WIKIPEDIA

Recurrent neural network

A **recurrent neural network** (**RNN**) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition[1] or speech recognition.[2][3]

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior.[4] A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units.

Contents

History

Recurrent neural networks were based on David Rumelhart's work in 1986.[5] Hopfield networks were discovered by John Hopfield in 1982. In 1993, a neural history compressor system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.[6]

LSTM

Long short-term memory (LSTM) networks were discovered by Hochreiter and Schmidhuber in 1997 and set accuracy records in multiple applications domains.[7]

Around 2007, LSTM started to revolutionize speech recognition, outperforming traditional models in certain speech applications.[8] In 2009, a Connectionist Temporal Classification (CTC)-trained LSTM network was the first RNN to win pattern recognition contests when it won several competitions in connected handwriting recognition.[9][10] In 2014, the Chinese search giant Baidu used CTC-trained RNNs to break the Switchboard Hub5'00 speech recognition benchmark without using any traditional speech processing methods.[11]

LSTM also improved large-vocabulary speech recognition[2][3] and text-to-speech synthesis[12] and was used in Google Android.[9][13] In 2015, Google's speech recognition reportedly experienced a dramatic performance jump of 49% through CTC-trained LSTM, which was used by Google voice search.[14]

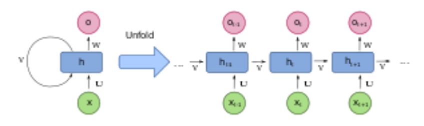
LSTM broke records for improved machine translation,[15] Language Modeling[16] and Multilingual Language Processing.[17] LSTM combined with convolutional neural networks (CNNs) improved automatic image captioning.[18]

Architectures

RNNs come in many variants.

Fully recurrent

Basic RNNs are a network of neuron-like nodes organized into successive "layers", each node in a given layer is connected with a directed (one-way) connection to every other node in the next successive layer. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside the network), output nodes (yielding results), or hidden nodes (that modify the data *en route* from input to output).



Unfolded basic recurrent neural network

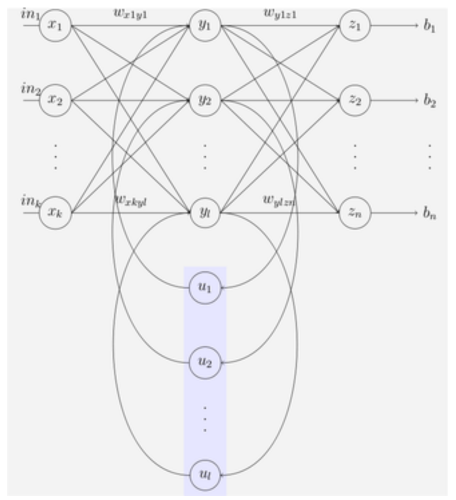
For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time step, each non-input unit computes its current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.

In reinforcement learning settings, no teacher provides target signals. Instead a fitness function or reward function is occasionally used to evaluate the RNN's performance, which influences its input stream through output units connected to actuators that affect the environment. This might be used to play a game in which progress is measured with the number of points won.

Each sequence produces an error as the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

Elman networks and Jordan networks

An Elman network is a three-layer network (arranged horizontally as *x*, *y*, and *z* in the illustration) with the addition of a set of "context units" (*u* in the illustration). The middle (hidden) layer is connected to these context units fixed with a weight of one.[19] At each time step, the input is fed-forward and a learning rule is applied. The fixed back-connections save a copy of the previous values of the hidden units in the context units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction that are beyond the power of a standard multilayer perceptron.



The Elman network

Jordan networks are similar to Elman networks. The context units are fed from the output layer instead of the hidden layer. The context units in a Jordan network are also referred to as the state layer. They have a recurrent connection to themselves.[19]

Elman and Jordan networks are also known as "simple recurrent networks" (SRN).

Elman network[20]

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Jordan network[21]

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Variables and functions

- x_t: input vector

- h_t: hidden layer vector
- y_t: output vector
- W, U and b: parameter matrices and vector
- σ_h and σ_y: Activation functions

Hopfield

The Hopfield network is an RNN in which all connections are symmetric. It requires stationary inputs and is thus not a general RNN, as it does not process sequences of patterns. It guarantees that it will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration.

Bidirectional associative memory

Introduced by Bart Kosko,[22] a bidirectional associative memory (BAM) network is a variant of a Hopfield network that stores associative data as a vector. The bi-directionality comes from passing information through a matrix and its transpose. Typically, bipolar encoding is preferred to binary encoding of the associative pairs. Recently, stochastic BAM models using Markov stepping were optimized for increased network stability and relevance to real-world applications.[23]

A BAM network has two layers, either of which can be driven as an input to recall an association and produce an output on the other layer.[24]

Echo state

The echo state network (ESN) has a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change (be trained). ESNs are good at reproducing certain time series.[25] A variant for spiking neurons is known as a liquid state machine.[26]

Independent RNN (IndRNN)

The Independently recurrent neural network (IndRNN)[27] addresses the gradient vanishing and exploding problems in the traditional fully connected RNN. Each neuron in one layer only receives its own past state as context information (instead of full connectivity to all other neurons in this layer) and thus neurons are independent of each other's history. The gradient backpropagation can be regulated to avoid gradient vanishing and exploding in order to keep long or short-term memory. The cross-neuron information is explored in the next layers. IndRNN can be robustly trained with the non-saturated nonlinear functions such as ReLU. Using skip connections, deep networks can be trained.

Recursive

A recursive neural network[28] is created by applying the same set of weights recursively over a differentiable graph-like structure by traversing the structure in topological order. Such networks are typically also trained by the reverse mode of automatic differentiation.[29][30] They can process distributed representations of structure, such as logical terms. A special case of recursive neural networks is the RNN whose structure corresponds to a linear chain. Recursive neural networks have been applied to natural language processing.[31] The Recursive Neural Tensor Network uses a tensor-based composition function for all nodes in the tree.[32]

Neural history compressor

The neural history compressor is an unsupervised stack of RNNs.[33] At the input level, it learns to predict its next input from the previous inputs. Only unpredictable inputs of some RNN in the hierarchy become inputs to the next higher level RNN, which therefore recomputes its internal state only rarely. Each higher level RNN thus studies a compressed representation of the information in the RNN below. This is done such that the input sequence can be precisely reconstructed from the representation at the highest level.

