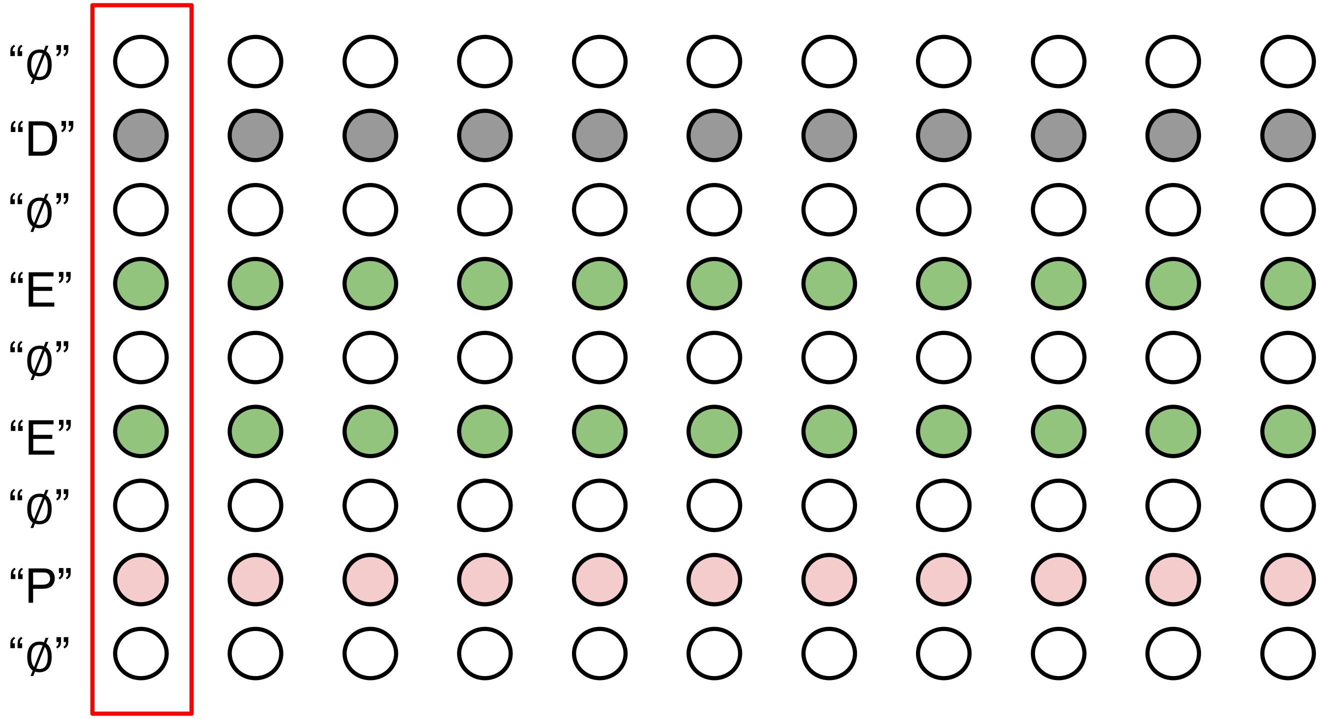
$P_2 = y_1^{D} \cdot y_2^{D} \cdot y_3^{E} \cdot y_4^{E} \cdot y_5^{E} \cdot \boldsymbol{y_6^{E}} \cdot y_7^{\emptyset} \cdot y_8^{E} \cdot y_9^{\emptyset} \cdot y_{10}^{P} \cdot y_{11}^{\emptyset}$

$P_3 = y_1^{\emptyset} \cdot y_2^{\emptyset} \cdot y_3^{D} \cdot y_4^{\emptyset} \cdot y_5^{\emptyset} \cdot \boldsymbol{y_6^{E}} \cdot y_7^{\emptyset} \cdot y_8^{E} \cdot y_9^{\emptyset} \cdot y_{10}^{P} \cdot y_{11}^{\emptyset}$

