RECURRENT NEURAL NETWORKS

Introduction to RNNs, LSTMs, Transformers

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER UNIVERSITY LINZ

Institute for Machine Learning

Overview

Standard recurrent neural networks (RNNs) and their great potentials

Sequence learning settings - using RNNs for different tasks

Classic RNNs and the Vanishing Gradients Problem

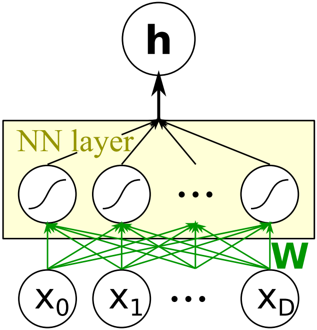
Long Short-Term Memory networks (LSTM)

Quick introduction to Transformers and modern Hopfield networks

Basics of Recurrent Neural Networks (RNNs)

Feedforward Neural Networks

- Size of input vector $\mathbf{x}$ is fixed
- Spatial relations of elements in sequence of inputs lost
  - No direct information about order of features
  - Restricted work-around:
    Windowing via convolution (1D CNN)



layer output $\mathbf{h} = \mathrm{act}\left(\mathbf{W}^T\mathbf{x}\right)$

M hidden units with
activation function $\mathrm{act}()$

weigth matrix $\mathbf{W} \in \mathbb{R}^{D \times M}$

input vector $\mathbf{x} \in \mathbb{R}^{D \times 1}$

Recurrent Neural Networks (RNNs) (1)

- Assume a sample is a sequence of length T with D features at each timestep t.
  - Each sample represented by matrix $\mathbf{X}$ of shape $T \times D$
  - T may vary between samples but D is constant

Recurrent Neural Networks (RNNs) (1)

- Assume a sample is a sequence of length T with D features at each timestep t.
  - Each sample represented by matrix $\mathbf{X}$ of shape $T \times D$
  - T may vary between samples but D is constant

- To address the mentioned limitations of feedforward networks, our network needs to:
  1. be able to handle variable sequence lengths T,
  2. remember previous inputs within a sequence

Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) (2)

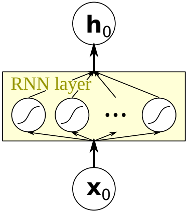
- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

  $\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$

Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

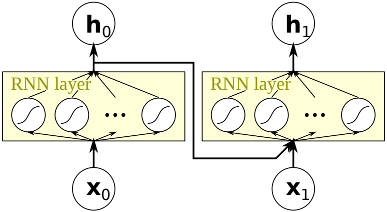
  $\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

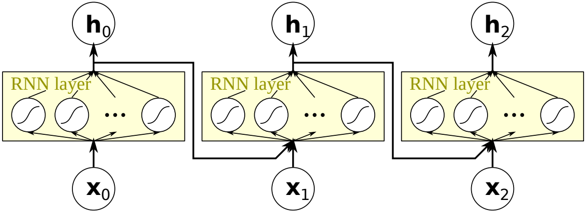
$\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

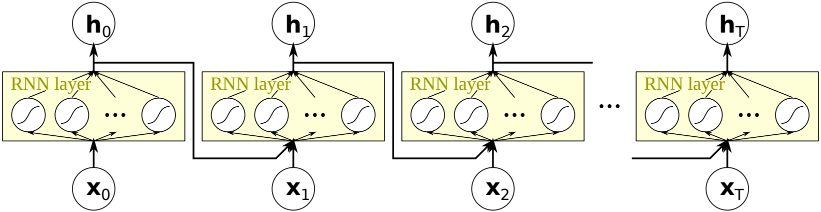
$\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

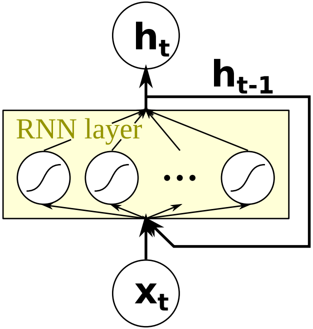
$\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Recurrent Neural Networks (RNNs) (2)

- Solution: Recurrent Neural Networks (RNNs)
  - Feed input sequence $\mathbf{X}$ timestep by timestep into network (= vector $\mathbf{x}_t$ with D features at each timestep t)
  - Add previous output $\mathbf{h}_{t-1}$ (=hidden state) to current input $\mathbf{x}_t$

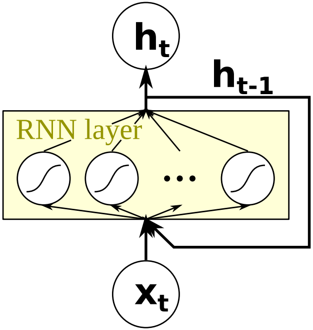
$\rightarrow$ $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are inputs to compute new $\mathbf{h}_t$: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Recurrent Neural Networks (RNNs) (3)

- Typically only single layers recursively connected
- Layer weight matrix $\mathbf{W}$ reused (=shared) for all timesteps
- Computation of $\mathbf{h}_t$ similar to feedforward networks:

$$\mathbf{h}_t = \text{act}\left(\mathbf{W}^T \cdot \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix} + \mathbf{b}\right)$$



Power of RNNs

- RNNs are in essence a state-space model: $y_t = f(x, y_{t-1})$
- Could be used for sequence classification, sequence generation, control systems, meta learning, ...

Power of RNNs

- RNNs are in essence a state-space model: $y_t = f(x, y_{t-1})$
- Could be used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically Turing Complete
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly (=you can simulate anything)

Power of RNNs

- RNNs are in essence a state-space model: $y_t = f(x, y_{t-1})$
- Could be used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically Turing Complete
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly (=you can simulate anything)
  - Like the Universal Approximation Theorem, this is mostly useless in practice

Power of RNNs

- RNNs are in essence a state-space model: $y_t = f(x, y_{t-1})$
- Could be used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically Turing Complete
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly (=you can simulate anything)
  - Like the Universal Approximation Theorem, this is mostly useless in practice
  - The hard problem is not "what can we represent?" but finding a good representation

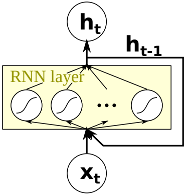
Power of RNNs

- RNNs are in essence a state-space model: $y_t = f(x, y_{t-1})$
- Could be used for sequence classification, sequence generation, control systems, meta learning, ...
- RNNs are theoretically Turing Complete
  - For each program in a turing complete programming language, you could find an RNN that executes the code correctly
  - Like the Universal Approximation Theorem, this is mostly useless in practice
  - The hard problem is not “what can we represent?” but finding a good representation

RNN Training

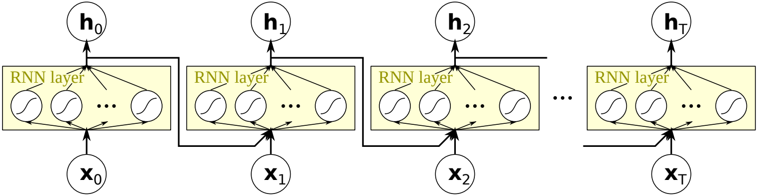
Unrolling an RNN

- RNN can be viewed as feed forward network with shared weights = unrolled over time



