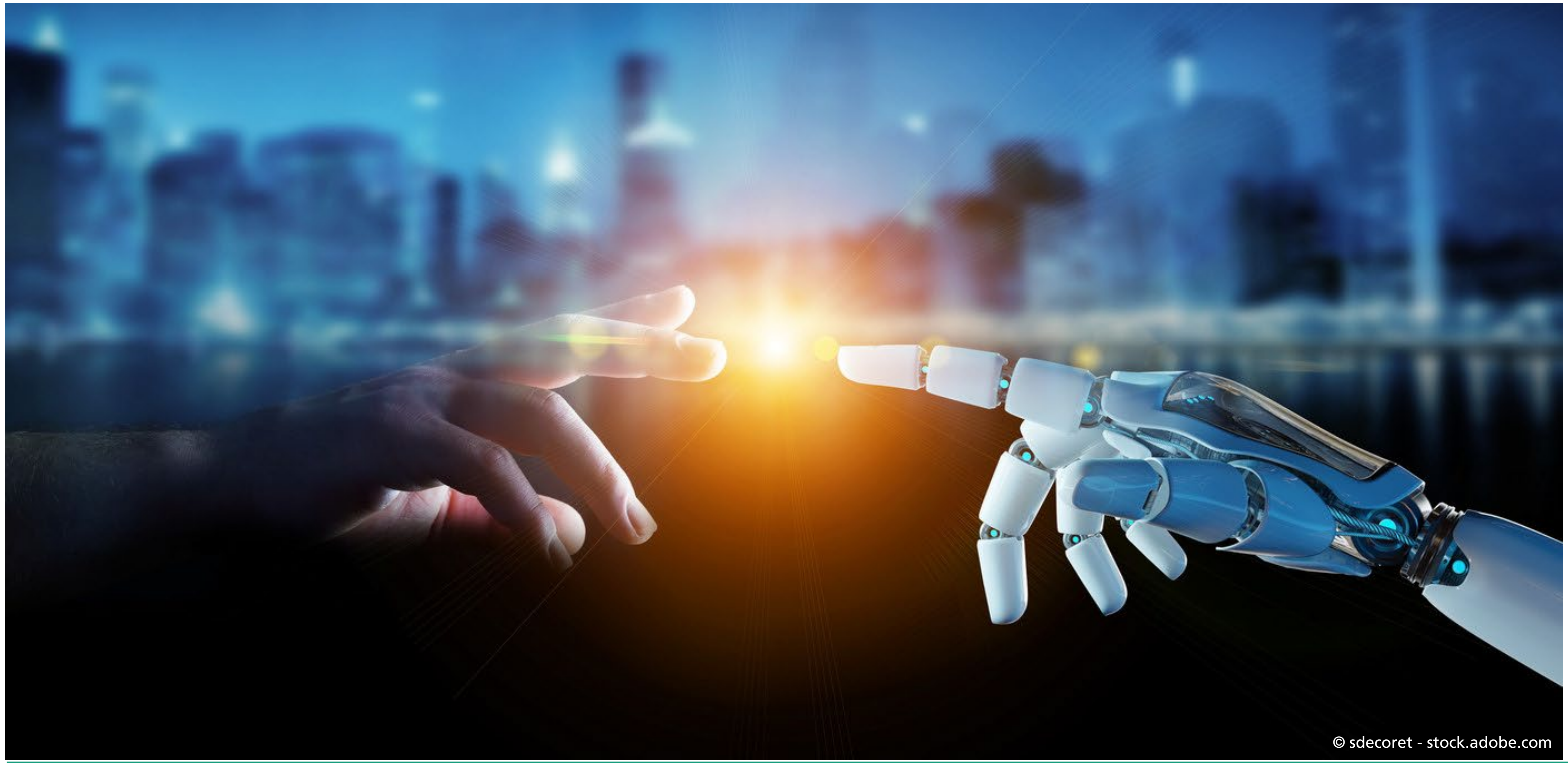
Generating Text Sequences

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



TensorFlow

Nicht zur Veröffentlichung! März 2024

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
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

Generating Text Sequences

Agenda

Language Model

$The\ cat\ sat\ on\ the$ mat

"Day 37: My Cat" cropped by Dusty J / CC BY 2.0

- **Predict** the current word from the past words "The cat sat on the"
- **Language Model**:
  Compute the probability $p(mat|The\ cat\ sat\ on\ the)$
- Recovers probability of the whole text
  - $\text{p}(The\ cat\ sat\ on\ the\ mat)\ = p(mat|The\ cat\ sat\ on\ the) \cdots p(cat|\ The)p(The)$

Applications

- Help to distinguish correct word order

$$p_{LM}(the\ house\ is\ small) > p_{LM}(small\ the\ is\ house)$$

- Decide if a text belongs to one writer from a set of possible writers

$$p_{Shakespeare}(To\ be, or\ not\ to\ be, that\ is\ the\ question)$$
$$> p_{Dickens}\ (To\ be, or\ not\ to\ be, that\ is\ the\ question)$$

G. language model

Nicht zur Veröffentlichung! März 2024

Fraunhofer BIG DATA AI

Capturing Long-Range Dependencies

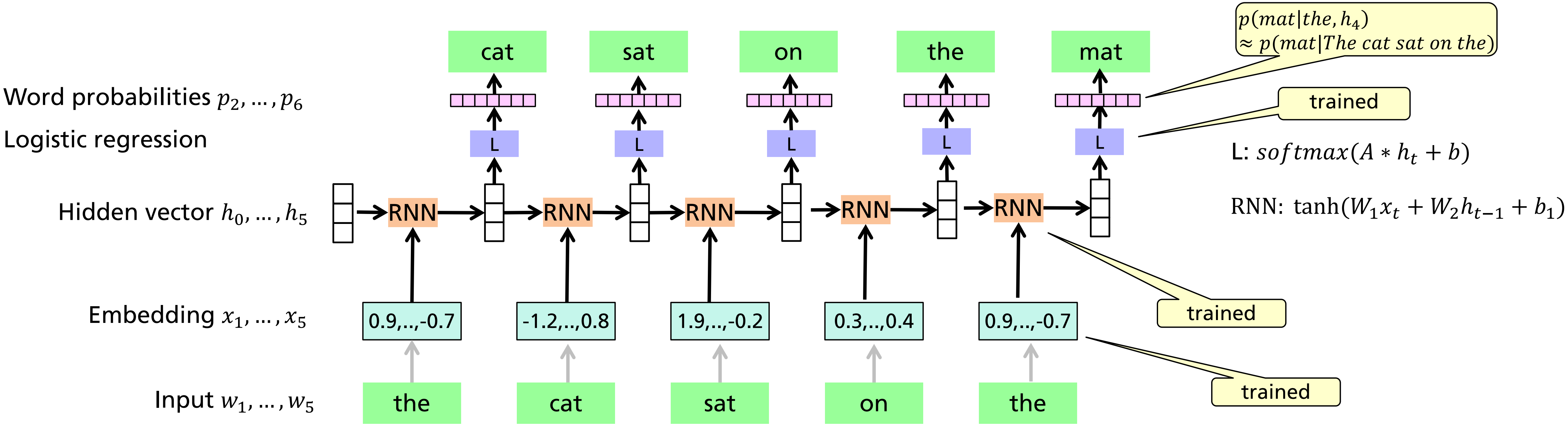
- approximate by **n-gram language model**: $p(x_t|x_{t-1},\cdots,x_1) \approx p(x_t|x_{t-1},x_{t-2},x_{t-3})$
  e.g. $p(x_t|\text{sat on the})$
  - n-grams for $n \geq 5$ are extremely rare ➔ uncertain probability estimates
  - loose too much information.
- Deep neural network
  - Has an input / output vectors of a **fixed** length
  - How to handle sequences of **varying length**,
    e.g. text documents, speech input?
- Idea: reduce to a task with vectors of **fixed length**
  - Use a **hidden vector** h_{t-1} to store information on past: $x_{t-1}, x_{t-2}, \cdots$
  - With input (h_{t-1}, x_{t-1}) predict (h_t, x_t)
  - Repeat this for each sequence element x_t.

