LSTM Topics

Learning algorithms

BPTT
- RNN is unrolled in time -> standard Backpropagation can be applied
- Backpropagation through layers and time
- Updates resulting from different time steps are added up.
- Complexitiy: O(TI*(I + K + D))
- Activations of the whole sequence have to be stored
- BPTT is **local in space** -> complexity independent of number of weights

Truncated BPTT
- Truncated BPTT introduces a maximum number of timesteps tau
- Tau -> how far do I look back?
- Truncated BPTT only approximates the gradient
- Complexity: O(tau*(I + K + D))

RTRL
- RTRL computes all contributions to the gradients during the forward pass
- Activations **need not to be stored** over the whole sequence
- Complexitiy: At least O(i^4)
- RTRL is **local in time** -> complexity independent of sequence length
- Good when dealing with long sequences

Focused Backpropagation

Schmidhubers approach
- Has complexitiy O(i^3)

1. Divide the sequence in Blocks of length I (hidden)
2. Perform BPTT on each block: O(I^3)
3. After each block RTRL is performed to collect the gradients

- Only activations of sequences length I have to be stored

Vanishing and Exploding Gradient

Important measure is the Jacobian J = W(t) * diag(f'(s-1))

Vanishing Gradient
|| J || < 1: contradicting -> no matter where you start, you always end up at the same spot, no way to trace back where you came from
- RNNs with vanishing gradient only recognize the end of a sequence

Exploding Gradient
|| J || > 1: expanding
- RNNs with exploding gradient only recognize the beginning of a sequence

Volume conserving
|| J || = 1 -> this is true for LSTMs

LSTM
- Weights are long term memory
- Activations are short term memory

- LSTMs memory cell stores activations -> long short-term memory
- Central element of the memory cell are the self-recurrent units c, with R=I to avoid vanishing/ exploding gradients -> constant error carousel (CEC) -> traps error signals and distributes them equally among previous time steps.
- Standard LSTM is not able to forget and free memory. Problematic when sequences are very long —> Solution: forget gate f

---

Dropout for LSTM

LSTMs need a specialized version of dropout, because the probability for a unit to survive is (1 - p)^T (decays exponentially with sequence length T). The network cannot learn because the internal memory gets cleared.

Special dropout techniques for LSTM
- Apply dropout mask to input connections (x(t)) only
- *Zoneout* - drop the updates to the memory cell

Elements of dropout mask
- d(t) is a random vector of length I (hidden) whose entries are all (1 - p)-Bernoulli distributed.

Tricks of the Trade

Focused LSTM
LSTM without forget gate and only recurrent connections (y(t-1)) for the side gates (i, o) and only input connections (x(t)) for the input (z)

Lightweight LSTM
Like a focused LSTM, but without output gate (o)
- For sigmoid input activation the memory content is always growing and will more and more activate or deactivate the output
- Activation functions g and h are both tanh usually

Ticker steps
Adding elements repeatedly to the end of a sequence, for example copy of the last image.
-> Adds depth to the network
- Large number of layers enables heavy data abstraction -> Number of layers equal to sequence length
- To get a deeper net than defined by sequence length, additional elements are added -> ticker steps

Gate biases
- Problem: Drifting effect for long sequences, because of very large cell states over time
- Initialize input gates with negative bias: -1, -2, -3, …
- This gears the LSTMs initial behavior towards ignoring most of its inputs and the network should learn to open the input gates only for relevant information
- Initializing the input and output gate biases with a sequence leads to a ranking of the memory cells —> cells with smaller (abs value) biases will be used first

Scaling of g or h
- When there is only a few relevant sections in a long sequence
- scaling input activation g with a positive value leads to a clearly recognizable contribution to the memory state

- When small changes in the memory content should be recognizable
- Scale output activation function h

Linear g or h
- Using identity as an activation function for the input (g)
- Linear h (output) is important if some counting or accumulation signal is stored

Sigmoid vs. tanh for g
- Sigmoid: better for detecting single patterns that rarely appear in the input sequence
- Tanh: better for detecting hints for or against a situation (text analysis)

Absolute timing
- Giving the LSTM access to the time index
- Problem: feeding t as additional input only leads to accumulation in the cell state
- Solution: Feed sin(a*t) and cos(a*t), where a can be used to adjust the time scale (vgl. transformers)
- Abstract: a locality indicating function (e.g. Gaussian) is distributed across the sequence.