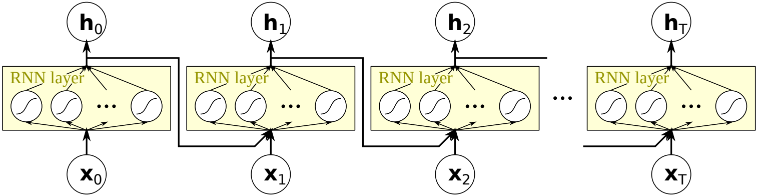
Unrolling an RNN

- RNN can be viewed as feed forward network with shared weights = unrolled over time



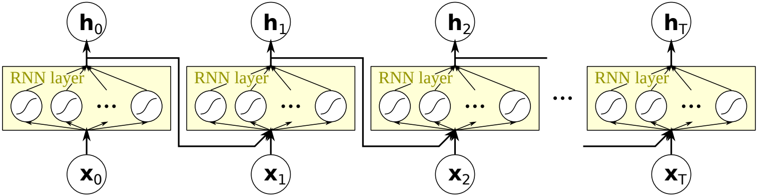
Back-Propagation Through Time (BPTT)

- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



Back-Propagation Through Time (BPTT)

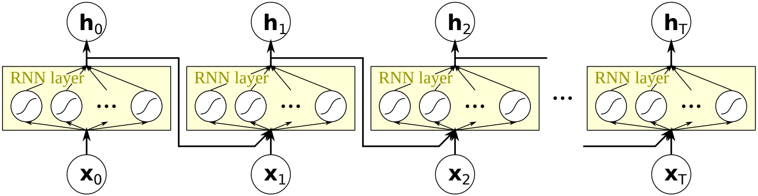
- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



- Complexity: $\mathcal{O}(N^2T)$
  - N: number of hidden units
  - T: Length of sequence

Back-Propagation Through Time (BPTT)

- Most common way to train RNNs: Back-Propagation Through Time (BPTT)



- Complexity: $\mathcal{O}(N^2T)$
  - N: number of hidden units
  - T: Length of sequence
- Truncated BPTT: only unfold n timesteps into the past

Real-Time Recurrent Learning (RTRL)

- Alternative to BPTT
- Computes all gradient information during forward pass
- Complexity $\mathcal{O}(N^4) \Rightarrow$ Independent of sequence length
- Very rarely used today

Sequence Learning Settings

Sequence Learning Settings (1)

- Alex Graves (2012) distinguishes 3 types of classification tasks for sequence data:

Sequence Classification: 1 label per sequence

Task	Input	Prediction
Predict color of rose:	The rose is red.	{"white", "red"}

Segment Classification: 1 label per part of sequence

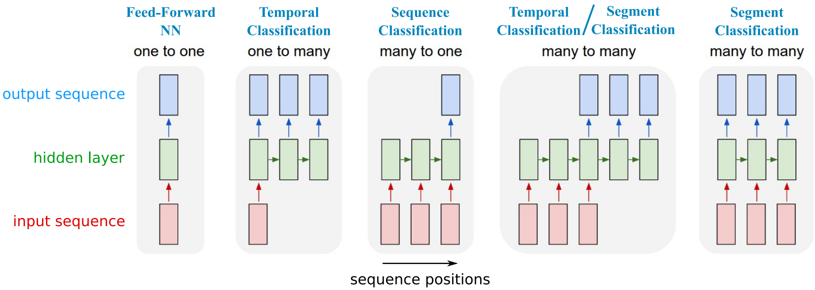
Task	Input	Prediction
Segment colors in sequence:	The rose is red.	The rose is red.

Temporal Classification: Sequence of labels per sequence

Task	Input	Prediction
Translate to German:	The rose is red.	Die Rose ist rot.

Sequence Learning Settings (2)

- Processing data using RNN layers (Karpathy, 2015)



[Taken with modifications from *The Unreasonable Effectiveness of Recurrent Neural Networks*, A. Karpathy, 2015]

Vanishing Gradients Problem

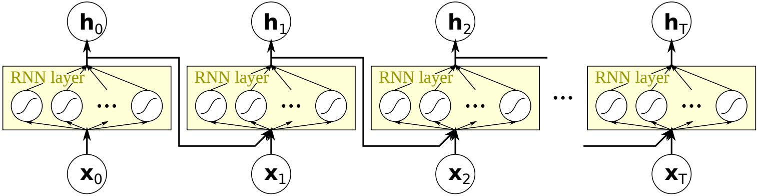
- BPTT generates very deep networks ($T \cong$ depth)

$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)

Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

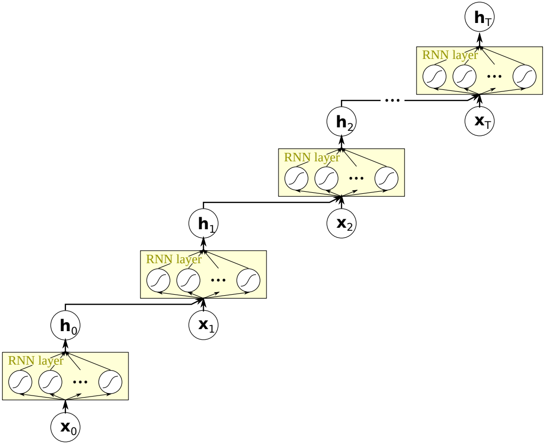
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

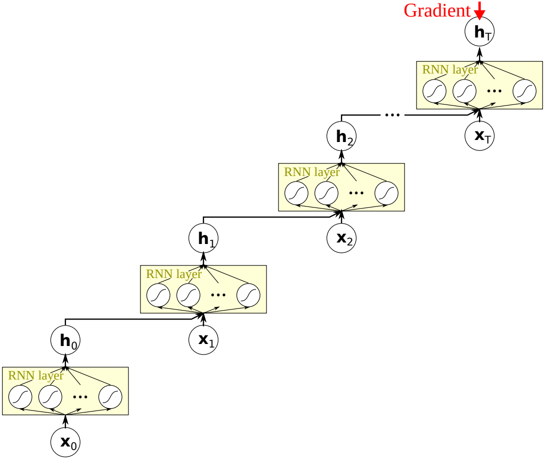
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

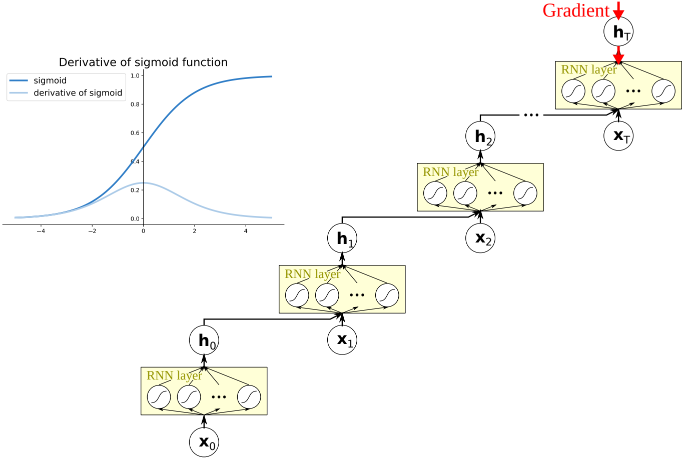
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

