Recent Advances in Recurrent Neural Networks

Hojjat Salehinejad, Julianne Baarbé, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee

***Abstract*—Recurrent neural networks (RNNs) are capable of learning features and long term dependencies from sequential and time-series data. The RNNs have a stack of non-linear units where at least one connection between units forms a directed cycle. A well-trained RNN can model any dynamical system; however, training RNNs is mostly plagued by issues in learning long-term dependencies. In this paper, we present a survey on RNNs and several new advances for newcomers and professionals in the field. The fundamentals and recent advances are explained and the research challenges are introduced.**

***Index Terms*—Deep learning, long-term dependency, recurrent neural networks, time-series analysis.**

I. Introduction

ARTIFICIAL neural networks (ANNs) are made from layers of connected units called artificial neurons. A "shallow network" refers to an ANN with one input layer, one output layer, and at most one hidden layer without a recurrent connection. As the number of layers increases, the complexity of network increases too. More number of layers or recurrent connections generally increases the depth of the network and empowers it to provide various levels of data representation and feature extraction, referred to as "deep learning". In general, these networks are made from nonlinear but simple units, where the higher layers provide a more abstract representation of data and suppresses unwanted variability [1]. Due to optimization difficulties caused by composition of the nonlinearity at each layer, not much work occurred on deep network architectures before significant advances in 2006 [2], [3]. ANNs with recurrent connections are called recurrent neural networks (RNNs), which are capable of modelling sequential data for sequence recognition and prediction [4]. RNNs are made of high dimensional hidden states with non-linear dynamics [5]. The structure of hidden states work as the memory of the network and state of the hidden layer at a time is conditioned on its previous state [6]. This structure enables the RNNs to store, remember, and process past complex signals for long time periods. RNNs can map an input sequence to the output sequence at the current timestep and predict the sequence in the next timestep.

H. Salehinejad is with the Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada, and Department of Medical Imaging, St. Michael's Hospital, University of Toronto, Toronto, Canada, e-mail: salehinejadh@smh.ca.

J. Baarbe is with the Institute of Medical Science, University of Toronto, Toronto, Canada, and Krembil Research Institute, University Health Network, Toronto, Canada, e-mail: j.baarbe@mail.utoronto.ca.

S. Sankar is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada, e-mail:sdsankar@edu.uwaterloo.ca.

J. Barfett and E. Colak are with the Department of Medical Imaging, St. Michael's Hospital, University of Toronto, Toronto, Canada, e-mail: {barfettj,cloake}@smh.ca.

S. Valaee is with the Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada, e-mail: valaee@ece.utoronto.ca.

TABLE I: Some of the major advances in recurrent neural networks (RNNs) at a glance.

Year	First Author	Contribution
1990	Elman	Popularized simple RNNs (Elman network)
1993	Doya	Teacher forcing for gradient descent (GD)
1994	Bengio	Difficulty in learning long term dependencies with gradient descend
1997	Hochreiter	LSTM: long short term memory for vanishing gradients problem
1997	Schuster	BRNN: Bidirectional recurrent neural networks
1998	LeCun	Hessian matrix approach for vanishing gradients problem
2000	Gers	Extended LSTM with forget gates
2001	Goodman	Classes for fast Maximum entropy training
2005	Morin	A hierarchical softmax function for language modeling using RNNs
2005	Graves	BLSTM: Bidirectional LSTM
2007	Jaeger	Leaky integration neurons
2007	Graves	MDRNN: Multi-dimensional RNNs
2009	Graves	LSTM for hand-writing recognition
2010	Mikolov	RNN based language model
2010	Neir	Rectified linear unit (ReLU) for vanishing gradient problem
2011	Martens	Learning RNN with Hessian-free optimization
2011	Mikolov	RNN by back-propagation through time (BPTT) for statistical language modelling
2011	Sutskever	Hessian-free optimization with structural damping
2011	Duchi	Adaptive learning rates for each weight
2012	Gutmann	Noise-contrastive estimation (NCE)
2012	Mnih	NCE for training neural probabilistic language models (NPLMs)
2012	Pascanu	Avoiding exploding gradient problem by gradient clipping
2013	Mikolov	Negative Sampling instead of hierarchical softmax
2013	Sutskever	Stochastic gradient descent (SGD) with momentum
2013	Graves	Deep LSTM RNNs (Stacked LSTM)
2014	Cho	Gated recurrent units
2015	Zaremba	Dropout for reducing Overfitting
2015	Mikolov	Structurally constrained recurrent network (SCRN) to enhance learning longer memory for vanishing gradient problem
2015	Visin	ReNet: A RNN-based alternative to convolutional neural networks
2015	Gregor	DRAW: Deep recurrent attentive writer
2015	Kalchbrenner	Grid Long Short-Term Memory
2015	Srivastava	Highway network
2017	Jing	Gated Orthogonal Recurrent Units

A large number of papers are published in the literature based on RNNs, from architecture design to applications development. In this paper, we focus on discussing discrete-time RNNs and recent advances in the field. Some of the major advances in RNNs through time are listed in Table I. The development of back-propagation using gradient descent (GD) has provided a great opportunity for training RNNs. This simple training approach has accelerated practical achievements in developing RNNs [5]. However, it comes with some challenges

in modelling long-term dependencies such as vanishing and exploding gradient problems, which are discussed in this paper.

The rest of paper is organized as follow. The fundamentals of RNNs are presented in Section II. Methods for training RNNs are discussed in Section III and a variety of RNNs architectures are presented in Section IV. The regularization methods for training RNNs are discussed in Section V. Finally, a brief survey on major applications of RNN in signal processing is presented in Section VI.

II. A Simple Recurrent Neural Network

RNNs are a class of supervised machine learning models, made of artificial neurons with one or more feedback loops [7]. The feedback loops are recurrent cycles over time or sequence (we call it time throughout this paper) [8], as shown in Figure 1. Training a RNN in a supervised fashion requires a training dataset of input-target pairs. The objective is to minimize the difference between the output and target pairs (i.e., the loss value) by optimizing the weights of the network.

A. Model Architecture

A simple RNN has three layers which are input, recurrent hidden, and output layers, as presented in Figure 1a. The input layer has N input units. The inputs to this layer is a sequence of vectors through time t such as $\{..., \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, ...\}$, where $\mathbf{x}_t = (x_1, x_2, ..., x_N)$. The input units in a fully connected RNN are connected to the hidden units in the hidden layer, where the connections are defined with a weight matrix $\mathbf{W}_{IH}$. The hidden layer has M hidden units $\mathbf{h}_t = (h_1, h_2, ..., h_M)$, that are connected to each other through time with recurrent connections, Figure 1b. The initialization of hidden units using small non-zero elements can improve overall performance and stability of the network [9]. The hidden layer defines the state space or "memory" of the system as

$$\mathbf{h}_t = f_H(\mathbf{o}_t), \tag{1}$$

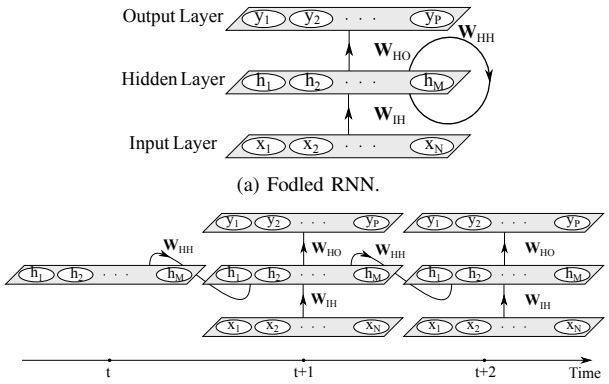
where

$$\mathbf{o}_t = \mathbf{W}_{IH}\mathbf{x}_t + \mathbf{W}_{HH}\mathbf{h}_{t-1} + \mathbf{b}_h, \tag{2}$$