The system effectively minimises the description length or the negative logarithm of the probability of the data.[34] Given a lot of learnable predictability in the incoming data sequence, the highest level RNN can use supervised learning to easily classify even deep sequences with long intervals between important events.

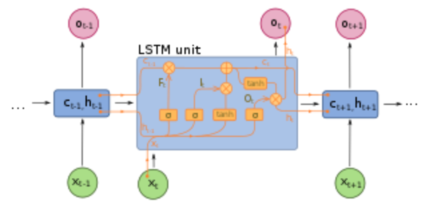
It is possible to distill the RNN hierarchy into two RNNs: the "conscious" chunker (higher level) and the "subconscious" automatizer (lower level).[33] Once the chunker has learned to predict and compress inputs that are unpredictable by the automatizer, then the automatizer can be forced in the next learning phase to predict or imitate through additional units the hidden units of the more slowly changing chunker. This makes it easy for the automatizer to learn appropriate, rarely changing memories across long intervals. In turn this helps the automatizer to make many of its once unpredictable inputs predictable, such that the chunker can focus on the remaining unpredictable events.[33]

A generative model partially overcame the vanishing gradient problem[35] of automatic differentiation or backpropagation in neural networks in 1992. In 1993, such a system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.[6]

Second order RNNs

Second order RNNs use higher order weights w_{ijk} instead of the standard w_{ij} weights, and states can be a product. This allows a direct mapping to a finite state machine both in training, stability, and representation.[36][37] Long short-term memory is an example of this but has no such formal mappings or proof of stability.

Long short-term memory



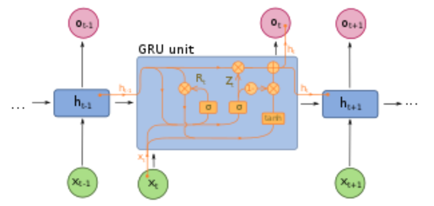
Long short-term memory unit

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget" gates.[38] LSTM prevents backpropagated errors from vanishing or exploding.[35] Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks[9] that require memories of events that happened thousands or even millions of discrete time steps earlier. Problem-specific LSTM-like topologies can be evolved.[39] LSTM works even given long delays between significant events and can handle signals that mix low and high frequency components.

Many applications use stacks of LSTM RNNs[40] and train them by Connectionist Temporal Classification (CTC)[41] to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences. CTC achieves both alignment and recognition.

LSTM can learn to recognize context-sensitive languages unlike previous models based on hidden Markov models (HMM) and similar concepts.[42]

Gated recurrent unit



Gated recurrent unit

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks introduced in 2014. They are used in the full form and several simplified variants.[43][44] Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory.[45] They have fewer parameters than LSTM, as they lack an output gate.[46]

Bi-directional

Bi-directional RNNs use a finite sequence to predict or label each element of the sequence based on the element's past and future contexts. This is done by concatenating the outputs of two RNNs, one processing the sequence from left to right, the other one from right to left. The combined outputs are the predictions of the teacher-given target signals. This technique proved to be especially useful when combined with LSTM RNNs.[47][48]

Continuous-time

A continuous time recurrent neural network (CTRNN) uses a system of ordinary differential equations to model the effects on a neuron of the incoming spike train.

For a neuron i in the network with action potential y_i, the rate of change of activation is given by:

$$\tau_i \dot{y}_i = -y_i + \sum_{j=1}^{n} w_{ji}\sigma(y_j - \Theta_j) + I_i(t)$$

Where:

- τ_i : Time constant of postsynaptic node
- y_i : Activation of postsynaptic node
- $\dot{y}_i$: Rate of change of activation of postsynaptic node
- w_{ji} : Weight of connection from pre to postsynaptic node
- $\sigma(x)$: Sigmoid of x e.g. $\sigma(x) = 1/(1+e^{-x})$.
- y_j : Activation of presynaptic node
- Θ_j : Bias of presynaptic node
- $I_i(t)$: Input (if any) to node

CTRNNs have been applied to evolutionary robotics where they have been used to address vision,[49] co-operation,[50] and minimal cognitive behaviour.[51]

Note that, by the Shannon sampling theorem, discrete time recurrent neural networks can be viewed as continuous-time recurrent neural networks where the differential equations have transformed into equivalent difference equations. This transformation can be thought of as occurring after the post-synaptic node activation functions $y_i(t)$ have been low-pass filtered but prior to sampling.

Hierarchical

Hierarchical RNNs connect their neurons in various ways to decompose hierarchical behavior into useful subprograms.[33][52]

Recurrent multilayer perceptron network

Generally, a Recurrent Multi-Layer Perceptron (RMLP) network consists of cascaded subnetworks, each of which contains multiple layers of nodes. Each of these subnetworks is feed-forward except for the last layer, which can have feedback connections. Each of these subnets is connected only by feed forward connections.[53]

Multiple timescales model

A multiple timescales recurrent neural network (MTRNN) is a neural-based computational model that can simulate the functional hierarchy of the brain through self-organization that depends on spatial connection between neurons and on distinct types of neuron activities, each with distinct time properties.[54][55] With such varied neuronal activities, continuous sequences of any set of behaviors are segmented into reusable primitives, which in turn are flexibly integrated into diverse sequential behaviors. The biological approval of such a type of hierarchy was discussed in the memory-prediction theory of brain function by Hawkins in his book *On Intelligence*.

Neural Turing machines

Neural Turing machines (NTMs) are a method of extending recurrent neural networks by coupling them to external memory resources which they can interact with by attentional processes. The combined system is analogous to a Turing machine or Von Neumann architecture but is differentiable end-to-end, allowing it to be efficiently trained with gradient descent.[56]

Differentiable neural computer

Differentiable neural computers (DNCs) are an extension of Neural Turing machines, allowing for usage of fuzzy amounts of each memory address and a record of chronology.