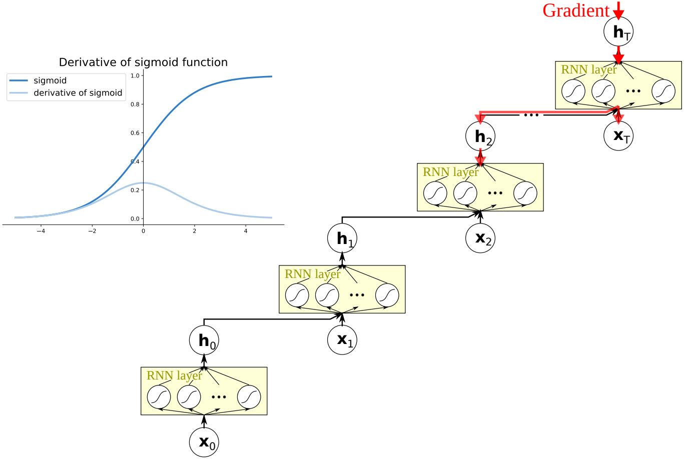
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

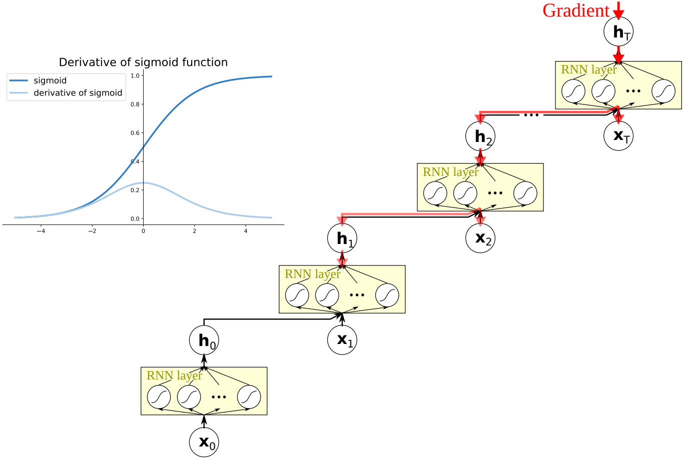
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

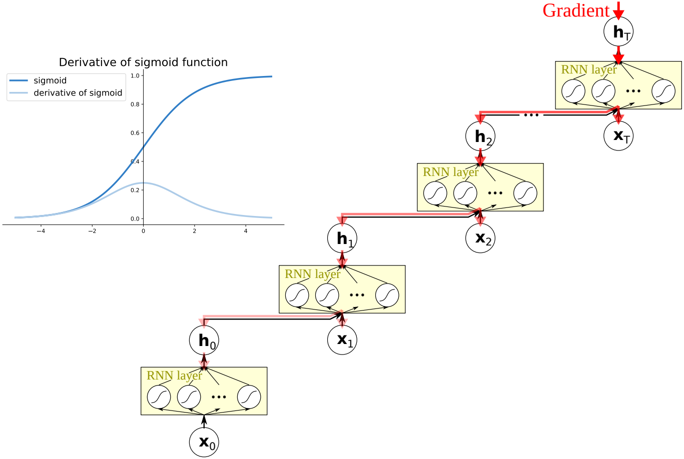
$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem

- BPTT generates very deep networks ($T \cong$ depth)

$\rightarrow$ Vanishing or Exploding Gradients (Hochreiter, 1991)



Vanishing Gradients Problem - Consequences

- RNNs tend to forget events that happened a long time ago
- Learning long-term dependencies depends on the recurrent weights
  - If $|f'| < 1$, we will forget things over time
  - If $|f'| > 1$, our system is unstable

  $\rightarrow$ we would need $|f'| = 1$

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store

Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
     - add up information over time
     - store information indefinitely
     - identity function, no vanishing gradients

Long Short-Term Memory (LSTM)

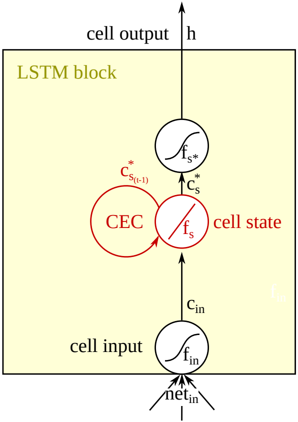
- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
     - add up information over time
     - store information indefinitely
     - identity function, no vanishing gradients
  2. Gates
     - hidden units
     - activations multiplied with input
     - write-access (remembering)
     - read-access (communicating)
     - reset (forgetting)

Long Short-Term Memory (LSTM)

- **Idea:** Store information indefinitely but be selective about what to store
- **Solution:**
  1. Integrator
     - add up information over time
     - store information indefinitely
     - identity function, no vanishing gradients
  2. Gates
     - hidden units
     - activations multiplied with input
     - write-access (remembering)
     - read-access (communicating)
     - reset (forgetting)
- This system is called Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997)

Long Short-Term Memory (LSTM)

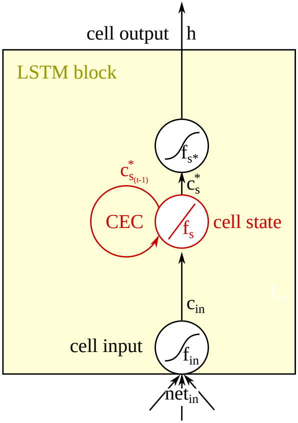
Constant Error Carousel (CEC)



Long Short-Term Memory (LSTM)

Constant Error Carousel (CEC)

- New information net_{in} is squashed to scalar c_{in} via function f_{in}

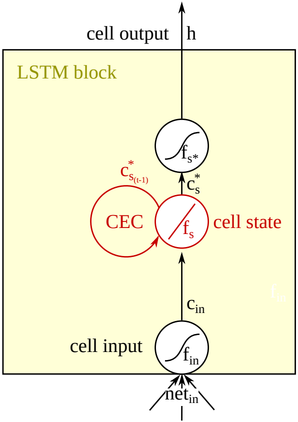


Long Short-Term Memory (LSTM)

Constant Error Carousel (CEC)

- New information net_{in} is squashed to scalar c_{in} via function f_{in}
- New cell state: $c^*_{s_t} = c^*_{s_{(t-1)}} + c_{in_t}$

$\Rightarrow$ simple integrator, no vanishing gradients!



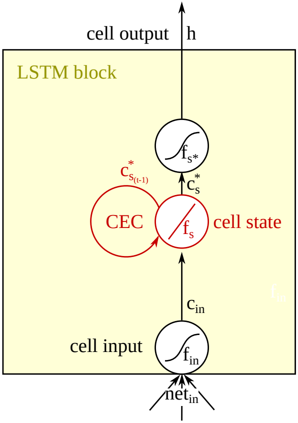
Long Short-Term Memory (LSTM)

Constant Error Carousel (CEC)

- New information net_{in} is squashed to scalar c_{in} via function f_{in}
- New cell state: $c^*_{s_t} = c^*_{s_{(t-1)}} + c_{in_t}$

$\Rightarrow$ simple integrator, no vanishing gradients!