$f_H(\cdot)$ is the hidden layer activation function, and $\mathbf{b}_h$ is the bias vector of the hidden units. The hidden units are connected to the output layer with weighted connections $\mathbf{W}_{HO}$. The output layer has P units $\mathbf{y}_t = (y_1, y_2, ..., y_P)$ that are computed as

$$\mathbf{y}_t = f_O(\mathbf{W}_{HO}\mathbf{h}_t + \mathbf{b}_o) \tag{3}$$

where $f_O(\cdot)$ is the activation functions and $\mathbf{b}_o$ is the bias vector in the output layer. Since the input-target pairs are sequential through time, the above steps are repeated consequently over time $t = (1, ..., T)$. The Eqs. (1) and (3) show a RNN is consisted of certain non-linear state equations, which are iterable through time. In each timestep, the hidden states provide a prediction at the output layer based on the input vector. The hidden state of a RNN is a set of values, which apart from the effect of any external factors, summarizes all the unique necessary information about the past states of the network over many timesteps. This integrated information can define future behaviour of the network and make accurate predictions at the output layer [5]. A RNN uses a simple nonlinear activation function in every unit. However, such simple structure is capable of modelling rich dynamics, if it is well trained through timesteps.



(a) Fodled RNN.

(b) Unfolded RNN through time.

Fig. 1: A simple recurrent neural network (RNN) and its unfolded structure through time t. Each arrow shows a full connection of units between the layers. To keep the figure simple, biases are not shown.

B. Activation Function

For linear networks, multiple linear hidden layers act as a single linear hidden layer [10]. Nonlinear functions are more powerful than linear functions as they can draw nonlinear boundaries. The nonlinearity in one or successive hidden layers in a RNN is the reason for learning input-target relationships.

Some of the most popular activation functions are presented in Figure 2. The "sigmoid", "tanh", and rectified linear unit (ReLU) have received more attention than the other activation functions recently. The "sigmoid" is a common choice, which takes a real-value and squashes it to the range $[0, 1]$. This activation function is normally used in the output layer, where a cross-entropy loss function is used for training a classification model. The "tanh" and "sigmoid" activation functions are defined as

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4}$$

and

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{5}$$

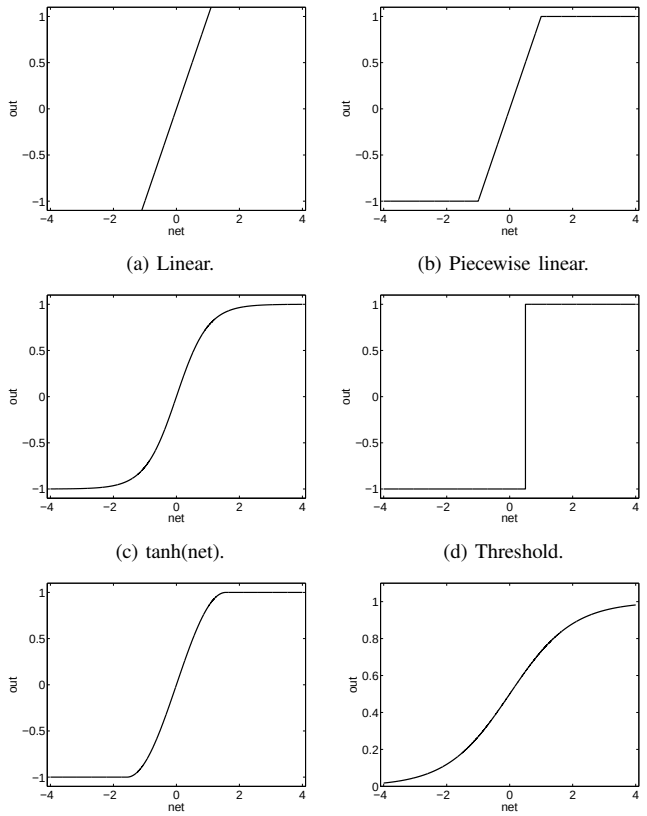
respectively. The "tanh" activation function is in fact a scaled "sigmoid" activation function such as

$$\sigma(x) = \frac{tanh(x/2) + 1}{2}. \tag{6}$$

ReLU is another popular activation function, which is open-ended for positive input values [3], defined as

$$y(x) = max(x, 0). \tag{7}$$

Selection of the activation function is mostly dependent on the problem and nature of the data. For example, "sigmoid" is suitable for networks where the output is in the range $[0, 1]$. However, the "tanh" and "sigmoid" activation functions



(a) Linear. (b) Piecewise linear.

(c) tanh(net). (d) Threshold.

(e) sin(net) until saturation. (f) Sigmoid.

Fig. 2: Most common activation functions.

saturate the neuron very fast and can vanish the gradient. Despite "tanh", the non-zero centered output from "sigmoid" can cause unstable dynamics in the gradient updates for the weights. The ReLU activation function leads to sparser gradients and greatly accelerates the convergence of stochastic gradient descent (SGD) compared to the "sigmoid" or "tanh" activation functions [11]. ReLU is computationally cheap, since it can be implemented by thresholding an activation value at zero. However, ReLU is not resistant against a large gradient flow and as the weight matrix grows, the neuron may remain inactive during training.

C. Loss Function

Loss function evaluates performance of the network by comparing the output $\mathbf{y}_t$ with the corresponding target $\mathbf{z}_t$ defined as

$$\mathcal{L}(\mathbf{y}, \mathbf{z}) = \sum_{t=1}^{T} \mathcal{L}_t(\mathbf{y}_t, \mathbf{z}_t), \tag{8}$$

that is an overall summation of losses in each timestep [12]. Selection of the loss function is problem dependent. Some popular loss function are Euclidean distance and Hamming distance for forecasing of real-values and cross-entropy over probablity distribution of outputs for classification problems [13].

III. Training Recurrent Neural Network

Efficient training of a RNN is a major problem. The difficulty is in proper initialization of the weights in the network and the optimization algorithm to tune them in order to minimize the training loss. The relationship among network parameters and the dynamics of the hidden states through time causes instability [4]. A glance at the proposed methods in the literature shows that the main focus is to reduce complexity of training algorithms, while accelerating the convergence. However, generally such algorithms take a large number of iterations to train the model. Some of the approaches for training RNNs are multi-grid random search, time-weighted pseudo-newton optimization, GD, extended Kalman filter (EKF) [15], Hessian-free, expectation maximization (EM) [16], approximated Levenberg-Marquardt [17], and global optimization algorithms. In this section, we discuss some of these methods in details. A detailed comparison is available in [18].