Recurrent Neural Network

G. n-gram G. n-gram language model

Recurrent Network as Language Model

- The input is a sequence of words from a sentence.
- The first hidden vector h_0 is initialized, e.g. as $[0.0, \dots, 0.0]$, embeddings randomly initialized



- The hidden vector h_t contains information about **all** the past inputs $x_{t-1}, x_{t-2}, \cdots$
- It can potentially capture relations between distant inputs
- At each position t the RNN has the **same** parameters

[Elman 1990]

G. recurrent neural network

Fraunhofer BIG DATA AI

Simple Recurrent Neural Network

- input word w_t
- embedded input $x_t = lookup(w_t)$
- hidden variable
- probability of next word

embedding of input word

previous hidden vector

$$h_t = \tanh(W_1 x_t + W_2 h_{t-1} + b_1)$$

$$p_{t+1} = softmax(A_2 h_t + b_2)$$

RNN **advantages**

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep:
  ➔ translation invariant

RNN **disadvantages**

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Generating Text Sequences

Agenda

Fraunhofer BIG DATA AI

Recurrent Network as Language Model

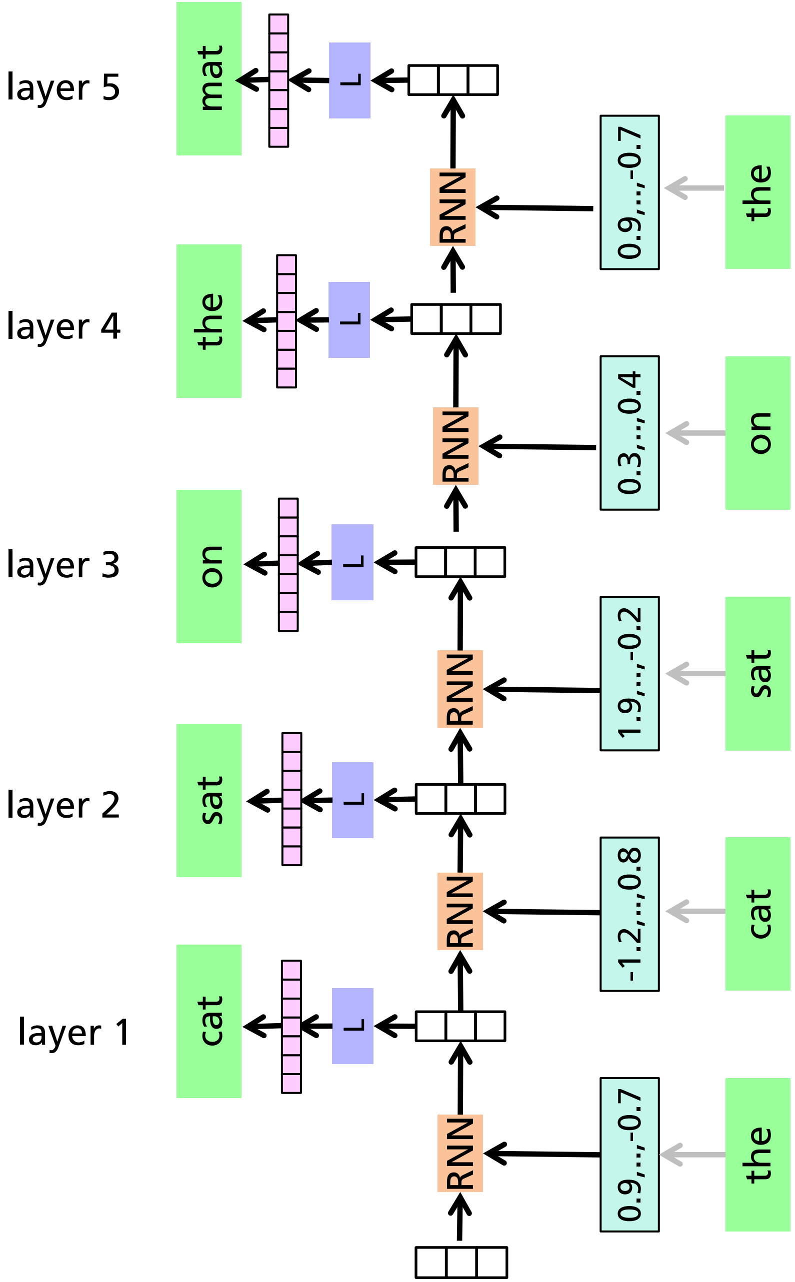
- network can be considered as a multilayer network: "unfolding in time"
- sentence length ➔ number of layers (dynamic)

Training on a big set of training documents

- Predict the network forward with previous words
  ➔ conditional prob. of next word given previous words
- **Total probability** of a sentence
  $\mathrm{p}(The\ cat\ sat\ on\ the\ mat) = p(mat|The\ cat\ sat\ on\ the) \cdots p(cat|\ The)p(The)$
  - Maximize product of conditional probabilities
    ➔ minimize sum of negative log of probabilities

loss $L(w) = -\log[p(mat|The\ cat\ sat\ on\ the)] - \cdots - \log[p(The)]$

- backpropagate gradients back along sentence: opposite direction of arrows
- Optimize by **stochastic gradient**: use only a minibatch of few sentences



Training the RNN

- **Backpropagation through time**
  - Forward propagation of activations along unfolded network in time
  - Backpropagation of derivatives along unfolded network in time
  - many derivatives $\partial L(w)/\partial w_i$ for the same parameter w_i
    ➔ all are added up
- Consider a very simple model

$$h_t = w * h_{t-1} + b * x_t$$

$$h_t = w^k * h_{t-k} + \cdots$$

$$\partial h_t / \partial h_{t-k} = w^k * \cdots$$

also for $\tanh(w * h_{t-1})$

- Ignoring the inputs we have after k timesteps:
  - For large k
  - for $w > 1$ the value w^k gets very large
    ➔ **exploding gradient** for long-range effects
  - for $w < 1$ and large k the value w^k gets very small
    ➔ **vanishing gradient** for long-range effects

$2^{20} = 1048576$

$0.5^{20} = 0.00000095$

"Explosion (Movie Park Germany)" by gnislew / CC BY-ND 2.0

G. exploding gradient G. vanishing gradient

Mitigation Approaches

- Large gradient moves parameter to a region with **constant** loss function ➔ no chance to escape
  - ➔ Make the gradient component smaller but keep its direction
- **Gradient Clipping**
  - Shorten gradient to a maximal length, the direction of the gradient is kept

Cheap and effective

- **Vanishing Gradient**: the effect of a gradient component in far distance is extremly small
  - Gradient effect often cancelled by the noise induced by stochastic gradient descent
- Construct a network that learns
  the **extent** and **duration** of long-range effects

LSTM Long Short-Term Memory

[Hochreiter, Schmidthuber 1998]