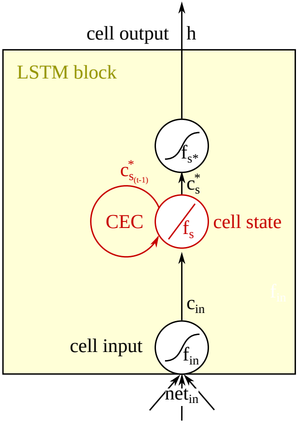
- f_{in}: e.g. $\tanh$
- f_{s^*}: e.g. $\tanh$, $linear$



Long Short-Term Memory (LSTM)

More terminology

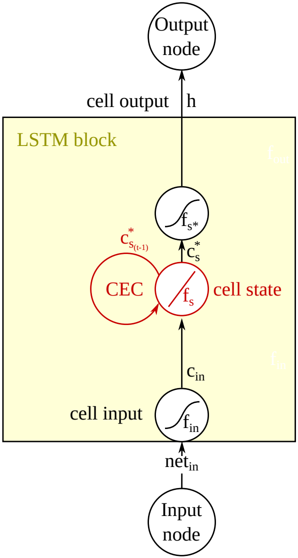
- CEC and gates constitute LSTM block or LSTM unit
- Cell output h (hidden state) is output of LSTM block
- Multiple LSTM blocks in one layer are referred to as LSTM layer



Long Short-Term Memory (LSTM)

More terminology

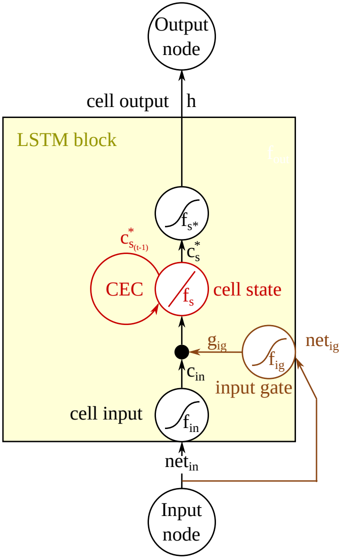
- CEC and gates constitute LSTM block or LSTM unit
- Cell output h (hidden state) is output of LSTM block
- Multiple LSTM blocks in one layer are referred to as LSTM layer



Long Short-Term Memory (LSTM)

Input gate

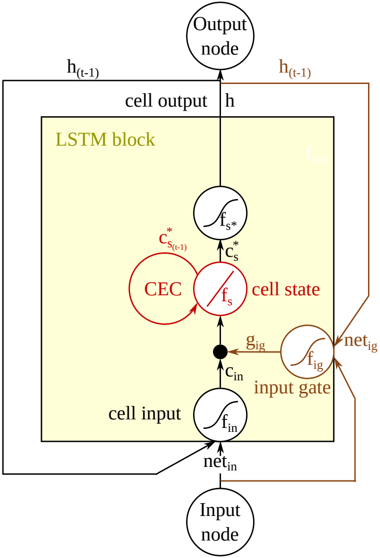
- Input gate serves as gating/attention mechanism
- c_{in} is multiplied by input gate activation g_{ig} before entering CEC
- f_{ig}: e.g. $sigmoid$



Long Short-Term Memory (LSTM)

Recurrent hidden state

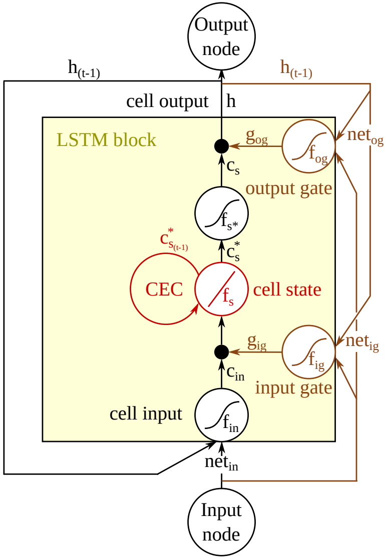
- Input gate and cell input may receive old hidden state $h_{(t-1)}$ as recurrent input
- In an LSTM layer, the hidden states of all LSTM blocks are the recurrent input per block
- But: fully connected LSTM might not always be the best way to go!



Long Short-Term Memory (LSTM)

Output gate

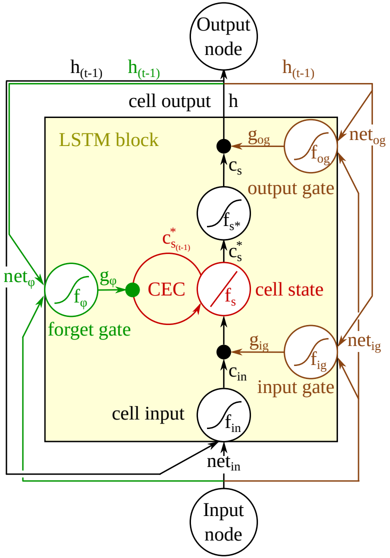
- Output gate mechanism analogous to input gate
- Output gate controls if cell state c_s is visible to rest of network
- f_{og}: e.g. $sigmoid$



Long Short-Term Memory (LSTM)

Forget gate

- Forget gate mechanism analogous to other gates
- Can reset or decrease CEC content
- f_φ: e.g. $sigmoid$

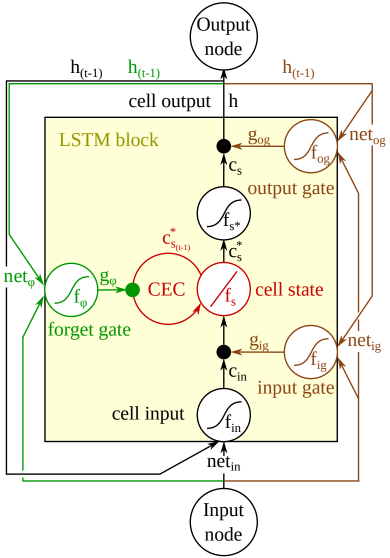


Long Short-Term Memory (LSTM)

Forget gate

- Forget gate mechanism analogous to other gates
- Can reset or decrease CEC content
- f_φ: e.g. $sigmoid$

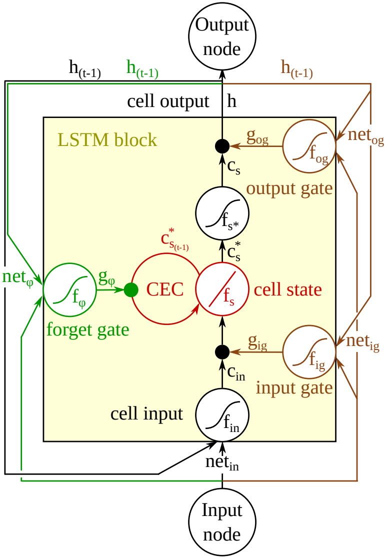
⇒ Problem: this re-introduces vanishing gradients! Only use if necessary!