A. Initialization

Initialization of weights and biases in RNNs is critical. A general rule is to assign small values to the weights. A Gaussian draw with a standard deviation of 0.001 or 0.01 is a reasonable choice [9], [19]. The biases are usually set to zero, but the output bias can also be set to a very small value [9]. However, the initialization of parameters is dependent on the task and properties of the input data such as dimensionality [9]. Setting the initial weight using prior knowledge or in a semi-supervised fashion are other approaches [4].

B. Gradient-based Learning Methods

Gradient descent (GD) is a simple and popular optimization method in deep learning. The basic idea is to adjust the weights of the model by finding the error function derivatives with respect to each member of the weight matrices in the model [4]. To minimize total loss, GD changes each weight in proportion to the derivative of the error with respect to that weight, provided that the non-linear activation functions are differentiable. The GD is also known as batch GD, as it computes the gradient for the whole dataset in each optimization iteration to perform a single update as

$$\theta_{t+1} = \theta_t - \frac{\lambda}{U} \sum_{k=1}^{U} \frac{\partial \mathcal{L}_k}{\partial \theta} \tag{9}$$

where U is size of training set, λ is the learning rate, and θ is set of parameters. This approach is computationally expensive for very large datasets and is not suitable for online training (i.e., training the models as inputs arrive).

Since a RNN is a structure through time, we need to extend GD through time to train the network, called back-propagation through time (BPTT) [20]. However, computing error-derivatives through time is difficult [21]. This is mostly due to the relationship among the parameters and the dynamics of the RNN, that is highly unstable and makes GD ineffective. Gradient-based algorithms have difficulty in capturing dependencies as the duration of dependencies increases [4]. The derivatives of the loss function with respect to the weights

TABLE II: Comparing major GD methods. N is number of nodes in the network. The $O(\cdot)$ is per data point. More details in [14].

Method	Description	Advantages	Disadvantages	$O(\cdot)$
RTRL	Computing error gradient after obtaining gradients of the network states w.r.t weights at time t in terms of those at time $t-1$	- online updating of weights - suitable for online adaption property applications	- large computational complexity	$O(N^4)$
BPTT	Unfolding time iterations into layers with identical weights converts the recurrent network into an equivalent feedforward network, suitable for training with back-propagation method.	- computationally efficient - suitable for offline training	- not practical for online training	$O(N^2)$
FFP	Recursive computing of boundary conditions of back-propagated gradients at time $t=1$.	- on-line technique - solving the gradient recursion forward in time, rather than backwards.	- more computational complexity than BPTT method	$O(N^3)$
GF	Computing the solution using the sought error gradient based on the recursive equations for the output gradients and a dot product.	- improving RTRL computational complexity - online method	- more computational complexity than BPTT method	$O(N^3)$
BU	Updating the weights every $O(N)$ data points using some aspects of the RTRL and BTT methods.	- online method	- more computational complexity than BPTT method	$O(N^3)$

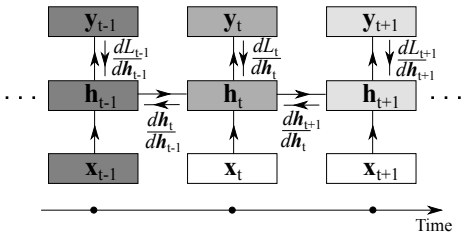


Fig. 3: As the network is receiving new inputs over time, the sensitivity of units decay (lighter shades in layers) and the back-propagation through time (BPTT) overwrites the activation in hidden units. This results in forgetting the early visited inputs.

only consider the distance between the current output and the corresponding target, without using the history information for weights updating [22]. RNNs cannot learn long-range temporal dependencies when GD is used for training [4]. This is due to the exponential decay of gradient, as it is back-propagated through time, which is called the vanishing gradient problem. In another occasional situation, the back-propagated gradient can exponentially blow-up, which increases the variance of the gradients and results in very unstable learning situation, called the exploding gradient problem [5]. These challenges are discussed in this section. A comparison of major GD methods is presented in Table II and an overview of gradient-based optimization algorithms is provided in [18].

1) Back-propagation through time (BPTT): BPTT is a generalization of back-propagation for feed-forward networks. The standard BPTT method for learning RNNs "unfolds" the network in time and propagates error signals backwards through time. By considering the network parameters in Figure 1b as the set $\theta = \{\mathbf{W}_{HH}, \mathbf{W}_{IH}, \mathbf{W}_{HO}, \mathbf{b}_H, \mathbf{b}_I, \mathbf{b}_O\}$ and $\mathbf{h}_t$ as the hidden state of network at time t, we can write the gradients as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial \theta} \tag{10}$$

where the expansion of loss function gradients at time t is

$$\frac{\partial \mathcal{L}_t}{\partial \theta} = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} \cdot \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \cdot \frac{\partial \mathbf{h}_k^+}{\partial \theta}\right) \tag{11}$$

where $\frac{\partial \mathbf{h}_k^+}{\partial \theta}$ is the partial derivative (i.e., "immediate" partial derivative). It describes how the parameters in the set θ affect the loss function at the previous timesteps (i.e., $k < t$). In order to transport the error through time from timestep t back to timestep k we can have

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}. \tag{12}$$

We can consider Eq. (12) as a Jacobian matrix for the hidden state parameters in Eq.(1) as

$$\prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{HH}^T diag|f_H^{'}(\mathbf{h}_{i-1})|, \tag{13}$$

where $f^{'}(\cdot)$ is the element-wise derivate of function $f(\cdot)$ and $diag(\cdot)$ is the diagonal matrix.

We can generally recognize the long-term and short-term contribution of hidden states over time in the network. The long-term dependency refers to the contribution of inputs and corresponding hidden states at time $k << t$ and short-term dependencies refer to other times [19]. Figure 3 shows that as the network makes progress over time, the contribution of the inputs x_{t-1} at discrete time $t-1$ vanishes through time to the timestep $t+1$ (the dark grey in the layers decays to higher grey). On the other hand, the contribution of the loss function value $\mathcal{L}_{t+1}$ with respect to the hidden state $\mathbf{h}_{t+1}$ at time $t+1$ in BPTT is more than the previous timesteps.

2) Vanishing Gradient Problem: According to the literature, it is possible to capture complex patterns of data in real-world by using a strong nonlinearity [6]. However, this may cause RNNs to suffer from the vanishing gradient problem [4]. This problem refers to the exponential shrinking of gradient magnitudes as they are propagated back through time. This phenomena causes memory of the network to ignore long

term dependencies and hardly learn the correlation between temporally distant events. There are two reasons for that: 1) Standard nonlinear functions such as the sigmoid function have a gradient which is almost everywhere close to zero; 2) The magnitude of gradient is multiplied over and over by the recurrent matrix as it is back-propagated through time. In this case, when the eigenvalues of the recurrent matrix become less than one, the gradient converges to zero rapidly. This happens normally after 5$\sim$10 steps of back-propagation [6].