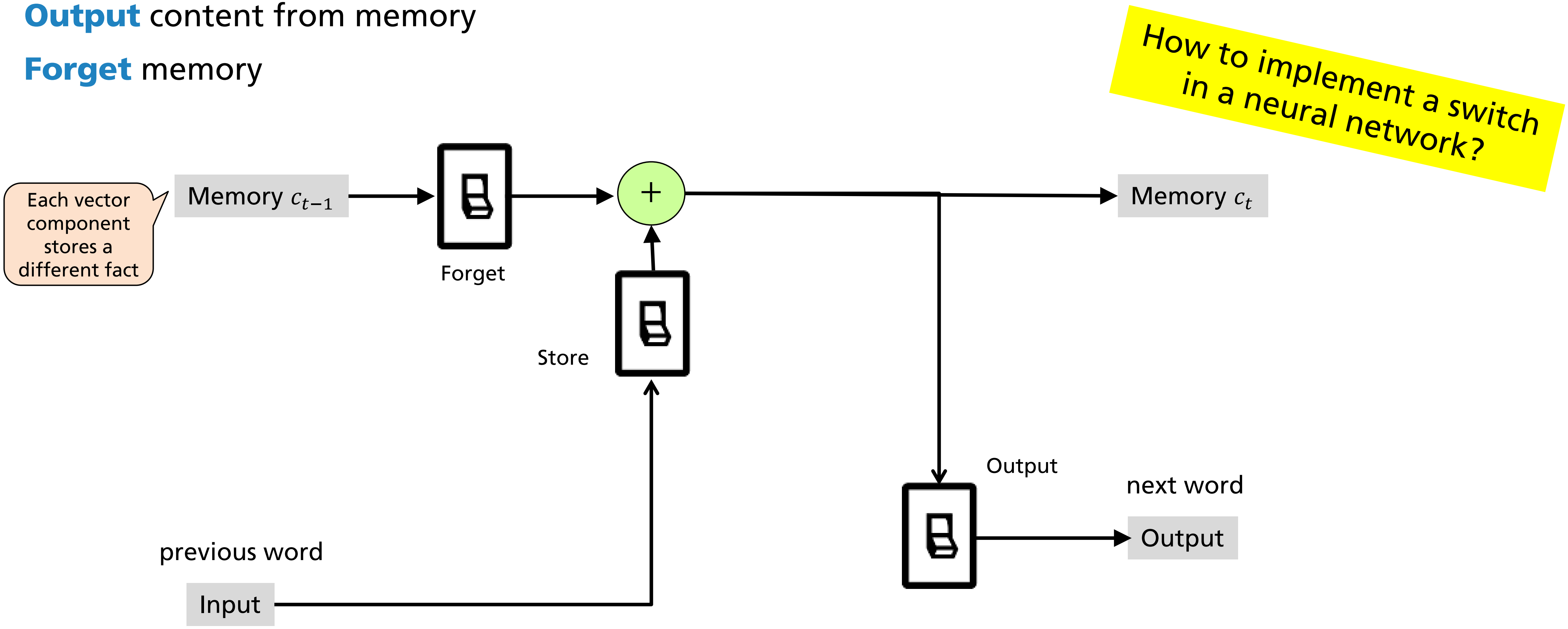
G. exploding gradient

Generating Text Sequences

Agenda

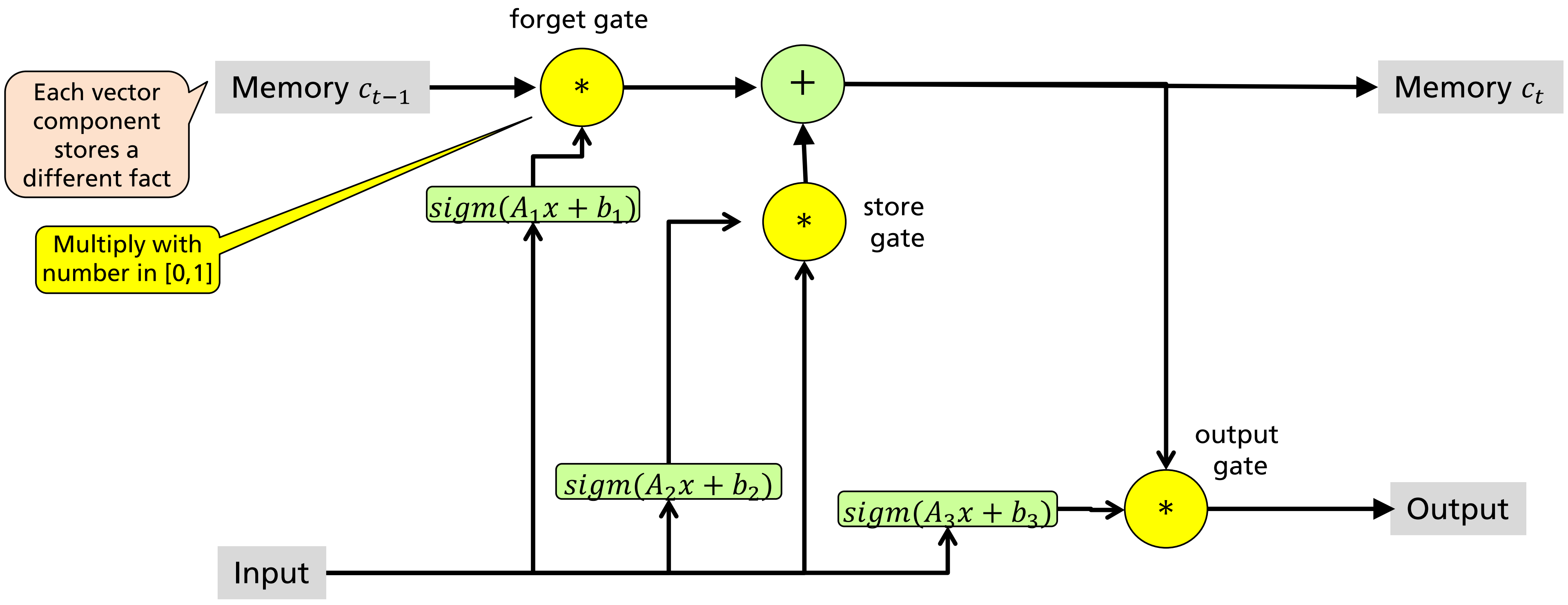
Functionality of a Memory Device

- **Store** content into memory: each vector component stores a different fact as a number
- **Output** content from memory
- **Forget** memory



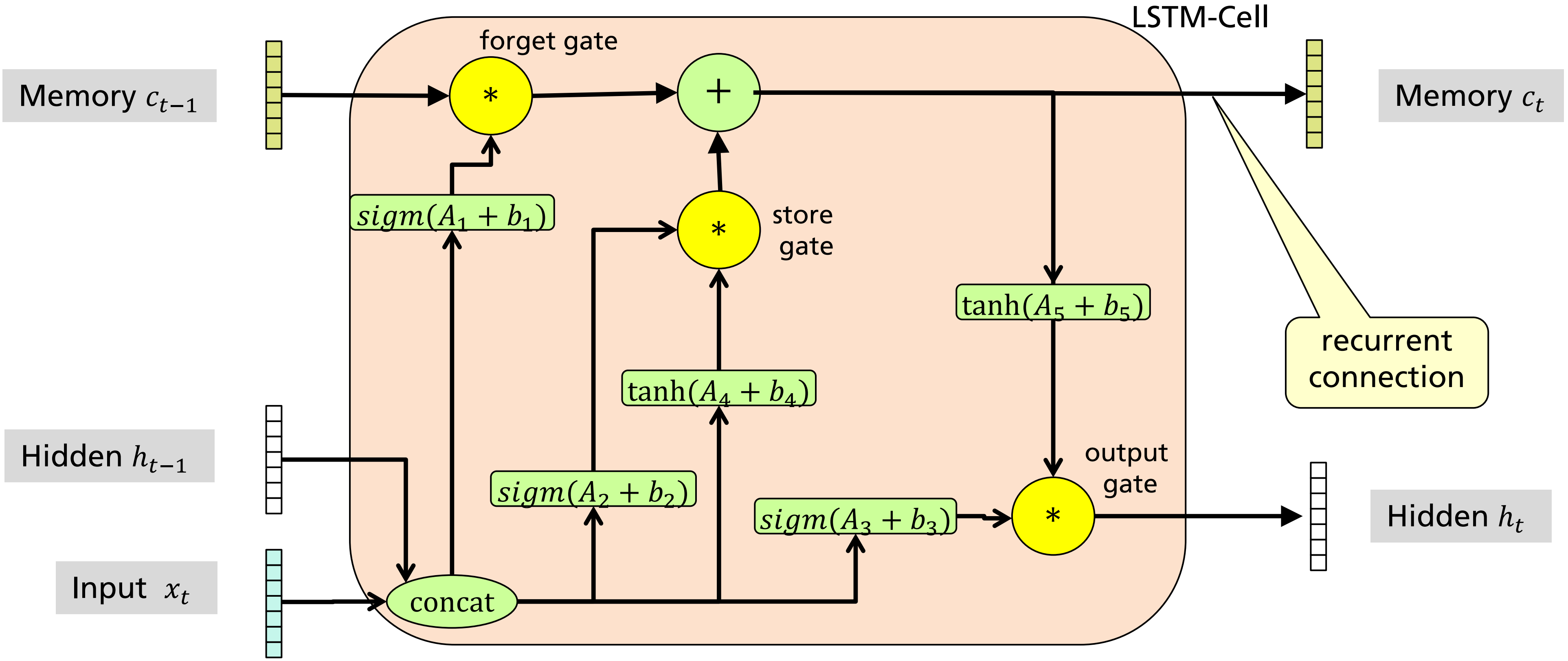
LSTM Cell

- Gate values: Compute by a **separate neural network** with sigmoid output in [0,1]
- Functions are smooth ➔ train by Backpropagation
- LSTM Cell automatically determines how long to keep a memory