Neural network pushdown automata

Neural network pushdown automata (NNPDA) are similar to NTMs, but tapes are replaced by analogue stacks that are differentiable and that are trained. In this way, they are similar in complexity to recognizers of context free grammars (CFGs).[57]

Training

Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. In neural networks, it can be used to minimize the error term by changing each weight in proportion to the derivative of the error with respect to that weight, provided the non-linear activation functions are differentiable. Various methods for doing so were developed in the 1980s and early 1990s by Werbos, Williams, Robinson, Schmidhuber, Hochreiter, Pearlmutter and others.

The standard method is called "backpropagation through time" or BPTT, and is a generalization of back-propagation for feed-forward networks.[58][59] Like that method, it is an instance of automatic differentiation in the reverse accumulation mode of Pontryagin's minimum principle. A more computationally expensive online variant is called "Real-Time Recurrent Learning" or RTRL,[60][61] which is an instance of automatic differentiation in the forward accumulation mode with stacked tangent vectors. Unlike BPTT, this algorithm is local in time but not local in space.

In this context, local in space means that a unit's weight vector can be updated using only information stored in the connected units and the unit itself such that update complexity of a single unit is linear in the dimensionality of the weight vector. Local in time means that the updates take place continually (on-line) and depend only on the most recent time step rather than on multiple time steps within a given time horizon as in BPTT. Biological neural networks appear to be local with respect to both time and space.[62][63]

For recursively computing the partial derivatives, RTRL has a time-complexity of O(number of hidden x number of weights) per time step for computing the Jacobian matrices, while BPTT only takes O(number of weights) per time step, at the cost of storing all forward activations within the given time horizon.[64] An online hybrid between BPTT and RTRL with intermediate complexity exists,[65][66] along with variants for continuous time.[67]

A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events.[35][68] LSTM combined with a BPTT/RTRL hybrid learning method attempts to overcome these problems.[7] This problem is also solved in the independently recurrent neural network (IndRNN)[27] by reducing the context of a neuron to its own past state and the cross-neuron information can then be explored in the following layers. Memories of different range including long-term memory can be learned without the gradient vanishing and exploding problem.

The on-line algorithm called causal recursive backpropagation (CRBP), implements and combines BPTT and RTRL paradigms for locally recurrent networks.[69] It works with the most general locally recurrent networks. The CRBP algorithm can minimize the global error term. This fact improves stability of the algorithm, providing a unifying view on gradient calculation techniques for recurrent networks with local feedback.

One approach to the computation of gradient information in RNNs with arbitrary architectures is based on signal-flow graphs diagrammatic derivation.[70] It uses the BPTT batch algorithm, based on Lee's theorem for network sensitivity calculations.[71] It was proposed by Wan and Beaufays, while its fast online version was proposed by Campolucci, Uncini and Piazza.[71]

Global optimization methods

Training the weights in a neural network can be modeled as a non-linear global optimization problem. A target function can be formed to evaluate the fitness or error of a particular weight vector as follows: First, the weights in the network are set according to the weight vector. Next, the network is evaluated against the training sequence. Typically, the sum-squared-difference between the predictions and the target values specified in the training sequence is used to represent the error of the current weight vector. Arbitrary global optimization techniques may then be used to minimize this target function.

The most common global optimization method for training RNNs is genetic algorithms, especially in unstructured networks.[72][73][74]

Initially, the genetic algorithm is encoded with the neural network weights in a predefined manner where one gene in the chromosome represents one weight link.The whole network is represented as a single chromosome. The fitness function is evaluated as follows:

- Each weight encoded in the chromosome is assigned to the respective weight link of the network.
- The training set is presented to the network which propagates the input signals forward.
- The mean-squared-error is returned to the fitness function.
- This function drives the genetic selection process.

Many chromosomes make up the population; therefore, many different neural networks are evolved until a stopping criterion is satisfied. A common stopping scheme is:

- When the neural network has learnt a certain percentage of the training data or
- When the minimum value of the mean-squared-error is satisfied or
- When the maximum number of training generations has been reached.

The stopping criterion is evaluated by the fitness function as it gets the reciprocal of the mean-squared-error from each network during training. Therefore, the goal of the genetic algorithm is to maximize the fitness function, reducing the mean-squared-error.

Other global (and/or evolutionary) optimization techniques may be used to seek a good set of weights, such as simulated annealing or particle swarm optimization.

Related fields and models

RNNs may behave chaotically. In such cases, dynamical systems theory may be used for analysis.

They are in fact recursive neural networks with a particular structure: that of a linear chain. Whereas recursive neural networks operate on any hierarchical structure, combining child representations into parent representations, recurrent neural networks operate on the linear progression of time, combining the previous time step and a hidden representation into the representation for the current time step.

In particular, RNNs can appear as nonlinear versions of finite impulse response and infinite impulse response filters and also as a nonlinear autoregressive exogenous model (NARX).[75]

Libraries

- Apache Singa
- Caffe (http://caffe.berkeleyvision.org/): Created by the Berkeley Vision and Learning Center (BVLC). It supports both CPU and GPU. Developed in C++, and has Python and MATLAB wrappers.
- Chainer (https://docs.chainer.org/en/stable/): The first stable deep learning library that supports dynamic, define-by-run neural networks. Fully in Python, production support for CPU, GPU, distributed training.
- Deeplearning4j: Deep learning in Java and Scala on multi-GPU-enabled Spark. A general-purpose deep learning library (http://deeplearning4j.org/) for the JVM production stack running on a C++ scientific computing engine (https://github.com/deeplearning4j/libnd4j). Allows the creation of custom layers. Integrates with Hadoop and Kafka.
- Dynet (http://dynet.io/): The Dynamic Neural Networks toolkit.
- Keras: High-level, easy to use API, providing a wrapper to many other deep learning libraries.
- Microsoft Cognitive Toolkit
- MXNet: a modern open-source deep learning framework used to train and deploy deep neural networks.
- pytorch (https://pytorch.org/): Tensors and Dynamic neural networks in Python with strong GPU acceleration.
- TensorFlow: Apache 2.0-licensed Theano-like library with support for CPU, GPU and Google's proprietary TPU,[76] mobile
- Theano: The reference deep-learning library for Python with an API largely compatible with the popular NumPy library. Allows user to write symbolic mathematical expressions, then automatically generates their derivatives, saving the user from having to code gradients or backpropagation. These symbolic expressions are automatically compiled to CUDA code for a fast, on-the-GPU implementation.
- Torch (www.torch.ch (http://www.torch.ch/)): A scientific computing framework with wide support for machine learning algorithms, written in C and lua. The main author is Ronan Collobert, and it is now used at Facebook AI Research and Twitter.