In training the RNNs on long sequences (e.g., 100 timesteps), the gradients shrink when the weights are small. Product of a set of real numbers can shrink/explode to zero/infinity, respectively. For the matrices the same analogy exists but shrinkage/explosion happens along some directions. In [19], it is showed that by considering ρ as the spectral radius of the recurrent weight matrix W_{HH}, it is necessary at $\rho > 1$ for the long term components to explode as $t \to \infty$. It is possible to use singular values to generalize it to the non-linear function $f_H^{'}(\cdot)$ in Eq. (1) by bounding it with $\gamma \in \mathcal{R}$ such as

$$||diag(f_H^{'}(\mathbf{h}_k))|| \leq \gamma. \tag{14}$$

Using the Eq. (13), the Jacobian matrix $\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}$, and the bound in Eq. (14), we can have

$$||\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}|| \leq ||\mathbf{W}_{HH}^T|| \cdot ||diag(f_H^{'}(\mathbf{h}_k))|| \leq 1. \tag{15}$$

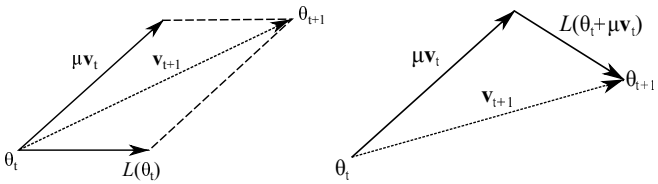
We can consider $||\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}|| \leq \delta < 1$ such as $\delta \in \mathcal{R}$ for each step k. By continuing it over different timesteps and adding the loss function component we can have

$$||\frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t}(\prod_{i=k}^{t-1} \frac{\partial \mathbf{h}_{i+1}}{\partial \mathbf{h}_i})|| \leq \delta^{t-k} ||\frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t}||. \tag{16}$$

This equation shows that as $t - k$ gets larger, the long-term dependencies move toward zero and the vanishing problem happens. Finally, we can see that the sufficient condition for the gradient vanishing problem to appear is that the largest singular value of the recurrent weights matrix $\mathbf{W}_{HH}$ (i.e., λ_1) satisfies $\lambda_1 < \frac{1}{\gamma}$ [19].

3) Exploding Gradient Problem: One of the major problems in training RNNs using BPTT is the exploding gradient problem [4]. Gradients in training RNNs on long sequences may explode as the weights become larger and the norm of the gradient during training largely increases. As it is stated in [19], the necessary condition for this situation to happen is $\lambda_1 > \frac{1}{\gamma}$.

In order to overcome the exploding gradient problem, many methods have been proposed recently. In 2012, Mikolov proposed a gradient norm-clipping method to avoid the exploding gradient problem in training RNNs with simple tools such as BPTT and SGD on large datasets. [23], [24]. In a similar approach, Pascanu has proposed an almost similar method to Mikolov, by introducing a hyper-parameter as threshold for norm-clipping the gradients [19]. This parameter can be set by heuristics; however, the training procedure is not very sensitive to that and behaves well for rather small thresholds.



(a) Classical momentum. (b) Nesterov accelerated gradient.

Fig. 4: The classical momentum and the Nesterov accelerated gradient schemes.

4) Stochastic Gradient Descent: The SGD (also called online GD) is a generalization of GD that is widely in use for machine learning applications [12]. The SGD is robust, scalable, and performs well across many different domains ranging from smooth and strongly convex problems to complex non-convex objectives. Despite the redundant computations in GD, the SGD performs one update at a time [25]. For an input-target pair $\{\mathbf{x}_k, \mathbf{z}\}$ in which $k \in \{1, ..., U\}$, the parameters in θ are updated according as

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial \mathcal{L}_k}{\partial \theta}. \tag{17}$$

Such frequent update causes fluctuation in the loss function outputs, which helps the SGD to explore the problem landscape with higher diversity with the hope of finding better local minima. An adaptive learning rate can control the convergence of SGD, such that as learning rate decreases, the exploration decreases and exploitation increases. It leads to faster convergence to a local minima. A classical technique to accelerate SGD is using momentum, which accumulates a velocity vector in directions of persistent reduction towards the objective across iterations [26]. The classical version of momentum applies to the loss function $\mathcal{L}$ at time t with a set of parameters θ as

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \lambda \nabla \mathcal{L}(\theta_t) \tag{18}$$

where $\nabla \mathcal{L}(\cdot)$ is the gradient of loss function and $\mu \in [0, 1]$ is the momentum coefficient [9], [12]. As figure 4a shows, the parameters in θ are updated as

$$\theta_{t+1} = \theta_t + \mathbf{v}_{t+1}. \tag{19}$$

By considering R as the condition number of the curvature at the minimum, the momentum can considerably accelerate convergence to a local minimum, requiring $\sqrt{R}$ times fewer iterations than steepest descent to reach the same level of accuracy [26]. In this case, it is suggested to set the learning rate to $\mu = (\sqrt{R} - 1)/(\sqrt{R} + 1)$ [26].

The Nesterov accelerated gradient (NAG) is a first-order optimization method that provides more efficient convergence rate for particular situations (e.g., convex functions with deterministic gradient) than the GD [27]. The main difference between NAG and GD is in the updating rule of the velocity vector $\mathbf{v}$, as presented in Figure 4b, defined as

$$\mathbf{v}_{t+1} = \mu \mathbf{v}_t - \lambda \nabla \mathcal{L}(\theta + \mu \mathbf{v}_t) \tag{20}$$

where the parameters in θ are updated using Eq. (19). By reasonable fine-tuning of the momentum coefficient μ, it is possible to increase the optimization performance [9].

5) Mini-Batch Gradient Descent: The mini-batch GD computes the gradient of a batch of training data which has more than one training sample. The typical mini-batch size is $50 \leq b \leq 256$, but can vary for different applications. Feeding the training samples in mini-batches accelerates the GD and is suitable for processing load distribution on graphical processing units (GPUs). The update rule modifies the parameters after b examples rather than needing to wait to scan the all examples such as

$$\theta_t = \theta_{t-1} - \frac{\lambda}{b} \sum_{k=i}^{i+b-1} \frac{\partial \mathcal{L}_k}{\partial \theta}. \tag{21}$$

Since the GD-based algorithms are generally dependent on instantaneous estimations of the gradient, they are slow for time series data [22] and ineffective on optimization of non-convex functions [28]. They also require setting of learning rate which is often tricky and application dependent.

The SGD is much faster than the GD and is useable for tracking of updates. However, since the mini-batch GD is easier for parallelization and can take advantage of vectorized implementation, it performs significantly better than GD and SGD [25]. A good vectorization can even lead to faster results compared to SGD. Also non-random initializations schemes, such as layer-wise pre-training, may help with faster optimization [29]. A deeper analyze is provided in [30].

6) Adam Stochastic Optimization: Adaptive Moment Estimation (Adam) is a first-order gradient-based optimization algorithm, which uses estimates of lower-order moments to optimize a stochastic objective function [31]. It needs initialization of first moment vector $\mathbf{m}_0$ and second moment vector $\mathbf{v}_0$ at time-stamp zero. These vector are updated as

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1 - \beta_1) g_{t+1} \tag{22}$$

and

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1 - \beta_2) g_{t+1}^2, \tag{23}$$

where g_{t+1} is the gradient of loss function. The exponential decay rates for the moment estimates are recommended to be $\beta_1 = 0.9$ and $\beta_2 = 0.999$ [31]. The bias correction of first and second moment estimates are

$$\hat{\mathbf{m}}_{t+1} = \hat{\mathbf{n}}_{t+1} = \mathbf{v}_t / (1 - \beta_1^{t+1}), \tag{24}$$

and

$$\hat{\mathbf{v}}_{t+1} = \mathbf{v}_t / (1 - \beta_2^{t+1}). \tag{25}$$

Then, the parameters are updated as

$$\theta_{t+1} = \theta_t - \frac{\alpha \cdot \hat{\mathbf{m}}_{t+1}}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \tag{26}$$

where $\epsilon = 10^{-8}$. The Adam algorithm is relatively simple to implement and is suitable for problems with very large datasets [31].