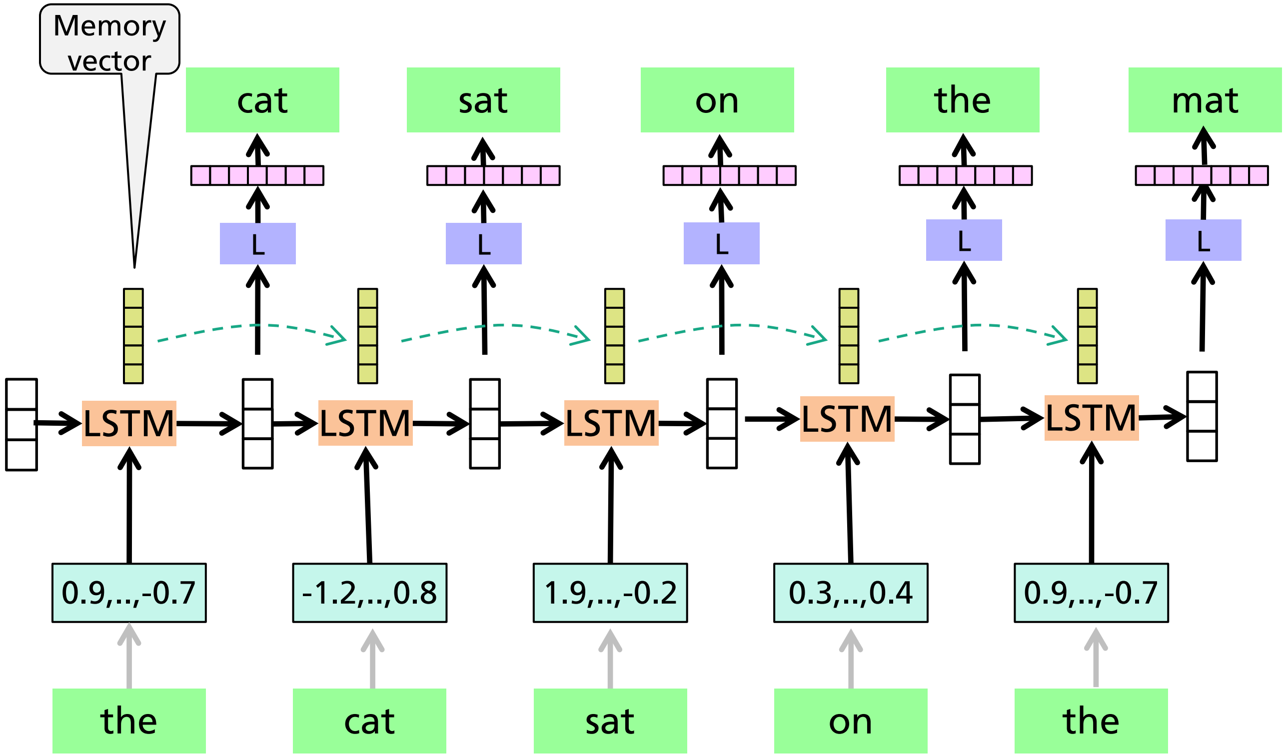


Connect to Inputs / Outputs

- The output usually is not observed: hidden vector h_t
- As input use concatenation of input x_t and hidden h_{t-1}
- Use $tanh$-transformations to generate new memory / hidden vector



Hidden and Memory Vectors



- Each component of the memory vector covers a **specific aspect**
- Need many components to cover all relevant aspects: e.g. $k = 200$
  ➔ Many components in hidden vector / memory vector

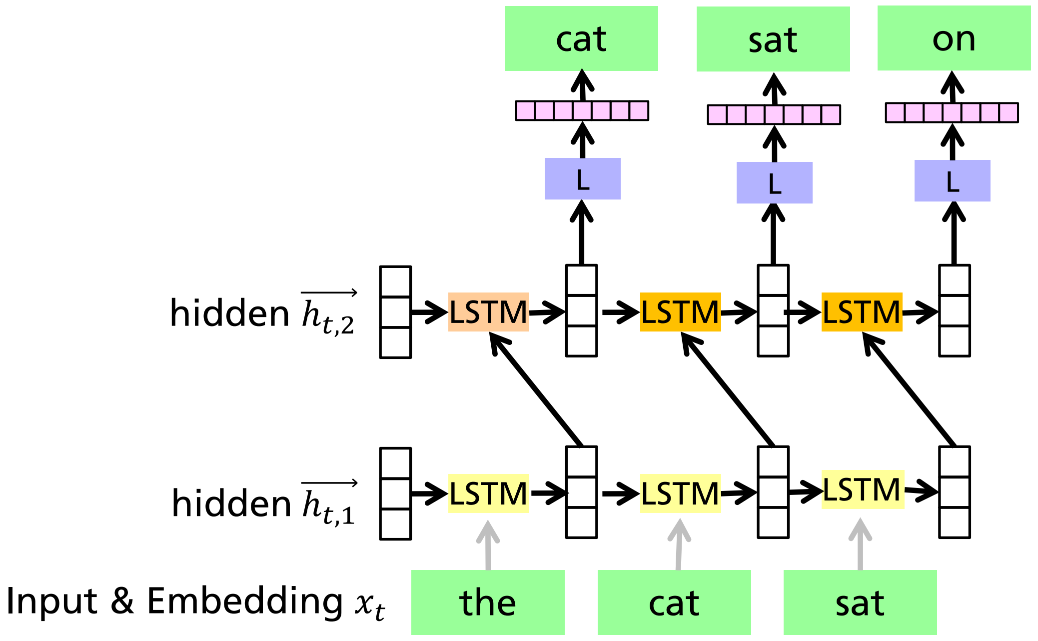
- hidden vector & memory vector are latent units
  - Memory vector c_t: long-term effects
  - Hidden vector h_t: short-term effects

Long Short-Term Memory

- During training the network automatically …
  - Estimates the internal parameters $A_1, b_1, \dots, A_5, b_5$
  - Determines all important aspects (hidden / memory vector components)

LSTM in Several Layers

- Increase the representional power of LSTMs
- Hidden vector of layer i is input for LSTM cell of layer $i+1$
- Marked increase of accuracy
- need more training data and regularization



Generating Text Sequences

Agenda

1. Motivation
2. Training an RNN
3. Long Short-Term Memory
4. TensorFlow: RNN Implementation
5. RNN Evaluations
6. Text Generation with GPT
7. Time Series Analysis
8. Summary

Preparing the Data

MSCOCO

Captions for images: total of ~400k descriptions

- download MSCOCO
- select 1000 different words: vocabulary
- end-of-sentence marker <eos>, special symbol <unk> for rare words

```
vocabulary = len(word_to_id)        #1000
len(train_data)                     #414113
```

- assign words to numbers

```
Original string:  A very clean and well decorated empty bathroom
Sequence of Word Ids:      [1, 77, 103, 3, 335, 245, 150, 8]
Max Sequence Length 38
```

- Each sentence is padded with 0 (=END) to fixed max_length

```
a very clean and well decorated empty bathroom END END END END END END END END END END END END END
END END END END END END END END END END END END END END END END END END
```

Model Specification using Keras

- Define model input

```
words = Input(batch_shape=(None, maxSequenceLength))
```

output shape = (batch_size, maxSequenceLength)

- Lookup embeddings from word codes, `emb_size = 300`

```
embeddings = Embedding(vocab_size, emb_size)(words)
```

output shape = (batch_size, maxSequenceLength, `emb_size`)

- First recurrent layer `hid_size = 500`, use dropout `dropout=0.3`

```
hiddenStates = LSTM(hid_size,     # may use GRU or LSTM
                    return_sequences=True,
                    input_shape=(maxSequenceLength, emb_size),
                    dropout=dropout)(embeddings)
```

output shape = (batch_size, maxSequenceLength, hid_size)

Model Specification using Keras

- Apply the Dense network to each position generating a probability

```
denseOutput = TimeDistributed(Dense(vocabularySize))(hiddenStates)
```

```
predictions = TimeDistributed(Activation("softmax"))(denseOutput)
```

output shape = (batch_size, maxSequenceLength, vocab_size)

- Define the model by specifying input and outputs

```
model = Model(inputs=words, outputs=predictions)
```

- Specify loss function and optimization algorithm, compile model

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer= RMSprop(lr=0.001))
```

Define Data

- Define
  - input: sequence without last element
  - output: sequence starting with 2nd element

```
input
 [   1  77 103   3 335 245 150   8   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0]
```

```
output
 [ 77 103   3 335 245 150   8   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0]
```

Train the Language Model

- Compute statistics (loss) on validation data after each epoch
- Afterwards save the model weights

```
model.fit(padded_seqs[:, :-1],                           # words 1, 2, 3, ... , (n-1)
          np.expand_dims(padded_seqs[:, 1:] , -1),       # words 2, 3, 4, ... , (n)
          epochs=2, batch_size=256)
model.save_weights('example_model/ptb_model_weights.h5') #Save model
```