Fully connected gates
- Synaptic connections from i, o, z and y to all other units. —> Gates can activate themselves and other gates
- Learning can benefit from a shortcut, since one gate can learn another gates behavior

Separation of Memory and Compute
- Combination of RNN (processing unit) and LSTM (memory unit)
- Problem: training is difficult since RNN learns much faster than the memory cells —> Solution: inputs are only fed to RNN, memory cells operate on the hidden representations of the RNN

Input window
- A whole window of inputs is fed to the LSTM, instead of the actual input
- LSTM can learn patterns in the input via its cell input activation

Online learning
- Online learning is advantageous at beginning of learning, since the online update has an **exploratory effect**

More cells than necessary
- LSTM learning is sped up if more cells than necessary are being used.
- With more cells the chance increases that an input activation function has an **initialization close to** the **relevant pattern**

Learning rate schedule
- For LSTM networks the learning rate can be decreased over time.
- In some cases it is beneficial to increase the learning rate again to encourage exploration

Sequence classification
- It is better to process a whole sequence and at the end give the target (instead of continuously predicting the target)

Continuous prediction vs. Multiple LSTM networks
- Use multiple LSTMs instead of continuous prediction

Target and input scaling
Standardize input to zero mean and unit variance. If outliers are present, then apply tanh und standardize again. Do this until outliers are gone.

LSTM applications

Attention in sequence-to-sequence models

Differences to sequence-to-sequence models
- The encoder passes a lot more data to the decoder. Instead of passing the last hidden state the encoder passes all the hidden states to the decoder
-

Attention mechanism in general
1. Calculation of an attention score e(i)
2. The transformation of the score to an attention vector a(i) (softmax)
3. Calculation of the context vector c(i)

1. Score each hidden state
2. Softmax the scores
3. Multiply each (hidden state) vector by its softmaxed score
4. Sum up the weighted vectors —> Context vector for decoder

This is done at every decoder step (at every word)

Additive Attention (Bahdanau)
An **encoder-decoder model** that that combines attention with alignment.
Encoder: bidirectional RNN. Forward RNN produces forward hidden states h_f, backward RNN produces backward RNN h_b, which are combined to a general hidden state $h(j) = [h_f(j), h_b(j)]$
Decoder *RNN*: generates output y(i) based on hidden state s(i), where s(i) is a function of the previous hidden state s(i - 1), the output y(i - 1) and a context vector c(i).
Attention score e: alignment score of how well the inputs match the outputs around a certain position.
Score function is a feedforward network, where the hidden states h from the encoder and the hidden states s from the decoder are added. Both states have to be high to achieve an attention score at that position.
Application: Sentence translation

Multiplicative Attention (Luong)
An encoder-decoder model.
Focuses more on how things are related
Encoder: unidirectional stacked LSTM with hidden states h(t)
Decoder: hidden states s(i)
Attention score e: **Multiplication** of **encoder hidden states** times **decoder hidden states**
Multiplicative attention is simpler but performs worse than additive attention for longer sequences.
Attention vector a is softmax(e)

Local Attention
Multiplicative Attention, but not all hidden states of the decoder are used. Instead only hidden states within a certain interval around the output state position i are used.

Self-Attention
With self-attention an LSTM has direct access to its own past cell states or hidden states.

Sentence Embedding via Self-Attention

Key-Value Attention

Problem: for attention in sequence-to-sequence models the hidden states have to fulfill multiple roles:
1. Encode a distribution for predicting the next token
2. Serve as a key to compute the attention vector
3. Encode relevant content to inform future predictions

—> the key-value mechanism circumvents the problem of those multiple roles
Key functions as an address (where to look) and the Value represents what is looked at.