C. Extended Kalman Filter-based Learning

Kalman filter is a method of predicting the future state of a system based on a series of measurements observed over time by using Bayesian inference and estimating a joint probability distribution over the variables for each timestep [32]. The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter. It relaxes the linear prerequisites of the state transition and observation models. However, they may instead need to be differentiable functions. The EKF trains RNNs with the assumption that the optimum setting of the weights is stationary [22], [33]. Comparing to back-propagation, the EKF helps RNNs to reach the training steady state much faster for non-stationary processes. It can excel the back-propagation algorithm in training with limited data [15]. Similar to SGD, it can train a RNN with incoming input data in an online manner [33].

A more efficient and effective version of EKF is the decoupled EKF (DEKF) method, which ignores the inter-dependencies of mutually exclusive groups of weights [32]. This technique can lower the computational complexity and the required storage per training instance. The decoupled extended Kalman filter (DEKF) applies the extended Kalman filter independently to each neuron in order to estimate the optimum weights feeding it. By proceeding this way, only local interdependencies are considered. The training procedure is modeled as an optimal filtering problem. It recursively and efficiently computes a solution to the least-squares problem to find the best fitted curve for a given set of data in terms of minimizing the average distance between data and curve. At a timestep t, all the information supplied to the network until time t is used, including all derivatives computed since the first iteration of the learning process. However, computation requires just the results from the previous step and there is no need to store results beyond that step [22]. Kalman-based models in RNNs are computationally expensive and have received little attention in the past years.

D. Second Order Optimization

The second order optimization algorithms use information of the second derivate of a function. With the assumption of having a quadratic function with good second order expansion approximation, Newton's method can perform better and faster than GD by moving toward the global minimum [34]. This is while the direction of optimization in GD is against the gradient and gets stuck near saddle points or local extrema. The other challenge with GD-based models is setting of learning rate, which is often tricky and application dependent. However, second order methods generally require computing the Hessian matrix and inverse of Hessian matrix, which is a difficult task to perform in RNNs comparing to GD approaches.

A general recursive Bayesian Levenberg-Marquardt algorithm can sequentially update the weights and the Hessian matrix in recursive second-order training of a RNN [35]. Such approach outperforms standard real-time recurrent learning and EKF training algorithms for RNNs [35]. The challenges

in computing Hessian matrix for time-series are addressed by introducing Hessian free (HF) optimization [34].

E. Hessian-Free Optimization

A well-designed and well-initialized HF optimizer can work very well for optimizing non-convex functions, such as training the objective function for deep neural networks, given sensible random initializations [34]. Since RNNs share weights across time, the HF optimizer should be a good optimization candidate [5]. Training RNNs via HF optimization can reduce training difficulties caused by gradient-based optimization [36]. In general, HF and truncated Newton methods compute a new estimate of the Hessian matrix before each update step and can take into account abrupt changes in curvature [19]. HF optimization targets unconstrained minimization of real-valued smooth objective functions [28]. Like standard Newton's method, it uses local quadratic approximations to generate update proposals. It belongs to the broad class of approximate Newton methods that are practical for problems of very high dimensionality, such as the training objectives of large neural networks [28].

With the addition of a novel damping mechanism to a HF optimizer, the optimizer is able to train a RNN on pathological synthetic datasets, which are known to be impossible to learn with GD [28]. Multiplicative RNNs (MRNNs) uses multiplicative (also called "gated") connections to allow the current input character to determine the transition matrix from one hidden state vector to the next [5]. This method demonstrates the power of a large RNN trained with this optimizer by applying them to the task of predicting the next character in a stream of text [5], [12].

The HF optimizer can be used in conjunction with or as an alternative to existing pre-training methods and is more widely applicable, since it relies on fewer assumptions about the specific structure of the network. HF optimization operates on large mini batches and is able to detect promising directions in the weight space that have very small gradients but even smaller curvature. Similar results have been achieved by using SGD with momentum and carefully initializating weights [9].

F. Global Optimization

In general, evolutionary computing methods initialize a population of search agents and evolve them to find local/global optimization solution(s) [37]. These methods can solve a wide range of optimization problems including multimodal, ill-behaved, high-dimensional, convex, and non-convex problems. However, evolutionary algorithms have some drawbacks in optimization of RNNs including getting stuck in local minima/maxima, slow speed of convergence, and network stagnancy.

Optimization of the parameters in RNNs can be modelled as a nonlinear global optimization problem. The most common global optimization method for training RNNs is genetic algorithms [38]. The alopex-based evolutionary algorithm (AEA) uses local correlations between changes in individual weights and changes in the global error measure and simultaneously updates all the weights in the network using only local computations [39]. Selecting the optimal topology of neural network for a particular application is a different task from optimizing the network parameters. A hybrid multi-objective evolutionary algorithm that trains and optimizes the structure of a RNN for time series prediction is proposed in [40]. Some models simultaneously acquire both the structure and weights for recurrent networks [38]. The covariance matrix adaptation evolution strategy (CMA-ES) is a global optimization method used for tuning the parameters of a RNN for language models [41]. Published literature on global optimization methods for RNNs is scattered and has not received much attention from the research community.This lack is mainly due to the computational complexity of these methods. However, the multi-agent philosophy of such methods in a low computational complexity manner, such as models with small population size [42], may result in much better performance than SGD.

IV. Recurrent Neural Networks Architectures

This section aims to provide an overview on the different architectures of RNNs and discuss the nuances between these models.

A. Deep RNNs with Multi-Layer Perceptron

Deep architectures of neural networks can represent a function exponentially more efficient than shallow architectures. While recurrent networks are inherently deep in time given each hidden state is a function of all previous hidden states [43], it has been shown that the internal computation is in fact quite shallow [44]. In [44], it is argued that adding one or more nonlinear layers in the transition stages of a RNN can improve overall performance by better disentangling the underlying variations the original input. The deep structures in RNNs with perceptron layers can fall under three categories: input to hidden, hidden to hidden, and hidden to output [44].