$P_4 = y_1^{D} \cdot y_2^{D} \cdot y_3^{E} \cdot y_4^{E} \cdot y_5^{E} \cdot \boldsymbol{y_6^{E}} \cdot y_7^{E} \cdot y_8^{\emptyset} \cdot y_9^{\emptyset} \cdot y_{10}^{E} \cdot y_{11}^{P}$

more paths than these 4 paths that pass $\boldsymbol{y_6^{E}}$

- **Forward computation $\alpha_t(s)$**



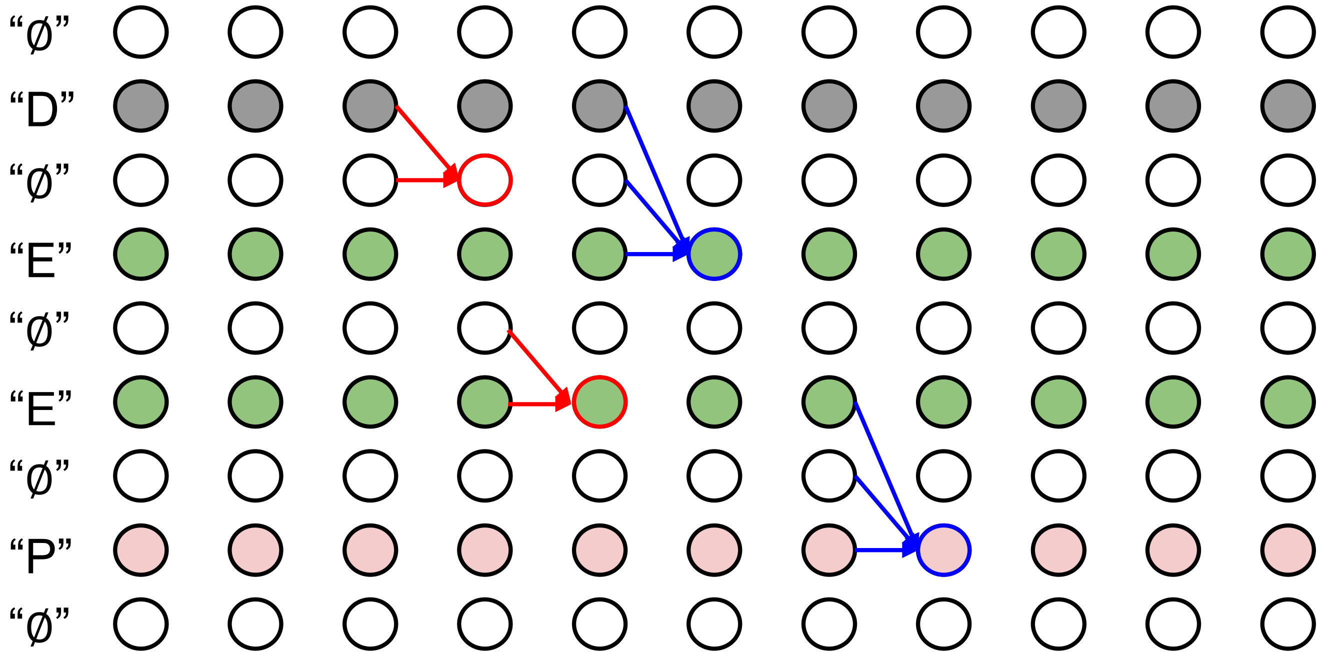
summed probability of **all** the CTC paths **ending** at time ***t*** with state ***s***

$$\boxed{\alpha_t(s)} \cdot y_t^s \cdot \beta_t(s)$$

$$\alpha_1(\phi) = y_1^\phi \qquad \alpha_1(D) = y_1^D \qquad a_1(l_s) = 0, \text{if } l_s \text{ is not } \phi \text{ or } D$$