Long Short-Term Memory (LSTM)

Learning behavior

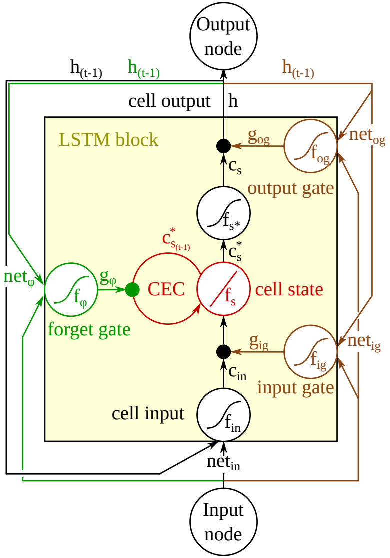
- LSTM core (CEC) is an integrator
- Gates introduce complex dynamics
- LSTM blocks (de)activate and complement each other dynamically



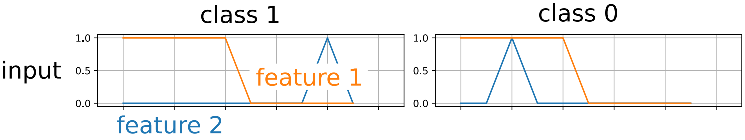
Long Short-Term Memory (LSTM)

Tricks of the trade

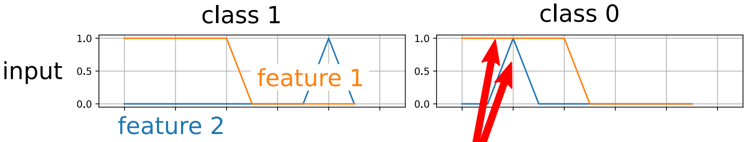
- Plot your LSTM cell- and hidden states & start small
- Fully connected LSTM not always needed
- Negative input gate bias helps for long sequences
- Use forget gate only if necessary



LSTM example: Task description

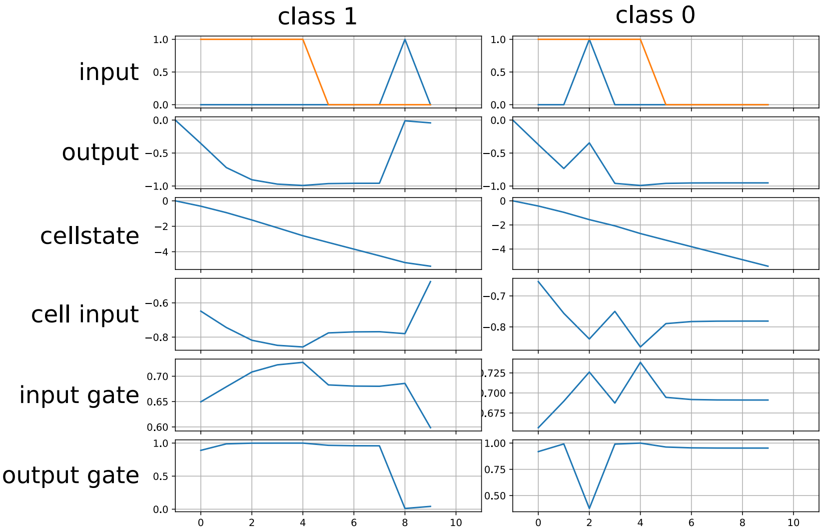


LSTM example: Task description



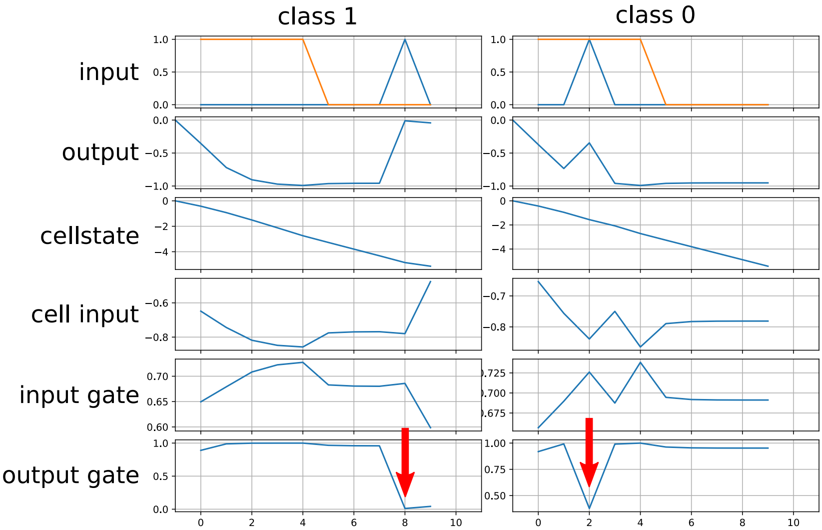
class 0: feature 1 and feature 2 active at same time

LSTM example: 1 LSTM (fully connected)



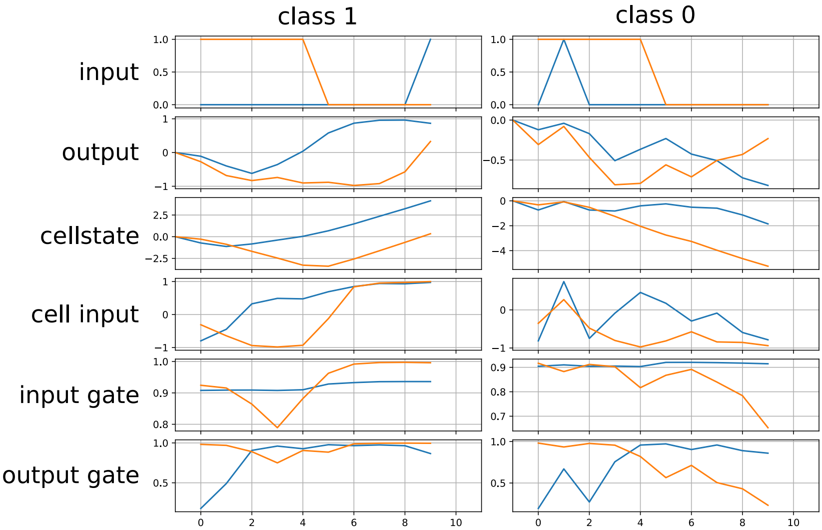
Created with: `https://github.com/widmi/widis-lstm-tools`

LSTM example: 1 LSTM (fully connected)