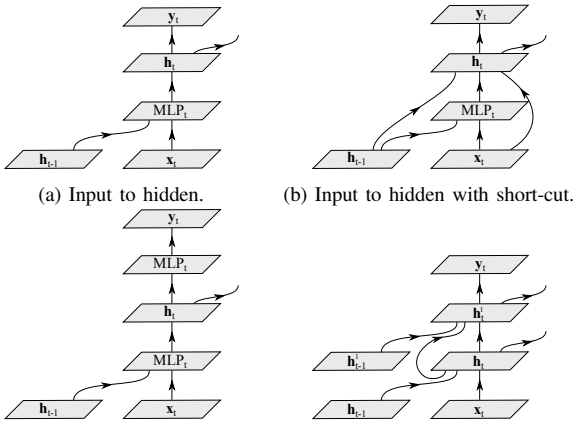
1) Deep input to hidden: One of the basic ideas is to bring the multi-layer perceptron (MLP) structure into the transition and output stages, called deep transition RNNs and deep output RNNs, respectively. To do so, two operators can be introduced. The first is a plus $\oplus$ operator, which receives two vectors, the input vector $\mathbf{x}$ and hidden state $\mathbf{h}$, and returns a summary as

$$\mathbf{h}^{'} = \mathbf{x} \oplus \mathbf{h}. \tag{27}$$

This operator is equivalent to the Eq. (1). The other operator is a predictor denoted as $\triangleright$, which is equivalent to the Eq. (3) and predicts the output of a given summary $\mathbf{h}$ as

$$\mathbf{y} = \triangleright\mathbf{h}. \tag{28}$$

Higher level representation of input data means easier representation of relationships between temporal structures of data. This technique has achieved better results than feeding the network with original data in speech recognition [43] and word embedding [45] applications. Structure of a RNN with an MLP in the input to hidden layers is presented in Figure 5a. In order to enhance long-term dependencies, an additional connection makes a short-cut between the input and hidden layer as in Figure 5b [44].



(a) Input to hidden. (b) Input to hidden with short-cut.

(c) Hidden to hidden and output. (d) Stack of hidden states.

Fig. 5: Some deep recurrent neural network (RNN) architectures with multi-layer perceptron (MLP).

2) Deep hidden to hidden and output: The most focus for deep RNNs is in the hidden layers. In this level, the procedure of data abstraction and/or hidden state construction from previous data abstractions and new inputs is highly non-linear. An MLP can model this non-linear function, which helps a RNN to quickly adapt to fast changing input modes while still having a good memory of past events. A RNN can have both an MLP in transition and an MLP before the output layer (an example is presented in Figure 5c) [44]. A deep hidden to output function can disentangle the factors of variations in the hidden state and facilitate prediction of the target. This function allows a more compact hidden state of the network, which may result in a more informative historical summary of the previous inputs.

3) Stack of hidden states: Another approach to construct deep RNNs is to have a stack of hidden recurrent layers as shown in Figure 5d. This style of recurrent levels encourages the network to operate at different timescales and enables it to deal with multiple time scales of inputs sequences [44]. However, the transitions between consecutive hidden states is often shallow, which results in a limited family of functions it can represent [44]. Therefore, this function cannot act as a universal approximation, unless the higher layers have feedback to the lower layers.

While the augmentation of a RNN for leveraging the benefits of deep networks has shown to yield performance improvements, it has also shown to introduce potential issues. By adding nonlinear layers to the network transition stages, there now exists additional layers through which the gradient must travel back. This can lead to issues such as vanishing and exploding gradients which can cause the network to fail to adequately capture long-term dependencies [44]. The addition of nonlinear layers in the transition stages of a RNN can also significantly increase the computation and speed of the network. Additional layers can significantly increase the training time of the network, must be unrolled at each iteration of training, and can thus not be parallelized.

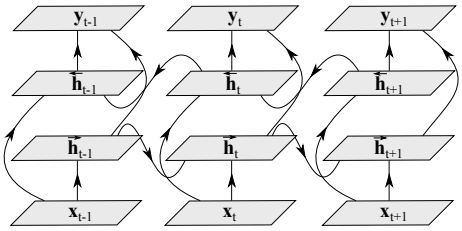


Fig. 6: Unfolded through time bi-directional recurrent neural network (BRNN).

B. Bidirectional RNN

Conventional RNNs only consider the previous context of data for training. While simply looking at previous context is sufficient in many applications such as speech recognition, it is also useful to explore the future context as well [43]. Previously, the use of future information as context for current prediction have been attempted in the basic architecture of RNNs by delaying the output by a certain number of time frames. However this method required a handpicked optimal delay to be chosen for any implementation. A bi-directional RNN (BRNN) considers all available input sequence in both the past and future for estimation of the output vector [46]. To do so, one RNN processes the sequence from start to end in a forward time direction. Another RNN processes the sequence backwards from end to start in a negative time direction as demonstrated in Figure 6. Outputs from forward states are not connected to inputs of backward states and vice versa and there are no interactions between the two types of state neurons [46]. In Figure 6, the forward and backward hidden sequences are denoted by $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$, respectively, at time t. The forward hidden sequence is computed as

$$\overrightarrow{\mathbf{h}}_t = f_H(\mathbf{W}_{\overrightarrow{IH}}\mathbf{x}_t + \mathbf{W}_{\overrightarrow{HH}}\overrightarrow{\mathbf{h}}_{t-1} + \mathbf{b}_{\overrightarrow{\mathbf{h}}}), \tag{29}$$

where it is iterated over $t = (1, ..., T)$. The backward layer is

$$\overleftarrow{\mathbf{h}}_t = f_H(\mathbf{W}_{\overleftarrow{IH}}\mathbf{x}_t + \mathbf{W}_{\overleftarrow{HH}}\overleftarrow{\mathbf{h}}_{t-1} + \mathbf{b}_{\overleftarrow{\mathbf{h}}}), \tag{30}$$

which is iterated backward over time $t = (T, ..., 1)$. The output sequence $\mathbf{y}_t$ at time t is

$$\mathbf{y}_t = \mathbf{W}_{\overrightarrow{HO}}\overrightarrow{\mathbf{h}}_t + \mathbf{W}_{\overleftarrow{HO}}\overleftarrow{\mathbf{h}}_t + \mathbf{b}_o. \tag{31}$$

BPTT is one option to train BRNNs. However, the forward and backward pass procedures are slightly more complicated because the update of state and output neurons can no longer be conducted one at a time [46]. While simple RNNs are constrained by inputs leading to the present time, the BRNNs extend this model by using both past and future information. However, the shortcoming of BRNNs is their requirement to know the start and end of input sequences in advance. An example is labeling spoken sentences by their phonemes [46].

C. Recurrent Convolutional Neural Networks

The rise in popularity of RNNs can be attributed to its ability to model sequential data. Previous models examined have augmented the underlying structure of a simple RNN

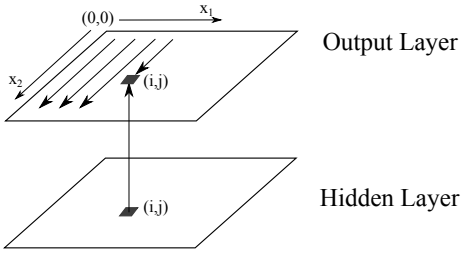


Fig. 7: Forward pass with sequence ordering in two-dimensional recurrent neural network (RNN). The connections within the hidden layer plane are recurrent. The lines along x_1 and x_2 show the scanning strips along which previous points were visited, starting at the top left corner.

to improve its performance on learning the contextual dependencies of single dimension sequences. However, there exists several problems, which require understanding of contextual dependencies over multiple dimensions. The most popular network architectures use convolutional neural networks (CNNs) to tackle these problems.