Applications

Applications of Recurrent Neural Networks include:

- Machine Translation
- Robot control[77]
- Time series prediction[78]
- Speech recognition[79][80][81]
- Time series anomaly detection[82]
- Rhythm learning[83]
- Music composition[84]
- Grammar learning[85][86][87]
- Handwriting recognition[88][89]
- Human action recognition[90]
- Protein Homology Detection[91]
- Predicting subcellular localization of proteins[92]
- Several prediction tasks in the area of business process management[93]
- Prediction in medical care pathways[94]

References

1. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" (http://www.idsia.ch/~juergen/tpami_2008.pdf) (PDF). *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **31** (5). doi:10.1109/tpami.2008.137 (https://doi.org/10.1109%2Ftpami.2008.137).
2. Sak, Hasim; Senior, Andrew; Beaufays, Francoise (2014). "Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling" (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf) (PDF).
3. Li, Xiangang; Wu, Xihong (2014-10-15). "Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition". arXiv:1410.4281 (https://arxiv.org/abs/1410.4281) [cs.CL (https://arxiv.org/archive/cs.CL)].
4. Miljanovic, Milos (Feb–Mar 2012). "Comparative analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction" (http://www.ijcse.com/docs/INDJCSE12-03-01-028.pdf) (PDF). *Indian Journal of Computer and Engineering*. **3** (1).
5. Williams, Ronald J.; Hinton, Geoffrey E.; Rumelhart, David E. (1986-10). "Learning representations by back-propagating errors" (https://www.nature.com/articles/323533a0). *Nature*. **323** (6088): 533–536. doi:10.1038/323533a0 (https://doi.org/10.1038%2F323533a0). ISSN 1476-4687 (https://www.worldcat.org/issn/1476-4687). Check date values in: `|date=` (help)
6. Schmidhuber, Jürgen (1993). *Habilitation thesis: System modeling and optimization* (ftp://ftp.idsia.ch/pub/juergen/habilitation.pdf) (PDF). Page 150 ff demonstrates credit assignment across the equivalent of 1,200 layers in an unfolded RNN.
7. Hochreiter, Sepp; Schmidhuber, Jürgen (1997-11-01). "Long Short-Term Memory". *Neural Computation*. **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735 (https://doi.org/10.1162%2Fneco.1997.9.8.1735).
8. Fernández, Santiago; Graves, Alex; Schmidhuber, Jürgen (2007). "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting" (http://dl.acm.org/citation.cfm?id=1778066.1778092). *Proceedings of the 17th International Conference on Artificial Neural Networks*. ICANN'07. Berlin, Heidelberg: Springer-Verlag: 220–229. ISBN 978-3-540-74693-5.
9. Schmidhuber, Jürgen (January 2015). "Deep Learning in Neural Networks: An Overview". *Neural Networks*. **61**: 85–117. arXiv:1404.7828 (https://arxiv.org/abs/1404.7828). doi:10.1016/j.neunet.2014.09.003 (https://doi.org/10.1016%2Fj.neunet.2014.09.003). PMID 25462637 (https://www.ncbi.nlm.nih.gov/pubmed/25462637).
10. Graves, Alex; Schmidhuber, Jürgen (2009). Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris editor-K. I.; Culotta, Aron, eds. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks" (https://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks). *Neural Information Processing Systems (NIPS) Foundation*: 545–552.
11. Hannun, Awni; Case, Carl; Casper, Jared; Catanzaro, Bryan; Diamos, Greg; Elsen, Erich; Prenger, Ryan; Satheesh, Sanjeev; Sengupta, Shubho (2014-12-17). "Deep Speech: Scaling up end-to-end speech recognition". arXiv:1412.5567 (https://arxiv.org/abs/1412.5567) [cs.CL (https://arxiv.org/archive/cs.CL)].
12. Bo Fan, Lijuan Wang, Frank K. Soong, and Lei Xie (2015). Photo-Real Talking Head with Deep Bidirectional LSTM. In Proceedings of ICASSP 2015.
13. Zen, Heiga; Sak, Hasim (2015). "Unidirectional Long Short-Term Memory Recurrent Neural Network with Recurrent Output Layer for Low-Latency Speech Synthesis" (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43266.pdf) (PDF). *Google.com*. ICASSP. pp. 4470–4474.
14. Sak, Haşim; Senior, Andrew; Rao, Kanishka; Beaufays, Françoise; Schalkwyk, Johan (September 2015). "Google voice search: faster and more accurate" (http://googleresearch.blogspot.ch/2015/09/google-voice-search-faster-and-more.html).
15. Sutskever, L.; Vinyals, O.; Le, Q. (2014). "Sequence to Sequence Learning with Neural Networks" (https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf) (PDF). *Electronic Proceedings of the Neural Information Processing Systems Conference*. **27**: 5346. arXiv:1409.3215 (https://arxiv.org/abs/1409.3215). Bibcode:2014arXiv1409.3215S (http://adsabs.harvard.edu/abs/2014arXiv1409.3215S).
16. Jozefowicz, Rafal; Vinyals, Oriol; Schuster, Mike; Shazeer, Noam; Wu, Yonghui (2016-02-07). "Exploring the Limits of Language Modeling". arXiv:1602.02410 (https://arxiv.org/abs/1602.02410) [cs.CL (https://arxiv.org/archive/cs.CL)].
17. Gillick, Dan; Brunk, Cliff; Vinyals, Oriol; Subramanya, Amarnag (2015-11-30). "Multilingual Language Processing From Bytes". arXiv:1512.00103 (https://arxiv.org/abs/1512.00103) [cs.CL (https://arxiv.org/archive/cs.CL)].
18. Vinyals, Oriol; Toshev, Alexander; Bengio, Samy; Erhan, Dumitru (2014-11-17). "Show and Tell: A Neural Image Caption Generator". arXiv:1411.4555 (https://arxiv.org/abs/1411.4555).
19. Cruse, Holk; *Neural Networks as Cybernetic Systems* (http://www.brains-minds-media.org/archive/615/bmm615.pdf), 2nd and revised edition
20. Elman, Jeffrey L. (1990). "Finding Structure in Time". *Cognitive Science*. **14** (2): 179–211. doi:10.1016/0364-0213(90)90002-E (https://doi.org/10.1016%2F0364-0213%2890%2990002-E).
21. Jordan, Michael I. (1997-01-01). "Serial Order: A Parallel Distributed Processing Approach". *Advances in Psychology*. Neural-Network Models of Cognition. **121**: 471–495. doi:10.1016/s0166-4115(97)80111-2 (https://doi.org/10.1016%2Fs0166-4115%2897%2980111-2). ISBN 9780444819314.
22. Kosko, B. (1988). "Bidirectional associative memories". *IEEE Transactions on Systems, Man, and Cybernetics*. **18** (1): 49–60. doi:10.1109/21.87054 (https://doi.org/10.1109%2F21.87054).