```
Train on 1820 samples, validate on 260 samples
Epoch 1/2
1820/1820 [==============================] - 97s 53ms/step - loss: 7.0466 - val_loss: 6.5286
Epoch 2/2
1820/1820 [==============================] - 96s 53ms/step - loss: 6.7005 - val_loss: 6.5357
```

Sampling a Complete New Sentence

- Start with some first word, e.g. ‘a’
- Compute probabilities for next word
- Randomly select next word according to probabilities

```
a          = given first word
bathroom
features
light
stands
yellow
beside
the
END
END
```

Generating Text Sequences

Agenda

1. Motivation
2. Training an RNN
3. Long Short-Term Memory
4. TensorFlow: RNN Implementation
5. RNN Evaluations
6. Text Generation with GPT
7. Time Series Analysis
8. Summary

Is LSTM Optimal?

- Details of the LSTM cell are very heuristic
  - There are many alternatives: gates, activation functions, etc. Which alternative is the best?
- Empirical investigations
  - generate 10000 different RNN architectures [Jozefowicz et al. 2015]
  - Evaluate them on 4 different tasks
- Result
  - LSTM with additional **forget bias** on all task has good performance
  - The forget gate and the output activation function are the most crucial components of the LSTM

See also [Greff et al. 15], [Melis et al. 2017]

- **Disadvantages of RNN**:
  - Exploding and vanishing gradients
  - Very difficult to communicate **long-distance correlations** between words.
  - Each word has a single embedding: **context-dependent meanings** are mixed

RNN Application: Shakespeare

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- **Character-level** language models
  - predict the next character of a word sequence
  - small vocabulary, but longer dependency ranges
- Shakespeare: train on all works of Shakespeare (4.4.MB)
  - 3-layer RNN with 512 nodes on each layer

PANDARUS: Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

Second Senator: They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

DUKE VINCENTIO: Well, your wit is in the care of side and that.

Second Lord: They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

Clown: Come, sir, I will make did behold your worship.

VIOLA: I'll drink it.

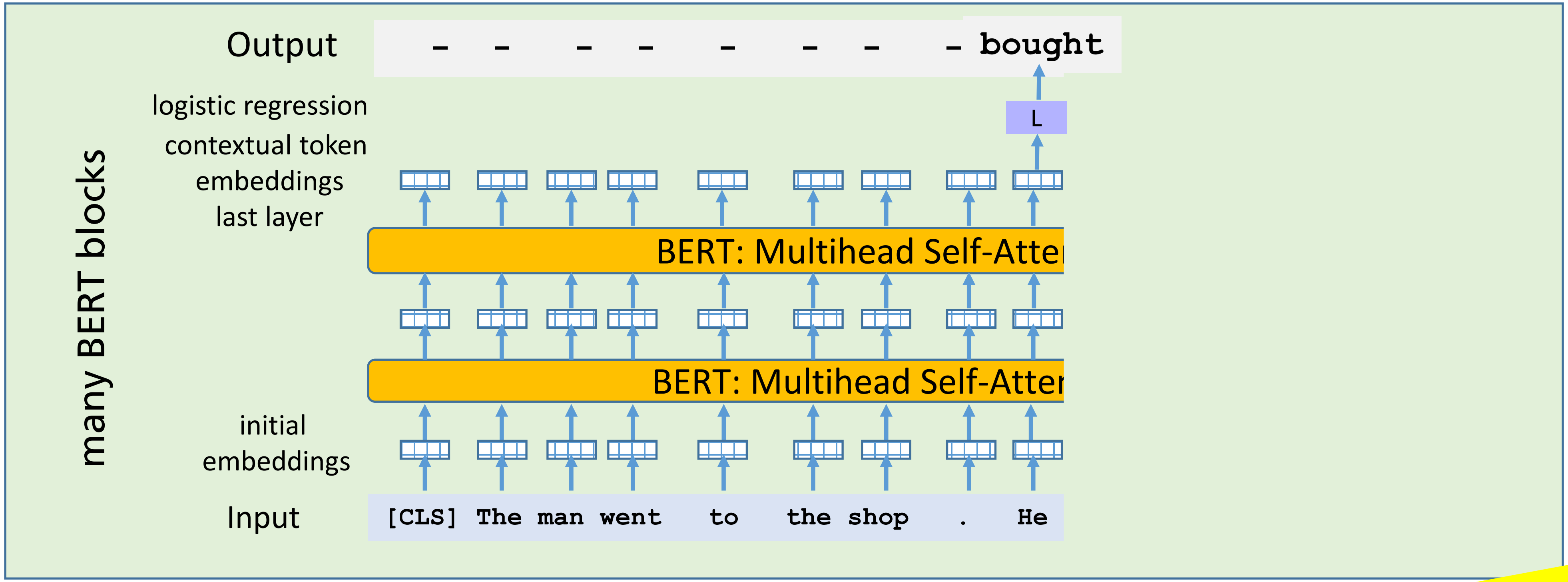
Recurrent LSTM models have large deficits

Generating Text Sequences

Agenda

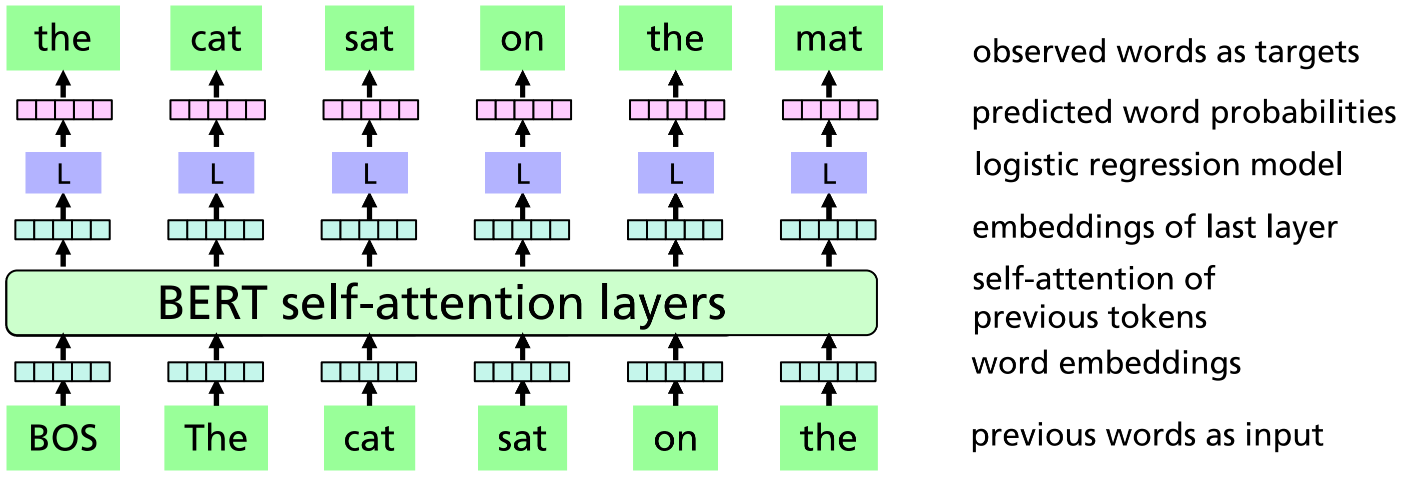
GPT: Use BERT to Predict the Next Token

- BERT predicts the **masked tokens based on full text**
  - logistic regression using last layer embeddings
- Variation to predict next word:
  compute self attention using **previous words** only
- predict next word using the contextual embedding of the last word



GPT Language Model

GPT Model Training



- **Language model**
  - successively predict probability of next observed word
  - use context-sensitive embedding of last layer
  - predict probabilities by logistic regression
- **Total probability** of a sentence for a given parameter w

  $$\mathrm{p}(The\ cat\ sat\ on\ the\ mat; w) = p(mat|The\ cat\ sat\ on\ the; w) * \cdots * p(cat|\ The; w) * p(The; w)$$

  - Change w to maximize product of conditional probabilities
    ➔ minimize sum of negative log of probabilities

loss $L(w) = -\log[p(mat|The\ cat\ sat\ on\ the; w)] - \cdots - \log[p(The; w)]$

- Optimize by **stochastic gradient**:
  - compute gradient $\partial loss\ L(w)/\partial w$ by backpropagation
  - use only a minibatch of few sentences

GPT Model Pre-Training

- **Pre-Training**: increase probabilities of observed next words
  - pretraining on a large corpus
- **Benefit of pre-training**