CNNs are very popular models for machine vision applications. CNNs may consist of multiple convolutional layers, optionally with pooling layers in between, followed by fully connected perceptron layers [11]. Typical CNNs learn through the use of convolutional layers to extract features using shared weights in each layer. The feature pooling layer (i.e., subsampling) generalizes the network by reducing the resolution of the dimensionality of intermediate representations (i.e., feature maps) as well as the sensitivity of the output to shifts and distortions. The extracted features, at the very last convolutional layer, are fed to fully connected perceptron model for dimensionality reduction of features and classification.

Incorporation of recurrent connections into each convolutional layer can shape a recurrent convolutional neural network (RCNN) [47]. The activation of units in RCNN evolve over time, as they are dependent on the neighboring unit. This approach can integrate the context information, important for object recognition tasks. This approach increases the depth of model, while the number of parameters is constant by weight sharing between layers. Using recurrent connections from the output into the input of the hidden layer allows the network to model label dependencies and smooth its own outputs based on its previous outputs [48]. This RCNN approach allows a large input context to be fed to the network while limiting the capacity of the model. This system can model complex spatial dependencies with low inference cost. As the context size increases with the built-in recurrence, the system identifies and corrects its own errors [48]. Quad-directional 2-dimensional RNNs can enhance CNNs to model long range spatial dependencies [49]. This method efficiently embeds the global spatial context into the compact local representation [49].

D. Multi-Dimensional Recurrent Neural Networks

Multi-dimensional recurrent neural networks (MDRNNs) are another implementation of RNNs to high dimensional sequence learning. This network utilizes recurrent connections for each dimension to learn correlations in the data. MDRNNs are a special case of directed acyclic graph RNNs [50], generalized to multidimensional data by replacing the one-dimensional chain of network updates with a D-dimensional grid [51]. In this approach, the single recurrent connection is replaced with recurrent connections of size D. A 2-dimensional example is presented in Figure 7. During the forward pass at each timestep, the hidden layer receives an external input as well as its own activation from one step back along all dimensions. A combination of the input and the previous hidden layer activation at each timestep is fed in the order of input sequence. Then, the network stores the resulting hidden layer activation [52]. The error gradient of an MDRNN can be calculated with BPTT. As with one dimensional BPTT, the sequence is processed in the reverse order of the forward pass. At each timestep, the hidden layer receives both the output error derivatives and its own future derivatives [52].

RNNs have suitable properties for multidimensional domains such as robustness to warping and flexible use of context. Furthermore, RNNs can also leverage inherent sequential patterns in image analysis and video processing that are often ignored by other architectures [53]. However, memory usage can become a significant problem when trying to model multidimensional sequences. As more recurrent connections in the network are increased, so too is the amount of saved states that the network must conserve. This can result in huge memory requirements, if there is a large number of saved states in the network. MDRNNs also fall victim to vanishing gradients and can fail to learn long-term sequential information along all dimensions. While applications of the MDRNN fall in line with RCNNs, there has yet to be any comparative examinations performed on the two models.

E. Long-Short Term Memory

Recurrent connections can improve performance of neural networks by leveraging their ability to understand sequential dependencies. However, the memory produced from the recurrent connections can severely be limited by the algorithms employed for training RNNs. All the models thus far have fallen victim to exploding or vanishing gradients during the training phase, resulting in the network failing to learn long-term sequential dependencies in data. The following models are specifically designed to tackle this problem, the most popular being the long-short term memory (LSTM) RNNs.

LSTM is one of the most popular and efficient methods for reducing the effects of vanishing and exploding gradients [54]. This approach changes the structure of hidden units from "sigmoid" or "tanh" to memory cells, in which their inputs and outputs are controlled by gates. These gates control flow of information to hidden neurons and preserve extracted features from previous timesteps [21], [54].

It is shown that for a continual sequence, the LSTM model's internal values may grow without bound [55]. Even when continuous sequences have naturally reoccurring properties, the network has no way to detect which information is no longer relevant. The forget gate learns weights that control the rate at which the value stored in the memory cell decays [55]. For periods when the input and output gates are off and the forget gate is not causing decay, a memory cell simply holds

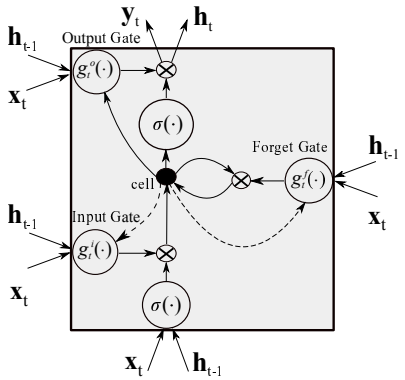


Fig. 8: The LSTM memory block with one cell. The dashed line represent time lag.

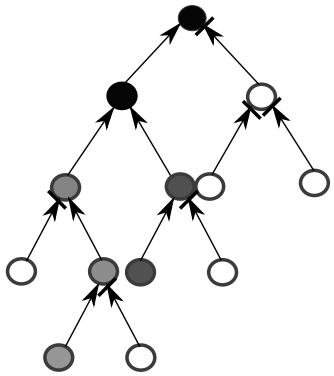


Fig. 9: An example of S-LSTM, a long-short term memory network on tree structures. A tree node can consider information from multiple descendants. Information of the other nodes in white are blocked. The short line (-) at each arrowhead indicates a block of information.

its value over time so that the gradient of the error stays constant during back-propagation over those periods [21]. This structure allows the network to potentially remember information for longer periods.

LSTM suffers from high complexity in the hidden layer. For identical size of hidden layers, a typical LSTM has about four times more parameters than a simple RNN [6]. The objective at the time of proposing the LSTM method was to introduce a scheme that could improve learning long-range dependencies, rather than to find the minimal or optimal scheme [21]. Multi-dimensional and grid LSTM networks have partially enhanced learning of long-term dependencies comparing to simple LSTM, which are discussed in this section.

1) Standard LSTM: A typical LSTM cell is made of input, forget, and output gates and a cell activation component as shown in Figure 8. These units receive the activation signals from different sources and control the activation of the cell by the designed multipliers. The LSTM gates can prevent the rest of the network from modifying the contents of the memory cells for multiple timesteps. LSTM networks preserve signals and propagate errors for much longer than ordinary RNNs. These properties allow LSTM networks to process data with complex and separated interdependencies and to excel in a range of sequence learning domains.