23. Rakkiyappan, R.; Chandrasekar, A.; Lakshmanan, S.; Park, Ju H. (2 January 2015). "Exponential stability for markovian jumping stochastic BAM neural networks with mode-dependent probabilistic time-varying delays and impulse control". *Complexity*. **20** (3): 39–65. Bibcode:2015Cmplx..20c..39R (http://adsabs.harvard.edu/abs/2015Cmplx..20c..39R). doi:10.1002/cplx.21503 (https://doi.org/10.1002%2Fcplx.21503).
24. Rául Rojas (1996). *Neural networks: a systematic introduction* (https://books.google.com/books?id=txsjjYzFJS4C&pg=PA336). Springer. p. 336. ISBN 978-3-540-60505-8.
25. Jaeger, Herbert; Haas, Harald (2004-04-02). "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication". *Science*. **304** (5667): 78–80. Bibcode:2004Sci...304...78J (http://adsabs.harvard.edu/abs/2004Sci...304...78J). doi:10.1126/science.1091277 (https://doi.org/10.1126%2Fscience.1091277). PMID 15064413 (https://www.ncbi.nlm.nih.gov/pubmed/15064413).
26. W. Maass, T. Natschläger, and H. Markram. A fresh look at real-time computation in generic recurrent neural circuits. Technical report, Institute for Theoretical Computer Science, TU Graz, 2002.
27. Li, Shuai; Li, Wanqing; Cook, Chris; Zhu, Ce; Yanbo, Gao (2018). "Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN" (https://arxiv.org/pdf/1803.04831.pdf) (PDF). *IEEE Conference on Computer Vision and Pattern Recognition*.
28. Goller, C.; Küchler, A. (1996). "Learning task-dependent distributed representations by backpropagation through structure" (https://pdfs.semanticscholar.org/794e/6ed81d21f1bf32a0fd3be05c44c1fa362688.pdf) (PDF). *IEEE International Conference on Neural Networks, 1996*. **1**: 347. doi:10.1109/ICNN.1996.548916 (https://doi.org/10.1109%2FICNN.1996.548916). ISBN 0-7803-3210-5.
29. Seppo Linnainmaa (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, 6-7.
30. Griewank, Andreas; Walther, Andrea (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (https://books.google.com/books?id=xoiiLaRxcbEC) (Second ed.). SIAM. ISBN 978-0-89871-776-1.
31. Socher, Richard; Lin, Cliff; Ng, Andrew Y.; Manning, Christopher D., "Parsing Natural Scenes and Natural Language with Recursive Neural Networks" (http://ai.stanford.edu/~ang/papers/icml11-ParsingWithRecursiveNeuralNetworks.pdf) (PDF), *28th International Conference on Machine Learning (ICML 2011)*
32. Socher, Richard; Perelygin, Alex; Y. Wu, Jean; Chuang, Jason; D. Manning, Christopher; Y. Ng, Andrew; Potts, Christopher. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank" (http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) (PDF). *Emnlp 2013*.
33. Schmidhuber, Jürgen (1992). "Learning complex, extended sequences using the principle of history compression" (ftp://ftp.idsia.ch/pub/juergen/chunker.pdf) (PDF). *Neural Computation*. **4** (2): 234–242. doi:10.1162/neco.1992.4.2.234 (https://doi.org/10.1162%2Fneco.1992.4.2.234).
34. Schmidhuber, Jürgen (2015). "Deep Learning" (http://www.scholarpedia.org/article/Deep_Learning#Fundamental_Deep_Learning_Problem_and_Unsupervised_Pre-Training_of_RNNs_and_FNNs). *Scholarpedia*. **10** (11): 32832. Bibcode:2015SchpJ..1032832S (http://adsabs.harvard.edu/abs/2015SchpJ..1032832S). doi:10.4249/scholarpedia.32832 (https://doi.org/10.4249%2Fscholarpedia.32832).
35. Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen (http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf), Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.
36. C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, "Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks" (https://clgiles.ist.psu.edu/pubs/NC1992-recurrent-NN.pdf), Neural Computation, 4(3), p. 393, 1992.
37. C.W. Omlin, C.L. Giles, "Constructing Deterministic Finite-State Automata in Recurrent Neural Networks" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.2364&rep=rep1&type=pdf) Journal of the ACM, 45(6), 937-972, 1996.
38. Gers, Felix; Schraudolph, Nicol N.; Schmidhuber, Jürgen. "Learning Precise Timing with LSTM Recurrent Networks (PDF Download Available)" (https://www.researchgate.net/publication/220320057_Learning_Precise_Timing_with_LSTM_Recurrent_Networks). *ResearchGate*. pp. 115–143. Retrieved 2017-06-13.
39. Bayer, Justin; Wierstra, Daan; Togelius, Julian; Schmidhuber, Jürgen (2009-09-14). "Evolving Memory Cell Structures for Sequence Learning". *Artificial Neural Networks – ICANN 2009*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. **5769**: 755–764. doi:10.1007/978-3-642-04277-5_76 (https://doi.org/10.1007%2F978-3-642-04277-5_76). ISBN 978-3-642-04276-8.