CTC Training

- **Forward computation $\alpha_t(s)$**



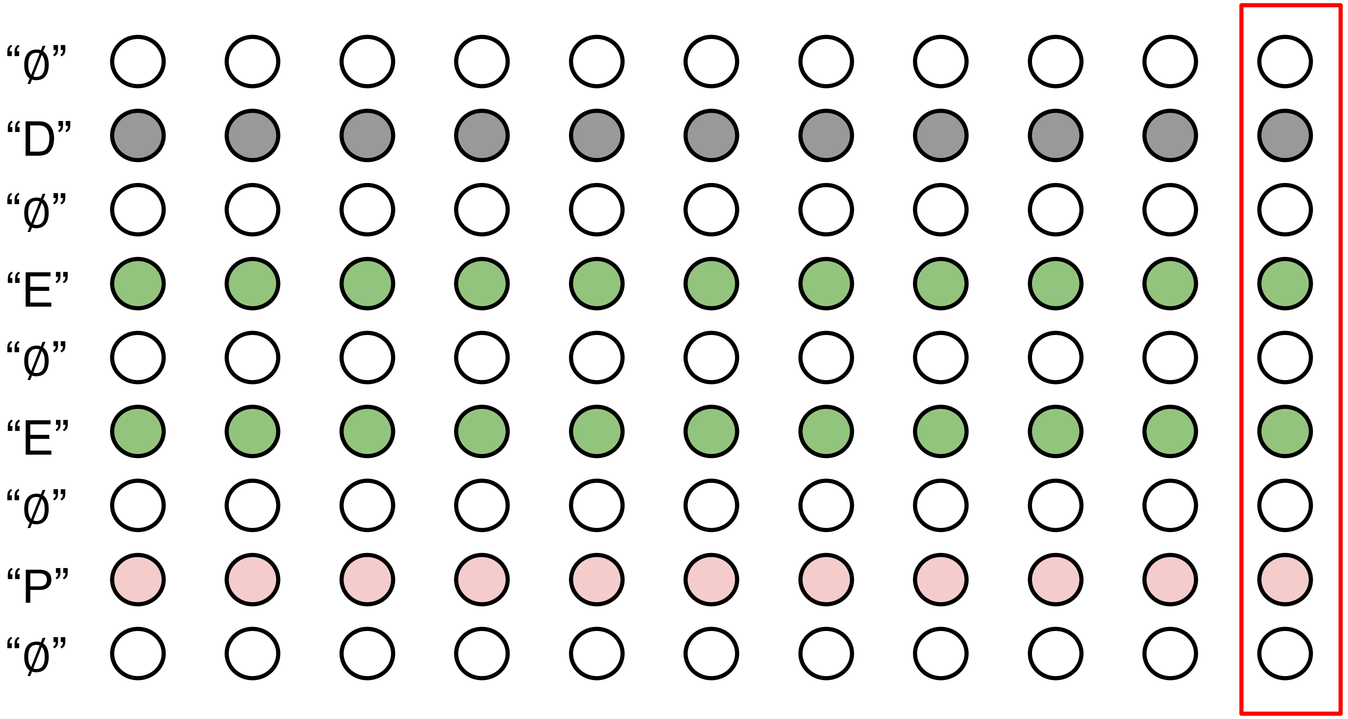
summed probability of **all** the CTC paths **ending** at time ***t*** with state ***s***

$$\boxed{\alpha_t(s)} \cdot y_t^s \cdot \beta_t(s)$$

$$a_t(s) = (a_{t-1}(s) + a_{t-1}(s-1))\, y_t^{l_s} \qquad \text{if } l_s = \phi \text{ or } l_s = l_{s-2}$$

$$a_t(s) = (a_{t-1}(s) + a_{t-1}(s-1) + a_{t-1}(s-2))\, y_t^{l_s} \qquad \text{otherwise}$$