  - build strong representation of language: good parameter initialization
  - Probability distributions over language that we can sample from
  - A pre-trained model stays **close** to the pre-trained parameters when finetuned on a small dataset
    ➔ does not forget the learned language contents
- **Fine-tuning (optional)**
  - adapt to special language corpus, e.g. song lyrics
    ➔ model transfers knowledge from pretraining corpus
  - supervised training, e.g. for classification task

Transfer learning

GPT Model Text Generation

- **Prompt**: user provides a starting text
  - give a context for text generation,
  - indicate the style of the desired continuation:
    e.g. question, beginning of a story, song lyrics
- **Predict Next Token**
  - compute contextual embedding for the already generated text
  - predict token probabilities for the next position from the last embedding using logistic regression
  - **randomly select** the next token according to these probabilities
- **Selection strategies for next token**
  - select according to probabilities: probabilities of rare words are unrealiable
  - **top-k sampling**: select among k tokens with highest probability
  - **top-p sampling**: select among tokens up to a probability sum of p

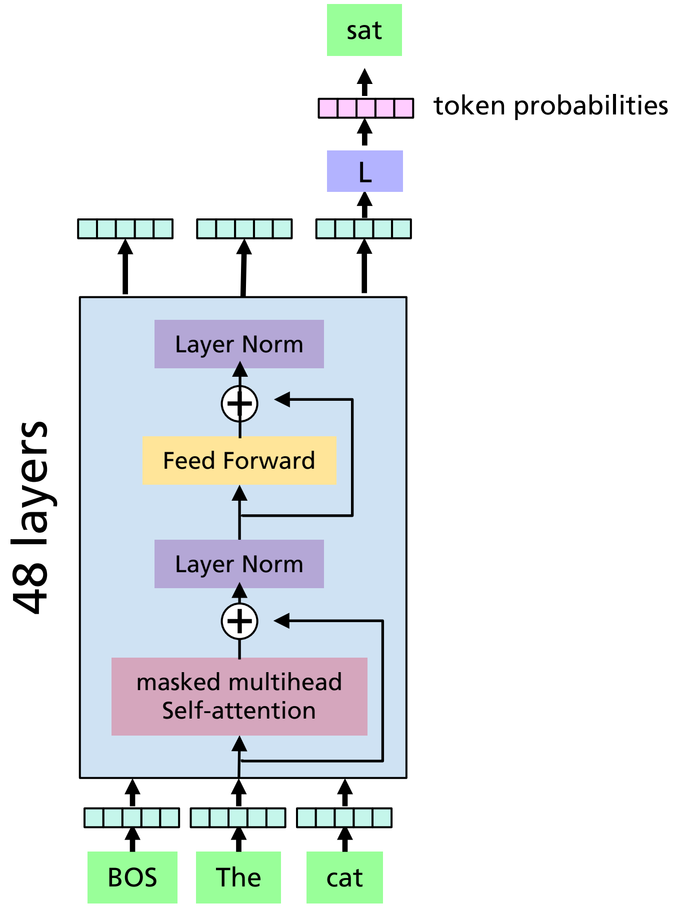
ensures diversity

- **Continue** with next but one token

GPT2 Details

- Input: Byte-Pair Encoding: character sequences for rare words
  - for each position t a token & position embedding $emb(u_t) + emp(t)$
- Output: probability of next token

- GPT2:
  - 1.5B parameters,
  - 48 layers
  - Input length 1024 token
- pre-trained on 40 GB of text
  - 8M docs,
  - linked from Reddit with at least 3karma.



Fraunhofer BIG DATA AI

Language model GPT2: Prediction Performance

- **Perplexity** = inverse likelihood per word
  - $x_1, \dots, x_n$ is a sample from the unknown probability distribution
  - $q(x)$ is a model estimating the true probability $p(x)$
  - perplexity: $[q(x_1) * \cdots * q(x_n)]^{-1/n}$

  Better models q of the unknown distribution p will tend to assign higher probabilities $q(xi)$.
  ➔ lower perplexity is better

- **state-of-the-art perplexity** in text generation
  - very good at exploiting information in far-away words

specific tasks

text generation

Data		prior SOTA	GPT2
LAMBADA	acc	59.2	63.2
ChildBT-CN	acc	85.7	93.3
ChildBT-NE	acc	82.3	89.1
WikiText2	perplexity	39.1	18.3
PTB	perplexity	45.5	38.8
enwik8	bpc	0.99	0.93
text8	bpc	1.08	0.98
WikiText103	perplexity	18.3	17.5

Zero-shot learning: no training on specific data

G. perplexity

Language model GPT2

- Generate a text from a given start sequence
  - top-k sampling: select among k tokens with highest probability
  - top-p sampling: select among tokens up to a probability sum of p
- Possible **Usage**:
  - AI writing assistants
  - More capable dialogue agents
  - Unsupervised translation between languages
  - Better speech recognition systems
- Possible **Dangers**:
  - Generate misleading news articles
  - Impersonate others online
  - Automatic production of faked content to post on social media
  - Automate the production of spam/phishing content

code and models

online access: https://talktotransformer.com/

Input

A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.

Generated by GPT2

The incident occurred on the downtown train line, which runs from Covington and Ashland stations.

In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief.