40. Fernández, Santiago; Graves, Alex; Schmidhuber, Jürgen (2007). "Sequence labelling in structured domains with hierarchical recurrent neural networks". *Proc. 20th Int. Joint Conf. on Artificial In^{TEL}ligence, Ijcai 2007*: 774–779. CiteSeerX 10.1.1.79.1887 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.1887).
41. Graves, Alex; Fernández, Santiago; Gomez, Faustino (2006). "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks". *In Proceedings of the International Conference on Machine Learning, ICML 2006*: 369–376. CiteSeerX 10.1.1.75.6306 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.6306).
42. Gers, F. A.; Schmidhuber, E. (November 2001). "LSTM recurrent networks learn simple context-free and context-sensitive languages" (http://ieeexplore.ieee.org/document/963769/). *IEEE Transactions on Neural Networks*. **12** (6): 1333–1340. doi:10.1109/72.963769 (https://doi.org/10.1109%2F72.963769). ISSN 1045-9227 (https://www.worldcat.org/issn/1045-9227).
43. Heck, Joel; Salem, Fathi M. (2017-01-12). "Simplified Minimal Gated Unit Variations for Recurrent Neural Networks". arXiv:1701.03452 (https://arxiv.org/abs/1701.03452) [cs.NE (https://arxiv.org/archive/cs.NE)].
44. Dey, Rahul; Salem, Fathi M. (2017-01-20). "Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks". arXiv:1701.05923 (https://arxiv.org/abs/1701.05923) [cs.NE (https://arxiv.org/archive/cs.NE)].
45. Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555 (https://arxiv.org/abs/1412.3555) [cs.NE (https://arxiv.org/archive/cs.NE)].
46. "Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano – WildML" (http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/). *Wildml.com*. Retrieved May 18, 2016.
47. Graves, Alex; Schmidhuber, Jürgen (2005-07-01). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". *Neural Networks*. IJCNN 2005. **18** (5): 602–610. doi:10.1016/j.neunet.2005.06.042 (https://doi.org/10.1016%2Fj.neunet.2005.06.042).
48. Thireou, T.; Reczko, M. (July 2007). "Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins". *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. **4** (3): 441–446. doi:10.1109/tcbb.2007.1015 (https://doi.org/10.1109%2Ftcbb.2007.1015).
49. Harvey, Inman; Husbands, P.; Cliff, D. (1994), "Seeing the light: Artificial evolution, real vision" (https://www.researchgate.net/publication/229091538_Seeing_the_Light_Artificial_Evolution_Real_Vision), *3rd international conference on Simulation of adaptive behavior: from animals to animats 3*, pp. 392–401
50. Quinn, Matthew (2001). "Evolving communication without dedicated communication channels". *Advances in Artificial Life*. Lecture Notes in Computer Science. **2159**: 357–366. doi:10.1007/3-540-44811-X_38 (https://doi.org/10.1007%2F3-540-44811-X_38). ISBN 978-3-540-42567-0.
51. Beer, R.D. (1997). "The dynamics of adaptive behavior: A research program". *Robotics and Autonomous Systems*. **20** (2–4): 257–289. doi:10.1016/S0921-8890(96)00063-2 (https://doi.org/10.1016%2FS0921-8890%2896%2900063-2).
52. Paine, Rainer W.; Tani, Jun (2005-09-01). "How Hierarchical Control Self-organizes in Artificial Adaptive Systems". *Adaptive Behavior*. **13** (3): 211–225. doi:10.1177/105971230501300303 (https://doi.org/10.1177%2F105971230501300303).
53. "Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications". CiteSeerX 10.1.1.45.3527 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.3527). Missing or empty |url= (help)
54. Yamashita, Yuichi; Tani, Jun (2008-11-07). "Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment" (http://journals.plos.org/ploscompbiol/article/file?id=10.1371/journal.pcbi.1000220&type=printable). *PLOS Computational Biology*. **4** (11): e1000220. Bibcode:2008PLSCB...4E0220Y (http://adsabs.harvard.edu/abs/2008PLSCB...4E0220Y). doi:10.1371/journal.pcbi.1000220 (https://doi.org/10.1371%2Fjournal.pcbi.1000220). PMC 2570613 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2570613). PMID 18989398 (https://www.ncbi.nlm.nih.gov/pubmed/18989398).
55. Shibata Alnajjar, Fady; Yamashita, Yuichi; Tani, Jun (2013). "The hierarchical and functional connectivity of higher-order cognitive mechanisms: neurorobotic model to investigate the stability and flexibility of working memory" (http://journal.frontiersin.org/article/10.3389/fnbot.2013.00002/pdf). *Frontiers in Neurorobotics*. **7**: 2. doi:10.3389/fnbot.2013.00002 (https://doi.org/10.3389%2Ffnbot.2013.00002). PMC 3575058 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3575058). PMID 23423881 (https://www.ncbi.nlm.nih.gov/pubmed/23423881).
56. Graves, Alex; Wayne, Greg; Danihelka, Ivo (2014). "Neural Turing Machines". arXiv:1410.5401 (https://arxiv.org/abs/1410.5401) [cs.NE (https://arxiv.org/archive/cs.NE)].