- **Backward computation $\beta_t(s)$**

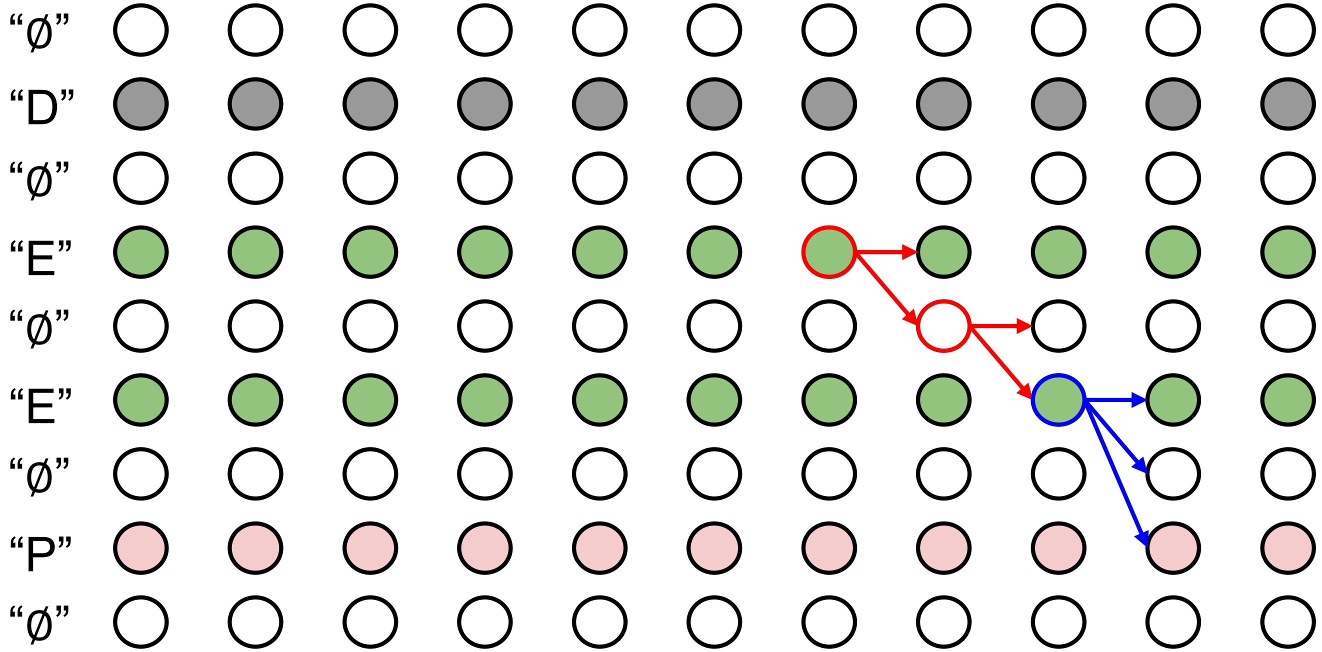


$$\alpha_t(s) \cdot y_t^s \cdot \boxed{\beta_t(s)}$$

summed probability of **all** the CTC paths **starting** at time ***t*** with state ***s***

$$\beta_T(\phi) = y_T^{\phi} \qquad \beta_T(P) = y_T^{P} \qquad \beta_T(l_s) = 0, \text{if } l_s \text{ is not } \phi \text{ or } P$$

- **Backward computation $\beta_t(s)$**



$$\alpha_t(s) \cdot y_t^s \cdot \boxed{\beta_t(s)}$$

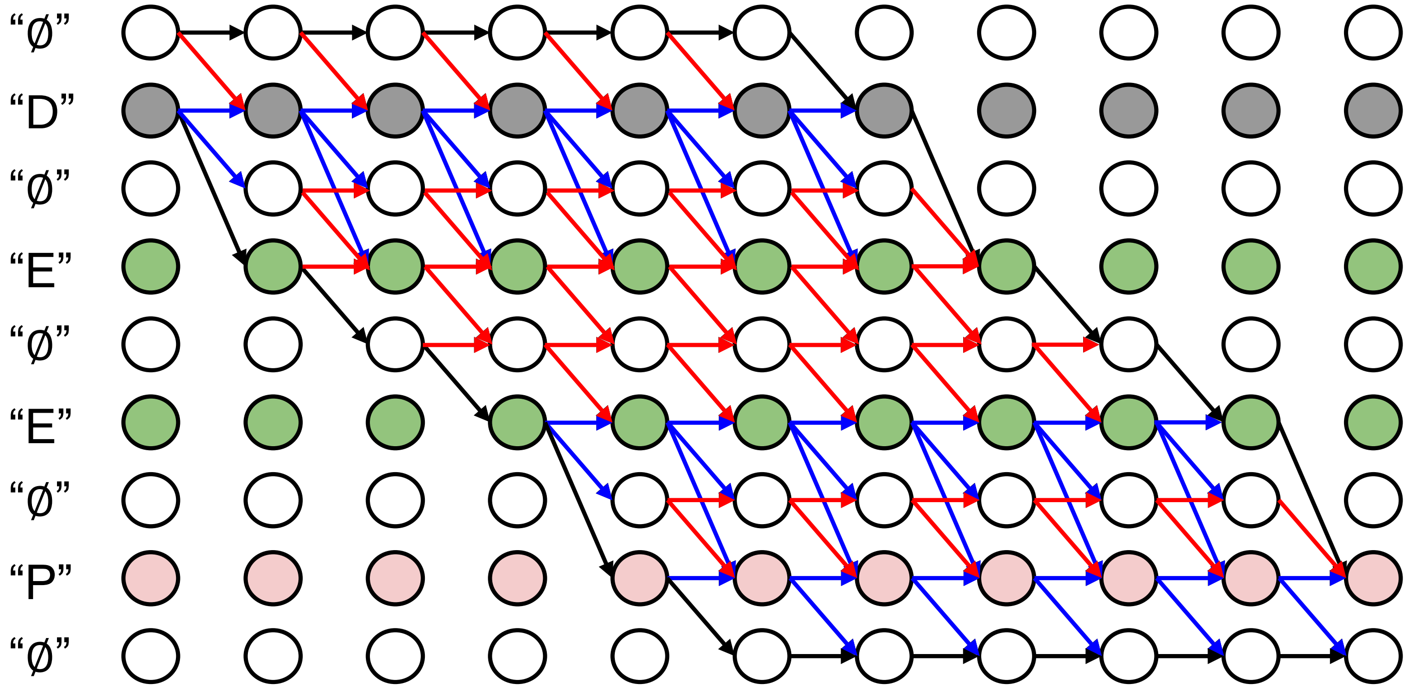
summed probability of **all** the CTC paths **starting** at time ***t*** with state ***s***