"The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation," said Tom Hicks, the U.S. Energy Secretary, in a statement. "Our top priority is to secure the theft and ensure it doesn't happen again."

The stolen material was taken from the University of Cincinnati's Research Triangle Park nuclear research site, according to a news release from Department officials.

The Nuclear Regulatory Commission did not immediately release any information.

According to the release, the U.S. Department of Energy's Office of Nuclear Material Safety and Security is leading

Language model GPT3

- Scaled-up version of GPT2:
  - embedding vector of length 12288
  - 96 layers, 96 embedding heads
  - batch size 3.2 M, 175 B Parameters
  - Training cost: 4.6 M Dollar
- Training data:
  ~ 500 B words from books and web
- performs many tasks given an **instruction**
  - e.g. *Translate to German: Peter went to Paris*
  - GPT3 *Peter fuhr nach Paris*
- Can be instructed with a few natural language **examples**: ➔ **few shot learning**
  - Accuracy sometimes better than after finetuning

Input

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for picking me as your designer. I appreciate it.

Poor English Input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did modifications.
Good English Output: The requested changes have been done. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I'd be more than happy to work with you in another project.

GPT3

Good English output: I'd be more than happy to work with you on another project.

[Brown et al. 2020]

New paradigm:
Solve a task without any training!

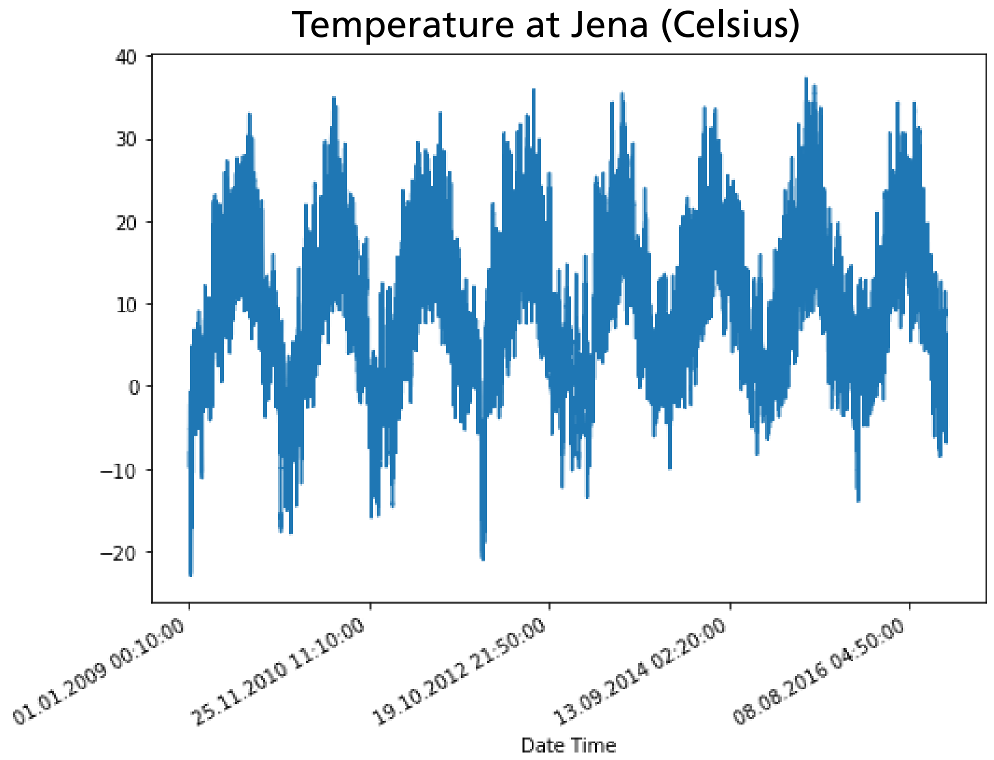
G. GPT-3
G. few-shot learning
G. instruction following

Generating Text Sequences

Agenda

Time Series

- A **time series** consists of variables recorded at different time points $t = 1,2, \ldots$
  - **univariate** time series: only one variable is recorded for each t, e.g. temperature
  - **multivariate** time series: a vector of variables is recorded for each t: e.g. temperature, wind speed
- **Examples of Applications**:
  - predict temperature in 10 minute intervals
  - predict temperature, wind speed, humidity in 10 minute intervals
  - Predict stock exchange prices for selected shares
  - Classify measurements for a wind turbine (normal or not normal):
    temperature, wind speed, vibration, humidity, generated current,...
  - Classify measurements for an intensive care patient (normal/alarm):
    Pulse, blood pressure, oxygen saturation, ...



Time Series

- **Autocorrelation**: next value is similar to the current
  - e.g. next day temperature at same time
  - seasonality: autocorrelation over the years
  - a time series depends on its own past values + past values of other variables (lagged variables)

- Time Series prediction:
  predict the mean value $\bar{x}_{t+1} = f(x_t, \ldots, x_{t-k}; w)$
  - Gaussian error $x_{t+1} \sim N(f(x_t, \ldots, x_{t-k}; w), \sigma^2)$
  - prediction of multiple values

- **stationary**: $f(\cdot)$ and σ^2 are independent of t

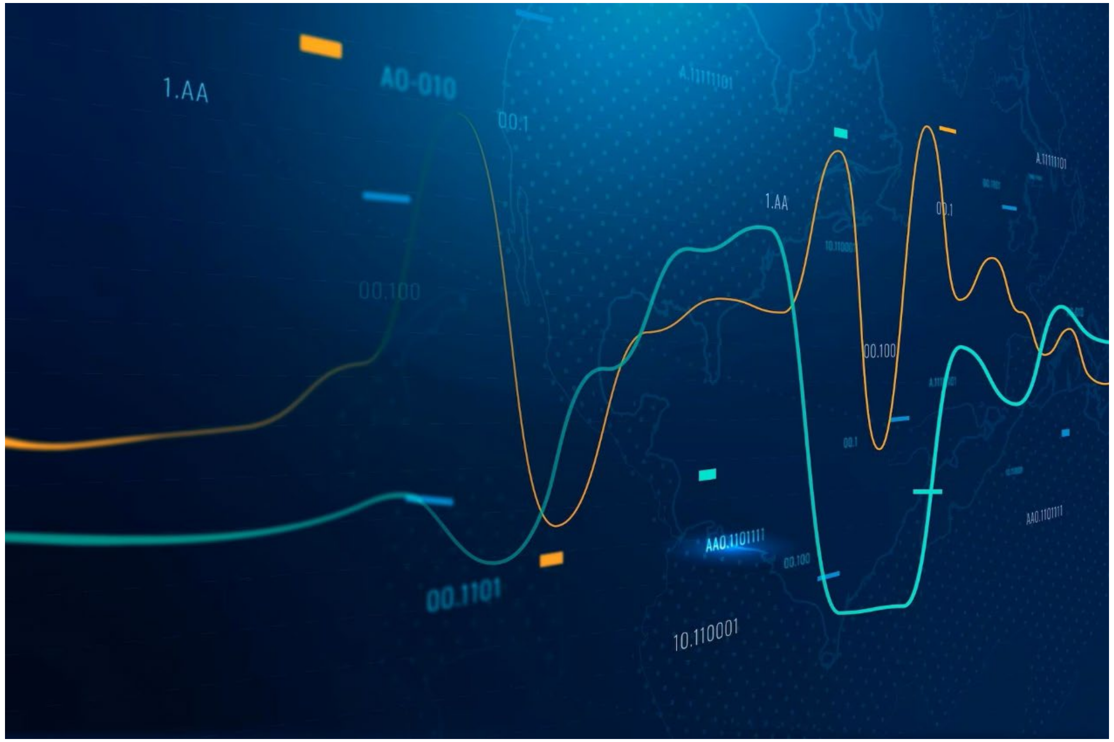


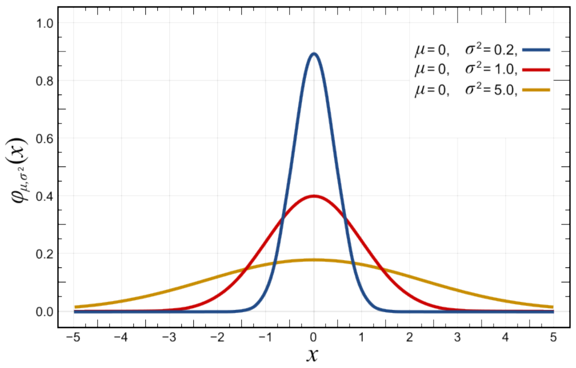
Image on rawpixel.com by Freepik

Fraunhofer BIG DATA AI

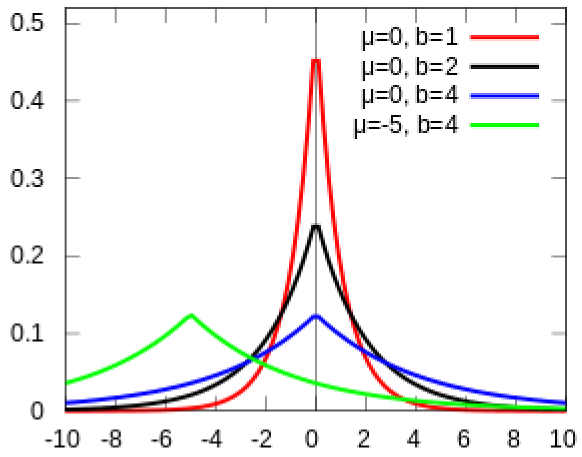
Estimation of Time Series Model

- Use an LSTM to predict the mean value of x_{t+1} given the past data $x_t, \ldots, x_{t-k}$
$$\bar{x}_{t+1} = LSTM(x_t, \ldots, x_{t-k}; \mathrm{w})$$
- Loss function: **Mean Square Error** $\mathrm{L(w)} = \frac{1}{T}\sum_{t=1}^{T}(\bar{x}_{t+1}(w) - x_{t+1})^2$
  - residual $\epsilon_t = \bar{x}_{t+1}(w) - x_{t+1}$ follows Gaussian distribution