Transformer Networks
- Does not rely on RNNs, is only feed-forward.
- Transformer uses key-value attention, dot-product attention and self-attention
- Encoder-Decoder structure with Multi-Head attention
- Multi-Head Attention: Multiple attention algorithms in parallel

In every step in the hierarchy of the transformer a NxN matrix is enriched with additional information.

The „multi-headed“ attention expands to focus on different positions and allows for multiple „representation subspaces“

Attention score e: an alignment score that determines how well the inputs around position j and the output at position i match.

BERT - Bidirectional Encoder Representations from Transformers

Word embedding

Word embedding -> vector space that hosts the vocabulary of an entire language

Latent semantic analysis
Count-based method that relies on the assumption that words of similar meaning will occur in similar text corpora.

Neural word embedding
Based on a next-word-prediction task. Trained neural network represents a linear mapping from one-hot word vectors to embedding space, a smaller vector space.
The linear mapping can be used as a lookup-table for words in embedding space.

Continuous bag-of-words (CBOW)
Objective is to predict a word given its context; certain number of words before and after. The order does not matter, therefore bag of words.

Sequence-to-Sequence learning with LSTM

2 different variants

One-to-one correspondence between input and output sequence.
Input sequence and output sequence have the same length and are semantically aligned

Utilize a second LSTM network and split the translation task into two phases: encoding and decoding.
The encoder reads the input and learns a vector representation of the sentence.
This vector representation is copied to the decoder which generates a sequence in the target space.

LSTM variants

Peephole connections
Idea is to use the cell state c(t-1) as additional input into each gate, except the output gate.
+) helps to learn precise timing
-) can cause instabilities while learning longer sequences, where cell states can grow to large values

Bidirectional LSTM
Learning two models, each of which processes the sequence in a different direction. The output is computed depending on both networks.

Successfully applied to:
- Protein secondary structure prediction
- Part-of-speech tagging
- Text-to-speech synthesis
- Handwriting recognition
- Predicting functions of DNA sequences

Multidimensional LSTM
Extending LSTMs to more than the temporal dimension. Processing of images for example. Basic idea is to replace single recurrent connection in RNNs with as many recurrent connections as there are dimensions in the data.

PyraMiD LSTM
Multidimensional LSTMs are hard to parallelize on GPUs. PyraMiD LSTMs solve this issue.
An input is precessed in pyramidal fashion. The current piece of data depends on its three predecessors and so on.
The inputs are sampled from random locations, then fed to six C-LSTMs over three axes.
The c-LSTMs are combined and sent to two fully-connected layers.

Stacked LSTM
Stacking of multiple LSTM layers after another. The output of one LSTM is the input of the next LSTM. The information is always going in two directions.

Grid LSTM
Grid LSTM is a generalization of LSTM to operate in N processing dimensions at the same time.
Instead of operating on a time line, Grid LSTM operates on an N-dimensional grid.
E.g. image data can be viewed as a grid of pixels.

Convolutional LSTM
Videos are sequences of images over time.

Basic idea of convolutional LSTMs: the fully connected operations (weight matrix multiplication) are replaced by convolutions.

Conv-LSTM in encoder-decoder architecture.
An LSTM sits on one pixel and through the convolution it sees its local neighborhood. LSTM knows what it saw previously in the local neighborhood.

Gated Recurrent Unit

- Simplification of the LSTM. It **combines input gate and forget gate**. **Forget gate is 1 - input gate**. If the input gate wants to store something the forget gate deletes the content.
- It has fewer parameters than the fully featured LSTM and can sometimes be easier and faster to train
- Disadvantage: Can’t do tasks like simple counting
- GRUs are often used for problems where they yield a similar performance to LSTMs and a simpler structure is preferable
- More efficient