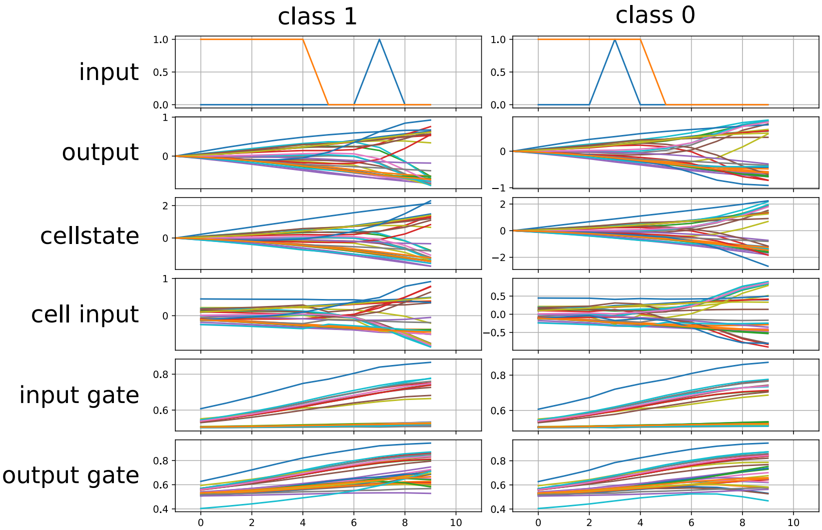
Created with: `https://github.com/widmi/widis-lstm-tools`

LSTM example: 2 LSTM (fully connected)



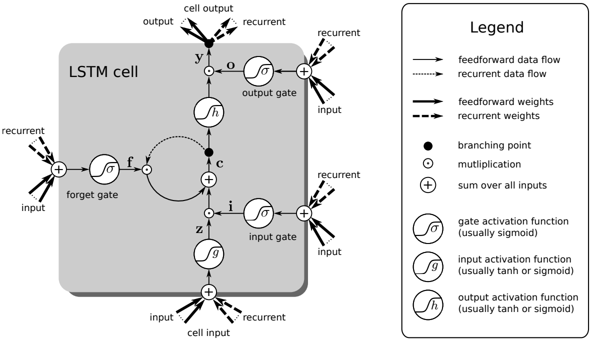
Created with: `https://github.com/widmi/widis-lstm-tools`

LSTM example: 32 LSTM (fully connected)



Created with: `https://github.com/widmi/widis-lstm-tools`

LSTM Formulas



$$\mathbf{z}^t = g\left(\boldsymbol{W}_\mathbf{z}\,\mathbf{x}^t + \boldsymbol{R}_\mathbf{z}\,\mathbf{y}^{t-1} + \boldsymbol{b}_\mathbf{z}\right) \quad \text{cell input}$$

$$\mathbf{i}^t = \sigma\left(\boldsymbol{W}_\mathbf{i}\,\mathbf{x}^t + \boldsymbol{R}_\mathbf{i}\,\mathbf{y}^{t-1} + \boldsymbol{b}_\mathbf{i}\right) \quad \text{input gate}$$

$$\mathbf{f}^t = \sigma\left(\boldsymbol{W}_\mathbf{f}\,\mathbf{x}^t + \boldsymbol{R}_\mathbf{f}\,\mathbf{y}^{t-1} + \boldsymbol{b}_\mathbf{f}\right) \quad \text{forget gate}$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \quad \text{cell state}$$

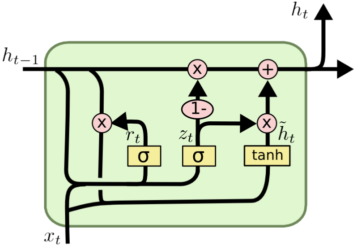
$$\mathbf{o}^t = \sigma\left(\boldsymbol{W}_\mathbf{o}\,\mathbf{x}^t + \boldsymbol{R}_\mathbf{o}\,\mathbf{y}^{t-1} + \boldsymbol{b}_\mathbf{o}\right) \quad \text{output gate}$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h\left(\mathbf{c}^t\right) \quad \text{cell output}$$

Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs)

- Reduced LSTM with merged gates (Cho et al, 2014)
- Suffers from Vanishing Gradients (always forgets)
- Less parameters, easier to use, lower complexity



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

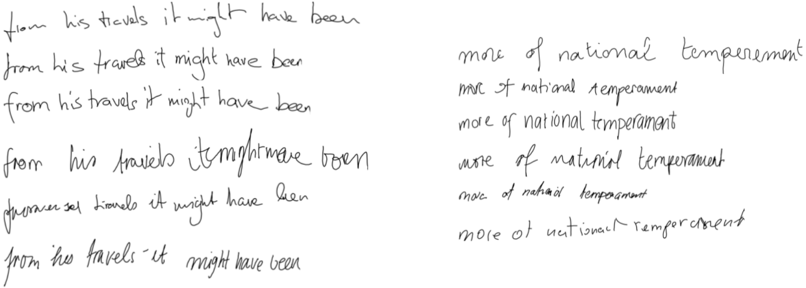
[Image source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`]

LSTM Applications

LSTM Applications

- LSTM can effectively learn long-term dependencies
- One of the most-used models today
- State of the Art in many applications
  - Speech/Text generation and recognition
  - Amino acid sequence classification
  - Time-Series classification/generation
  - ...

Handwriting Generation



[Generating Sequences With Recurrent Neural Networks, A. Graves, arxiv 2013]

Online interactive example: `https://www.cs.toronto.edu/~graves/handwriting.html`

Source Code Generation

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

[The Unreasonable Effectiveness of Recurrent Neural Networks, A. Karpathy, 2015]

Many more examples: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

Image Captioning

[Show and Tell: A Neural Image Caption Generator, Vinyals & Toshev & Bengio & Erhan, arxiv 2015]

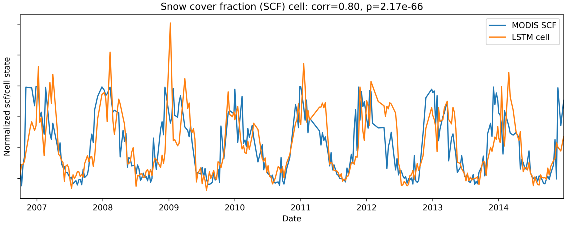
Language Translation

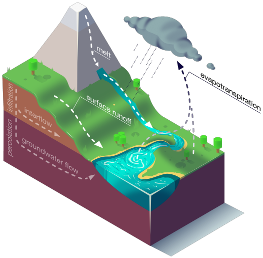
Type	Sentence
Our model	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
Truth	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
Our model	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
Truth	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
Our model	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
Truth	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

Table 3: A few examples of long translations produced by the LSTM alongside the ground truth translations. The reader can verify that the translations are sensible using Google translate.