- Loss function: **Mean Absolute Error** $\mathrm{L(w)} = \frac{1}{T}\sum_{t=1}^{T}|\bar{x}_{t+1}(w) - x_{t+1}|$
  - less influenced by outliers
  - residual$\epsilon_t = \bar{x}_{t+1}(w) - x_{t+1}$ follows Laplace distribution
- Estimation of w by **Stochastic Gradient Descent** (SGD)
  - normalize data $\tilde{x}_t = (x_t - mean(x_t))/standardDeviation(x_t)$ where mean and sd are determined from training data
  - Training data: pairs $[(x_t, \ldots, x_{t-k}), x_{t+1}]$
- **Checking assumptions**:
  - residual ϵ_t should be uncorrelated to other ϵ_{t-k} e.g. the Ljung-Box test https://en.wikipedia.org/wiki/Ljung%E2%80%93Box_test

different Gaussian distributions



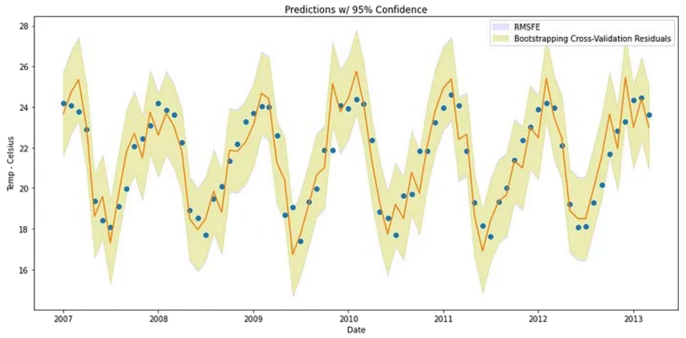
different Laplace distributions



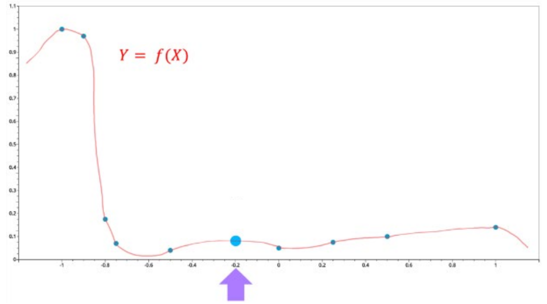
Details in the Time Series Course

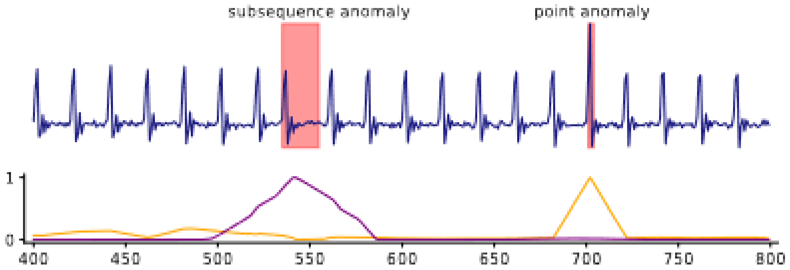
Time Series Analysis Task Groups

- **Prediction** of future values
  - start with actual values
  - predict one or more time steps, compute prediction interval
- Time series **interpolation**
  - use model to estimate values at intermediate positions
  - use information from past and future values
- Time series **anomaly** detection
  - compare to training set of „normal“ time series
  - determine significant deviations
- Time series **classification**
  - assign time series to different classes
  - supervised training of a classifier based on training data



images by author





Schmidl et al. 2022, Anomaly detection in time series, CC NC ND 4.0

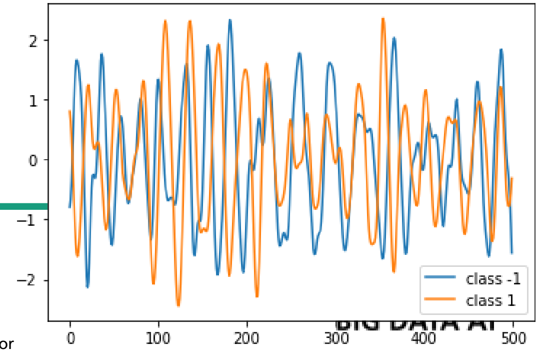
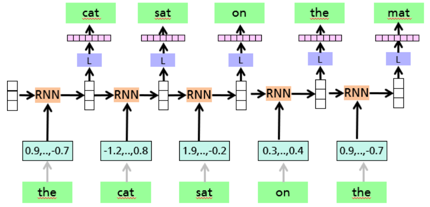
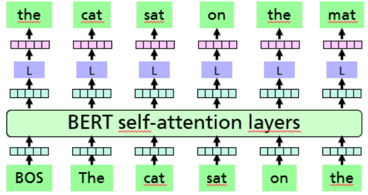


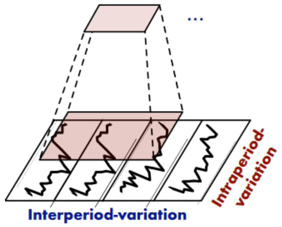
image by author

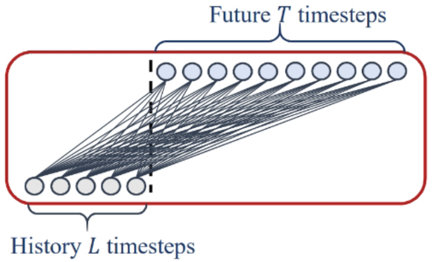
Main Algorithms for Time Series Prediction

- **Benchmark:** ETTh1 (720) Electricity Tansformer Temperature long sequence
- **Classical Algorithms:** Gluon-TS : MAE=0.766
  - ARIMA autoregressive moving average, state space models
- **Recurrent Neural Networks** LSTMa: MAE=1.322
  - state is stored in a hidden vector
- Attention Models like Transformers and **GPT** QuerySelector : MAE= 0.373
  - direct relation to far away inputs
- **Convolutional Neural Network** approaches SCINet: MAE=0.450
  - rearrange time series to 2D structures according to seasons
  - use 1D or 2D convolution layers like ResNet
- Direct **multi-step forecasting** (DMS) N-Linear: MAE=0.226
  - Create long-term predictions in a single step





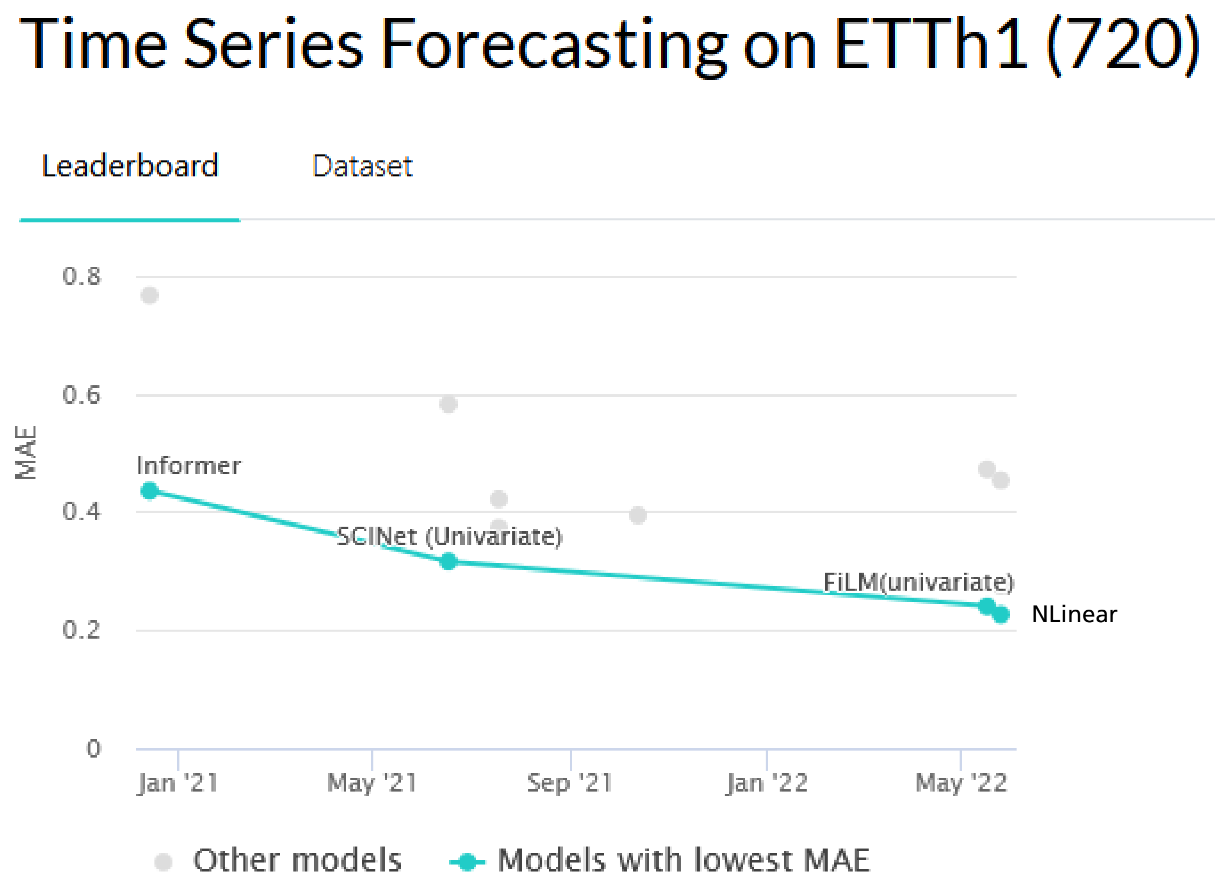




Details in the time series course

Time Series Prediction Benchmarks

- Many different time series approaches
- Comparison for different benchmarks:
  - Electricity Transformer Temperature (ETT)
- good source for new algorithms



Generating Text Sequences

Agenda

Summary

- Recurrent Neural Networks are effective in processing sequences
- **LSTMs** are able to capture long-term dependencies
- Possible targets:
  - next input: Word model
  - derivation of embeddings
  - analysis of numerical time series
- Training by backpropagation through time & stochastic gradient descent
- Use recursive modified BERT model to predict next word: **GPT Language Model**
  - only use previous words of a text to predict next word
  - multihead self-attention provides probability estimate
  - excellent knowledge about language and commonsense facts
- Time Series Analysis
  - Transformer is one possible alternative

Instructing Language Models:
New Paradigm of Artificial Intelligence

Disclaimer

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