$$\beta_t(s) = y_t^{l_s}\left(\beta_{t+1}(s) + \beta_{t+1}(s+1)\right) \qquad \text{if } l_s = \phi \text{ or } l_s = l_{s+2}$$

$$\beta_t(s) = y_t^{l_s}\left(\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2)\right) \qquad \text{otherwise}$$

Universität Augsburg University

- **Total CTC Loss**



$$L_{CTC} = -\ln Pr(Y \mid X) = -\ln \boxed{\sum_s \alpha_t(s)\beta_t(s)}$$

compute gradients to update the weights

Universität Augsburg University

Best path decoding

- Compute the *best path* by taking the most likely prediction per frame

$$p^{\star} = \operatorname{argmax} Pr(p|X)$$

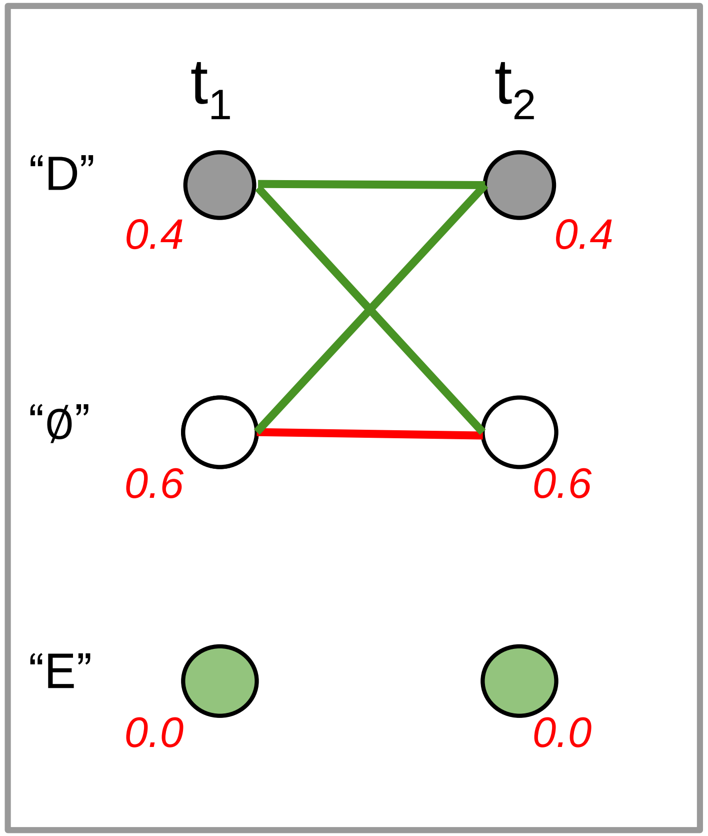
- map the best path to a label sequence by removing duplicated successive predictions and by removing all blanks from the path

\+ *simple and fast*

\- *lead to errors in many practical situations*

Universität Augsburg University

A toy example



Best path decoding outputs "blank"

Pr(Y=blank) = Pr(p=ØØ)
= 0.6 * 0.6
= 0.36

Pr(Y=D) = Pr(p=DD) + Pr(p=DØ) + Pr(p=ØD)
= 0.4*0.4 + 0.4*0.6 + 0.6*0.4
= 0.64

output labelling "D" is better in fact