[Sequence to Sequence Learning with Neural Networks, Sutskever & Vinyals & Le, NIPS 2014]

Hydrology Forecasts





[Sequence to Sequence Learning with Neural Networks, Kratzert & Herrneggerr & Klotz & Hochreiter & Klambauer]

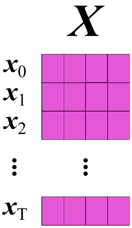
Transformers, Attention, and Modern Hopfield Networks

Differentiable Attention

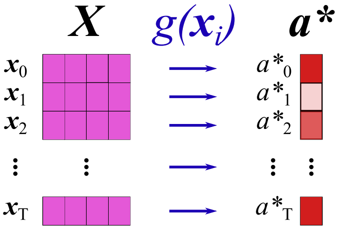
- Assume we have a set or bag of instances per sample $\mathbf{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_T\}$
- We can compute an attention weight a for each instance and combine the instances
  - Function g computes attention weight: $a_i^* = g(\mathbf{x}_i)$
  - Normalization e.g. via softmax: $\mathbf{a} = \text{softmax}\left(\mathbf{a}^*\right)$
  - Combination of instances, e.g. via weighted sum: $\mathbf{h} = \sum_0^T \left(a_i * \mathbf{x}_i\right)$

$\rightarrow$ We can attend to (=retrieve) specific instances in $\mathbf{X}$

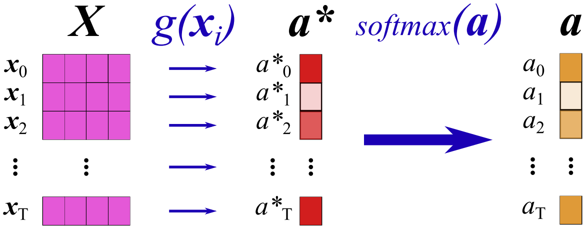
Differentiable Attention



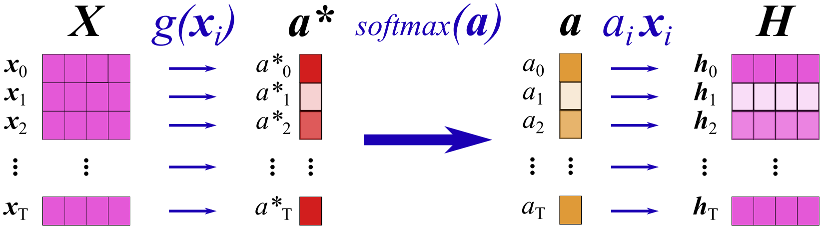
Differentiable Attention



Differentiable Attention



Differentiable Attention

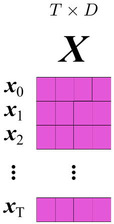


Transformer self-attention (1)

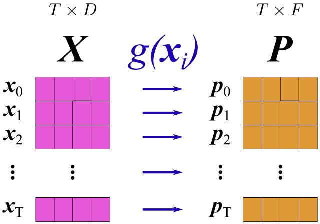
- Assume we want a_i^* to incorporate information about other instances in the bag

  …we could compute a outer product of our instance representations (combine everything with everything)

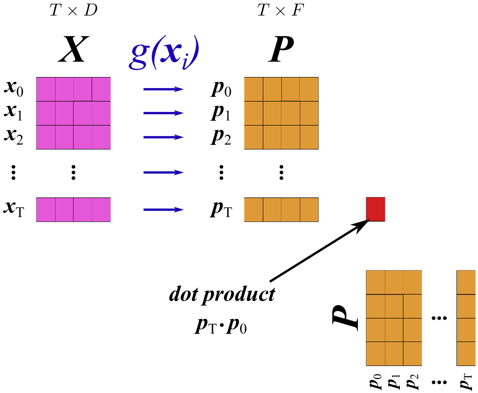
Transformer self-attention (1)



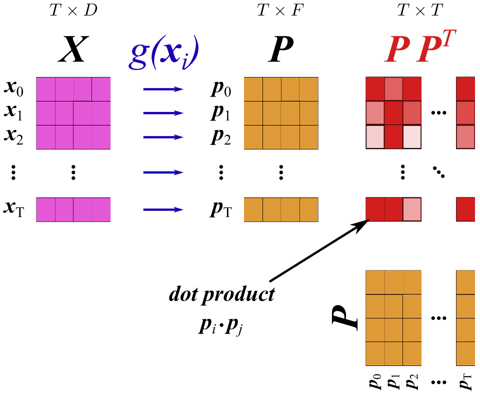
Transformer self-attention (1)



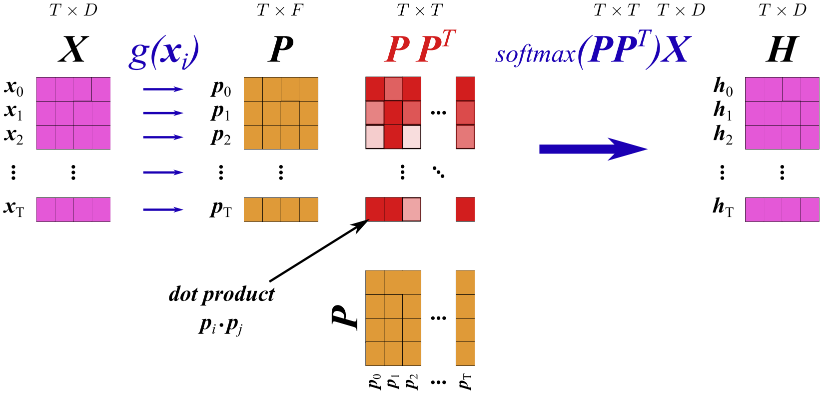
Transformer self-attention (1)



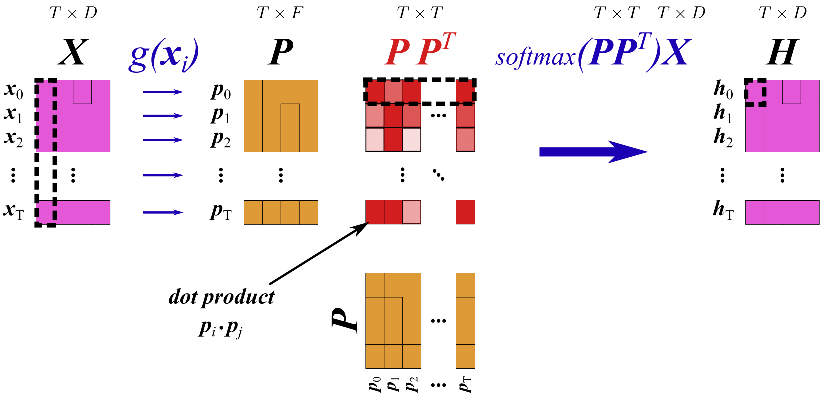
Transformer self-attention (1)



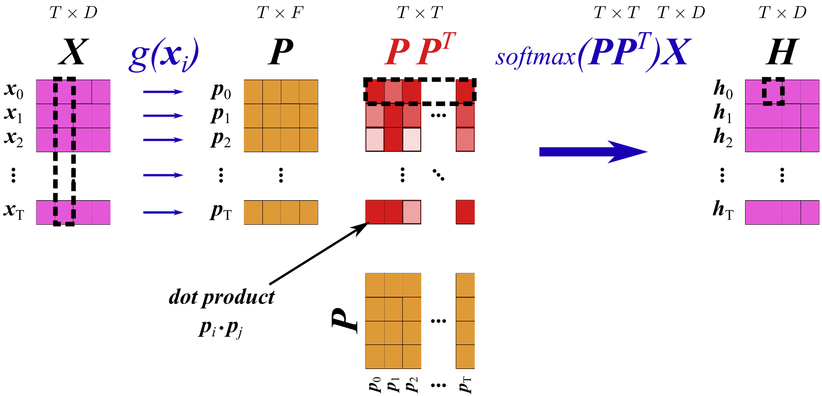
Transformer self-attention (1)