57. Sun, Guo-Zheng; Giles, C. Lee; Chen, Hsing-Hen (1998). "The Neural Network Pushdown Automaton: Architecture, Dynamics and Training". In Giles, C. Lee; Gori, Marco. *Adaptive Processing of Sequences and Data Structures*. Lecture Notes in Computer Science. Springer Berlin Heidelberg. pp. 296–345. doi:10.1007/bfb0054003 (https://doi.org/10.1007%2Fbfb0054003). ISBN 9783540643418.
58. Werbos, Paul J. (1988). "Generalization of backpropagation with application to a recurrent gas market model" (http://linkinghub.elsevier.com/retrieve/pii/089360808890007X). *Neural Networks*. **1** (4): 339–356. doi:10.1016/0893-6080(88)90007-x (https://doi.org/10.1016%2F0893-6080%2888%2990007-x).
59. Rumelhart, David E. (1985). *Learning Internal Representations by Error Propagation* (https://books.google.com/books?id=Ff9iHAAACAAJ). Institute for Cognitive Science, University of California, San Diego.
60. Robinson, A. J. (1987). *The Utility Driven Dynamic Error Propagation Network. Technical Report CUED/F-INFENG/TR.1* (https://books.google.com/books?id=6JYYMwEACAAJ). University of Cambridge Department of Engineering.
61. Williams, R. J.; Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity, D. (1 February 2013). Chauvin, Yves; Rumelhart, David E., eds. *Backpropagation: Theory, Architectures, and Applications* (https://books.google.com/books?id=B71nu3LDpREC). Psychology Press. ISBN 978-1-134-77581-1.
62. SCHMIDHUBER, JURGEN (1989-01-01). "A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks". *Connection Science*. **1** (4): 403–412. doi:10.1080/09540098908915650 (https://doi.org/10.1080%2F09540098908915650).
63. Príncipe, José C.; Euliano, Neil R.; Lefebvre, W. Curt (2000). *Neural and adaptive systems: fundamentals through simulations* (https://books.google.com/books?id=jgMZAQAAIAAJ). Wiley. ISBN 978-0-471-35167-2.
64. Yann, Ollivier; Corentin, Tallec; Guillaume, Charpiat (2015-07-28). "Training recurrent networks online without backtracking". arXiv:1507.07680 (https://arxiv.org/abs/1507.07680) [cs.NE (https://arxiv.org/archive/cs.NE)].
65. Schmidhuber, Jürgen (1992-03-01). "A Fixed Size Storage O(n3) Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks". *Neural Computation*. **4** (2): 243–248. doi:10.1162/neco.1992.4.2.243 (https://doi.org/10.1162%2Fneco.1992.4.2.243).
66. Williams, R. J. (1989). "Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27" (http://citeseerx.ist.psu.edu/showciting?cid=128036). Boston: Northeastern University, College of Computer Science.
67. Pearlmutter, Barak A. (1989-06-01). "Learning State Space Trajectories in Recurrent Neural Networks". *Neural Computation*. **1** (2): 263–269. doi:10.1162/neco.1989.1.2.263 (https://doi.org/10.1162%2Fneco.1989.1.2.263).
68. Hochreiter, S.; et al. (15 January 2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In Kolen, John F.; Kremer, Stefan C. *A Field Guide to Dynamical Recurrent Networks* (https://books.google.com/books?id=NWOcMVA64aAC). John Wiley & Sons. ISBN 978-0-7803-5369-5.
69. Campolucci; Uncini, A.; Piazza, F.; Rao, B. D. (1999). "On-Line Learning Algorithms for Locally Recurrent Neural Networks". *IEEE Transactions on Neural Networks*. **10** (2): 253–271. doi:10.1109/72.750549 (https://doi.org/10.1109%2F72.750549).
70. Wan, E. A.; Beaufays, F. (1996). "Diagrammatic derivation of gradient algorithms for neural networks". *Neural Computation*. **8**: 182–201. doi:10.1162/neco.1996.8.1.182 (https://doi.org/10.1162%2Fneco.1996.8.1.182).
71. Campolucci, P.; Uncini, A.; Piazza, F. (2000). "A Signal-Flow-Graph Approach to On-line Gradient Calculation". *Neural Computation*. **12** (8): 1901–1927. doi:10.1162/089976600300015196 (https://doi.org/10.1162%2F089976600300015196).
72. Gomez, F. J.; Miikkulainen, R. (1999), "Solving non-Markovian control tasks with neuroevolution" (http://www.cs.utexas.edu/users/nn/downloads/papers/gomez.ijcai99.pdf) (PDF), *IJCAI 99*, Morgan Kaufmann, retrieved 5 August 2017
73. "Applying Genetic Algorithms to Recurrent Neural Networks for Learning Network Parameters and Architecture" (http://arimaa.com/arimaa/about/Thesis/).
74. Gomez, Faustino; Schmidhuber, Jürgen; Miikkulainen, Risto (June 2008). "Accelerated Neural Evolution Through Cooperatively Coevolved Synapses" (http://dl.acm.org/citation.cfm?id=1390681.1390712). *J. Mach. Learn. Res.* **9**: 937–965.
75. Siegelmann, Hava T.; Horne, Bill G.; Giles, C. Lee (1995). *Computational Capabilities of Recurrent NARX Neural Networks* (https://books.google.com/books?id=830-HAAACAAJ&pg=PA208). University of Maryland. pp. 208–215.
76. Cade Metz (May 18, 2016). "Google Built Its Very Own Chips to Power Its AI Bots" (https://www.wired.com/2016/05/google-tpu-custom-chips/). *Wired*.
77. Mayer, H.; Gomez, F.; Wierstra, D.; Nagy, I.; Knoll, A.; Schmidhuber, J. (October 2006). "A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks" (http://ieeexplore.ieee.org/document/4059310/). *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*: 543–548. doi:10.1109/IROS.2006.282190 (https://doi.org/10.1109%2FIROS.2006.282190). ISBN 1-4244-0258-1.
78. Wierstra, Daan; Schmidhuber, J.; Gomez, F. J. (2005). "Evolino: Hybrid Neuroevolution/Optimal Linear Search for Sequence Learning" (https://www.academia.edu/5830256/Evolino_Hybrid_Neuroevolution_Optimal_Linear_Search_for_Sequence_Learning). *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh*: 853–858.