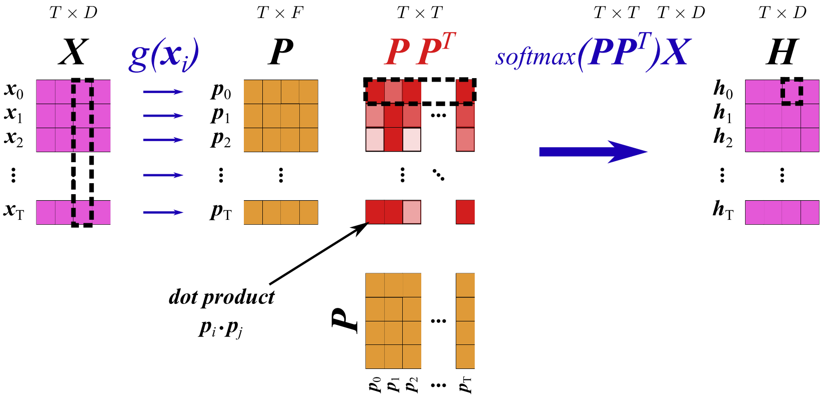
Transformer self-attention (1)



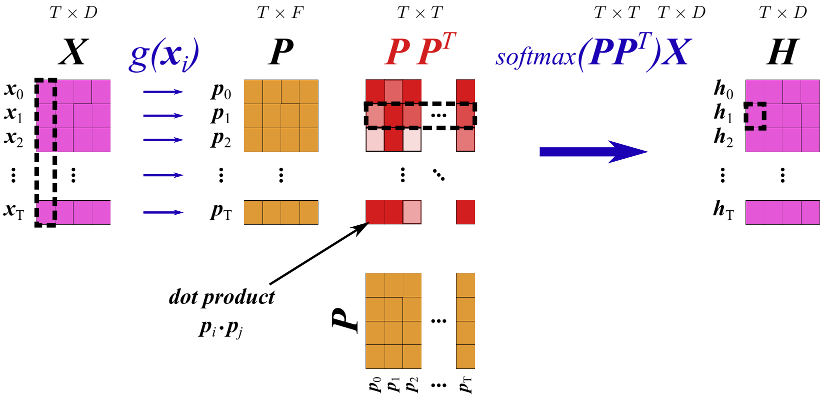
Transformer self-attention (1)



Transformer self-attention (1)



Transformer self-attention (1)



Transformer self-attention (2)

- Assume we want a_i^* to incorporate information about other instances in the bag

  ...we could compute a outer product of our instance representations...

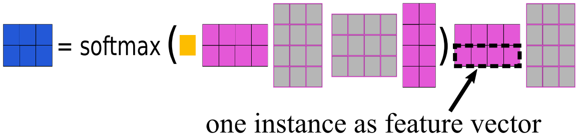
Transformer self-attention (2)

- Assume we want a_i^* to incorporate information about other instances in the bag

  ... we could compute a outer product of our instance representations... or better yet:

- Map our instances $\mathbf{X}$ to queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$ before the product! $\rightarrow$Transformer self-attention (Vaswani et al, 2017)

$$\boldsymbol{Z} = \text{softmax}\left(\beta \; \boldsymbol{X} \; \boldsymbol{W}_Q \; \boldsymbol{W}_K^T \; \boldsymbol{X}^T\right) \; \boldsymbol{X} \; \boldsymbol{W}_V$$



More details on Transformers: `https://www.youtube.com/watch?v=iDulhoQ2pro`

Modern Hopfield Networks (1)

- At closer inspection, such attention and memory mechanisms seem familiar?

Modern Hopfield Networks (1)

- At closer inspection, such attention and memory mechanisms seem familiar?

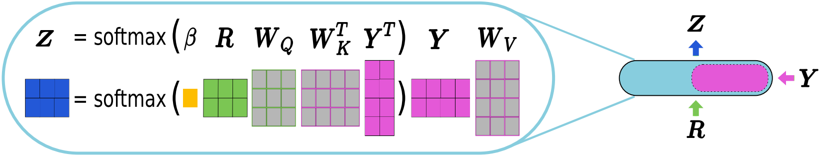
- Classic binary Hopfield networks:

  Sum of outer products is a simple associative memory! (Hebbian learning rule)

  1. Store patterns $\mathbf{X}$ in matrix $\mathbf{W}$
     $\mathbf{W} = \sum_i^N \mathbf{x}_i \mathbf{x}_i^T$ with $\mathbf{x} \in \{-1, 1\}^D$
  2. Retrieve pattern $\boldsymbol{\xi}^*$ based on query $\boldsymbol{\xi}$
     $\boldsymbol{\xi}^* = \text{sgn}(\boldsymbol{W}\boldsymbol{\xi}^t - \boldsymbol{b})$

Modern Hopfield Networks (2)

- Ramsauer et al (2020) generalized binary modern Hopfield networks to continuous modern Hopfield networks (MHN)
  - Differentiable and can be used as memory-equipped layers or pooling layers in NNs
  - Very large memory capacity (Widrich et al, 2020)
  - Special case of MHN: Transformer self-attention



More details on MHN: `https://ml-jku.github.io/hopfield-layers`, `https://www.youtube.com/watch?v=nv6oFDp6rNQ`

MHN: Tricks of the trade (1)

- MHN are order-invariant w.r.t. instances $\mathbf{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_T\}$
  - Instances might need position information
  - Might need to mask out future information in the sequence
- Associative memory of MHN can retrieve specific instances or meta-stable states
  - Interpolate between instances in high-dimensional space
- LSTM integrate information along sequence, MHN rather pinpoint sequence positions
  - LSTM or MHN better depending on task

MHN: Tricks of the trade (2)

- Attention matrix creates dynamic virtual weights
  - Relations between intances/features can be freely learned, e.g. learning to behave like CNN (Dosovitskiy et al, 2020)
  - Less assumptions on input relations BUT requires more data to learn the relations
  - Fast computation (parallel) if embedding allows for parallelization
  - Reduction of memory consumption exist, e.g. via Performers (Choromanski et al, 2021)
- MHN allow for different realizations
  - From very sample efficient (similar to SVM or KNN) to high complexity like Transformers

    `https://ml-jku.github.io/hopfield-layers`

Summary

Summary

- Recurrent Neural Networks (RNNs):
  - Can handle sequence data of variable length
  - Turing Complete but Vanishing Gradients problem

- Long Short-Term Memory (LSTM):
  - Integrator with gating mechanisms
  - Solves Vanishing Gradients problem

- Transformers, attention, modern Hopfield networks:
  - Sample is set/bag of instances (feature vectors)
  - Attention weight computed for each instance and used for weighted sum over instances
  - Requires information about instance position in sequences

Slides:

`https://github.com/widmi/aidd-school-2021-rnn-lstm-mhn`