79. Graves, A.; Schmidhuber, J. (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". *Neural Networks*. **18** (5–6): 602–610. doi:10.1016/j.neunet.2005.06.042 (https://doi.org/10.1016%2Fj.neunet.2005.06.042).
80. Fernández, Santiago; Graves, Alex; Schmidhuber, Jürgen (2007). "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting" (http://dl.acm.org/citation.cfm?id=1778066.1778092). *Proceedings of the 17th International Conference on Artificial Neural Networks*. ICANN'07. Berlin, Heidelberg: Springer-Verlag: 220–229. ISBN 3540746935.
81. Graves, Alex; Mohamed, Abdel-rahman; Hinton, Geoffrey (2013). "Speech Recognition with Deep Recurrent Neural Networks". *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*: 6645–6649.
82. Malhotra, Pankaj; Vig, Lovekesh; Shroff, Gautam; Agarwal, Puneet (April 2015). "Long Short Term Memory Networks for Anomaly Detection in Time Series" (https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf) (PDF). *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning — ESANN 2015*.
83. Gers, F.; Schraudolph, N.; Schmidhuber, J. (2002). "Learning precise timing with LSTM recurrent networks" (http://www.jmlr.org/papers/volume3/gers02a/gers02a.pdf) (PDF). *Journal of Machine Learning Research*. **3**: 115–143.
84. Eck, Douglas; Schmidhuber, Jürgen (2002-08-28). "Learning the Long-Term Structure of the Blues" (https://link.springer.com/chapter/10.1007/3-540-46084-5_47). *Artificial Neural Networks — ICANN 2002*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. **2415**: 284–289. doi:10.1007/3-540-46084-5_47 (https://doi.org/10.1007%2F3-540-46084-5_47). ISBN 3540460845.
85. Schmidhuber, J.; Gers, F.; Eck, D.; Schmidhuber, J.; Gers, F. (2002). "Learning nonregular languages: A comparison of simple recurrent networks and LSTM". *Neural Computation*. **14** (9): 2039–2041. doi:10.1162/089976602320263980 (https://doi.org/10.1162%2F089976602320263980).
86. Gers, F. A.; Schmidhuber, J. (2001). "LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages" (ftp://ftp.idsia.ch/pub/juergen/L-IEEE.pdf) (PDF). *IEEE Transactions on Neural Networks*. **12** (6): 1333–1340. doi:10.1109/72.963769 (https://doi.org/10.1109%2F72.963769).
87. Perez-Ortiz, J. A.; Gers, F. A.; Eck, D.; Schmidhuber, J. (2003). "Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets". *Neural Networks*. **16** (2): 241–250. doi:10.1016/s0893-6080(02)00219-8 (https://doi.org/10.1016%2Fs0893-6080%2802%2900219-8).
88. A. Graves, J. Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. Advances in Neural Information Processing Systems 22, NIPS'22, pp 545–552, Vancouver, MIT Press, 2009.
89. Graves, Alex; Fernández, Santiago; Liwicki, Marcus; Bunke, Horst; Schmidhuber, Jürgen (2007). "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks" (http://dl.acm.org/citation.cfm?id=2981562.2981635). *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. USA: Curran Associates Inc.: 577–584. ISBN 9781605603520.
90. M. Baccouche, F. Mamalet, C Wolf, C. Garcia, A. Baskurt. Sequential Deep Learning for Human Action Recognition. 2nd International Workshop on Human Behavior Understanding (HBU), A.A. Salah, B. Lepri ed. Amsterdam, Netherlands. pp. 29–39. Lecture Notes in Computer Science 7065. Springer. 2011
91. Hochreiter, S.; Heusel, M.; Obermayer, K. (2007). "Fast model-based protein homology detection without alignment". *Bioinformatics*. **23** (14): 1728–1736. doi:10.1093/bioinformatics/btm247 (https://doi.org/10.1093%2Fbioinformatics%2Fbtm247). PMID 17488755 (https://www.ncbi.nlm.nih.gov/pubmed/17488755).
92. Thireou, T.; Reczko, M. (2007). "Bidirectional Long Short-Term Memory Networks for predicting the subcellular localization of eukaryotic proteins". *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*. **4** (3): 441–446. doi:10.1109/tcbb.2007.1015 (https://doi.org/10.1109%2Ftcbb.2007.1015).
93. Tax, N.; Verenich, I.; La Rosa, M.; Dumas, M. (2017). "Predictive Business Process Monitoring with LSTM neural networks" (https://link.springer.com/chapter/10.1007/978-3-319-59536-8_30). *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*. Lecture Notes in Computer Science. 10253: 477–492. arXiv:1612.02130 (https://arxiv.org/abs/1612.02130). doi:10.1007/978-3-319-59536-8_30 (https://doi.org/10.1007%2F978-3-319-59536-8_30). ISBN 978-3-319-59535-1.
94. Choi, E.; Bahadori, M.T.; Schuetz, E.; Stewart, W.; Sun, J. (2016). "Doctor AI: Predicting Clinical Events via Recurrent Neural Networks" (http://proceedings.mlr.press/v56/Choi16.html). *Proceedings of the 1st Machine Learning for Healthcare Conference*: 301–318.

- Mandic, D. & Chambers, J. (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley. ISBN 0-471-49517-4.

External links

- RNNSharp (https://github.com/zhongkaifu/RNNSharp) CRFs based on recurrent neural networks (C#, .NET)
- Recurrent Neural Networks (http://www.idsia.ch/~juergen/rnn.html) with over 60 RNN papers by Jürgen Schmidhuber's group at Dalle Molle Institute for Artificial Intelligence Research
- Elman Neural Network implementation (http://jsalatas.ictpro.gr/weka) for WEKA
- Recurrent Neural Nets & LSTMs in Java (http://deeplearning4j.org/recurrentnetwork)
- an alternative try for complete RNN / Reward driven (https://gammastorm.github.io/TheBrain/Brain.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=875191630"