The input gate of LSTM is defined as

$$\mathbf{g}_t^i = \sigma(\mathbf{W}_{Ig^i}\mathbf{x}_t + \mathbf{W}_{Hg^i}\mathbf{h}_{t-1} + \mathbf{W}_{g^cg^i}\mathbf{g}_{t-1}^c + \mathbf{b}_{g^i}), \quad (32)$$

where $\mathbf{W}_{Ig^i}$ is the weight matrix from the input layer to the input gate, $\mathbf{W}_{Hg^i}$ is the weight matrix from hidden state to the input gate, $\mathbf{W}_{g^cg^i}$ is the weight matrix from cell activation to the input gate, and b_{g^i} is the bias of the input gate. The forget gate is defined as

$$\mathbf{g}_t^f = \sigma(\mathbf{W}_{Ig^f}\mathbf{x}_t + \mathbf{W}_{Hg^f}\mathbf{h}_{t-1} + \mathbf{W}_{g^cg^f}\mathbf{g}_{t-1}^c + \mathbf{b}_{g^f}), \quad (33)$$

where $\mathbf{W}_{Ig^f}$ is the weight matrix from the input layer to the forget gate, $\mathbf{W}_{Hg^f}$ is the weight matrix from hidden state to the forget gate, $\mathbf{W}_{g^cg^f}$ is the weight matrix from cell activation to the forget gate, and b_{g^f} is the bias of the forget gate. The cell gate is defined as

$$\mathbf{g}_t^c = \mathbf{g}_t^i\ tanh(\mathbf{W}_{Ig^c}\mathbf{x}_t + \mathbf{W}_{Hg^c}\mathbf{h}_{t-1} + \mathbf{b}_{g^c}) + \mathbf{g}_t^f\mathbf{g}_{t-1}^c, \quad (34)$$

where $\mathbf{W}_{Ig^c}$ is the weight matrix from the input layer to the cell gate, $\mathbf{W}_{Hg^c}$ is the weight matrix from hidden state to the cell gate, and b_{g^c} is the bias of the cell gate. The output gate is defined as

$$\mathbf{g}_t^o = \sigma(\mathbf{W}_{Ig^o}\mathbf{x}_t + \mathbf{W}_{Hg^o}\mathbf{h}_{t-1} + \mathbf{W}_{g^cg^o}\mathbf{g}_t^c + \mathbf{b}_{g^o}), \quad (35)$$

where $\mathbf{W}_{Ig^o}$ is the weight matrix from the input layer to the output gate, $\mathbf{W}_{Hg^o}$ is the weight matrix from hidden state to the output gate, $\mathbf{W}_{g^cg^o}$ is the weight matrix from cell activation to the output gate, and $\mathbf{b}_{g^o}$ is the bias of the output gate. Finally, the hidden state is computed as

$$\mathbf{h}_t = \mathbf{g}_t^o\ tanh(\mathbf{g}_t^c). \quad (36)$$

2) S-LSTM: While the LSTM internal mechanics help the network to learn longer sequence correlation, it may fail to understand input structures more complicated than a sequence. The S-LSTM model is designed to overcome the gradient vanishing problem and learn longer term dependencies from input. An S-LSTM network is made of S-LSTM memory blocks and works based on a hierarchical structure. A typical memory block is made of input and output gates. In this tree structure, presented in Figure 9, the memory of multiple descendant cells over time periods are reflected on a memory cell recursively. This method learns long term dependencies over the input by considering information from long-distances on the tree (i.e., branches) to the principal (i.e., root). A typical S-LSTM has "sigmoid" function and therefore, the gating signal works in the range of [0,1]. Figure 9 shows that the closer gates to the root suffer less from gradient vanishing problem (darker circle) while the branches at lower levels of tree loose their memory due to gradient vanishing (lighter circles). A gate can be closed to not receive signal from lower branches using a dash.

The S-LSTM method can achieve competitive results comparing to the recursive and LSTM models. It has the potential of extension to other LSTM models. However, its performance is not compared with other state-of-the-art LSTM models. The reader may refer to [56] for more details about S-LSTM memory cell.

3) Stacked LSTM: The idea of depth in ANNs is also applicable to LSTMs by stacking different hidden layers with LSTM cells in space to increase the network capacity [43], [57]. A hidden layer l in a stack of L LSTMs using the hidden layer in Eq. (1) is defined as

$$\mathbf{h}_t^l = f_H(\mathbf{W}_{IH}\mathbf{h}_t^{l-1} + \mathbf{W}_{HH}\mathbf{h}_{t-1}^l + \mathbf{b}_h^l), \tag{37}$$

where the hidden vector sequence $\mathbf{h}_t^l$ is computed over time $t = (1, ..., T)$ for $l = (1, ..., L)$. The initial hidden vector sequence is defined using the input sequence $\mathbf{h}^0 = (\mathbf{x}_1, ..., \mathbf{x}_T)$ [43]. Then, the output of the network is

$$\mathbf{y}_t = f_O(\mathbf{W}_{HO}\mathbf{h}_t^L + \mathbf{b}_0). \tag{38}$$

In stacked LSTM, a stack pointer can determine which cell in the LSTM provides state and memory cell of a previous timestep [58]. In such a controlled structure, not only the controller can push to and pop from the top of the stack in constant time but also an LSTM can maintain a continuous space embedding of the stack contents [58], [59].

The combination of stacked LSTM with different RNN structures for different applications needs investigation. One example is combination of stacked LSTM with frequency domain CNN for speech processing [43], [60].

4) Bidirectional LSTM: It is possible to increase capacity of BRNNs by stacking hidden layers of LSTM cells in space, called deep bidirectional LSTM (BLSTM) [43]. BLSTM networks are more powerful than unidirectional LSTM networks [61]. These networks theoretically involve all information of input sequences during computation. The distributed representation feature of BLSTM is crucial for different applications such as language understanding [62]. The BLSTM model leverages the same advantages discussed in the Bidirectional RNN section, while also overcoming the the vanishing gradient problem.

5) Multidimensional LSTM: The classical LSTM model has a single self-connection which is controlled by a single forget gate. Its activation is considered as one dimensional LSTM. Multi-dimensional LSTM (MDLSTM) uses interconnection from previous state of cell to extend the memory of LSTM along every N dimensions [52], [63]. The MDLSTM receives inputs in a N-dimensional arrangement (e.g. two dimensions for an image). Hidden state vectors $(\mathbf{h}_1, ..., \mathbf{h}_N)$ and memory vectors $(\mathbf{m}_1, ..., \mathbf{m}_N)$ are fed to each input of the array from the previous state for each dimension. The memory vector is defined as

$$\mathbf{m} = \sum_{j=1}^{N} \mathbf{g}_j^f \odot \mathbf{m}_j + \mathbf{g}_j^i \odot \mathbf{g}_j^c, \tag{39}$$

where $\odot$ is the element-wise product and the gates are computed using Eq.(32) to Eq.(36), [57].

Spatial LSTM is a particular case of MDLSTM [64], which is a two-dimensional grid for image modelling. This model generates a hidden state vector for a particular pixel in an image by sequentially reading the pixels in its small neighbourhood [64]. The state of the pixel is generated by feeding the state hidden vector into a factorized mixture of conditional Gaussian scale mixtures (MCGSM) [64].

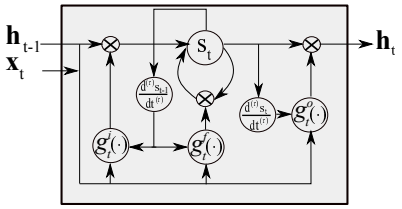


Fig. 10: Architecture of the differential recurrent neural network (dRNN) at time t. The input gate i and the forget gate f are controlled by the DoS at times $t-1$ and t, respectively, [65].

6) Grid LSTM: The MDLSTM model becomes unstable, as the grid size and LSTM depth in space grows. The grid LSTM model provides a solution by altering the computation of output memory vectors. This method targets deep sequential computation of multi-dimensional data. The model connects LSTM cells along the spatiotemporal dimensions of input data and between the layers. Unlike the MDLSTM model, the block computes N transforms and outputs N hidden state vectors and N memory vectors. The hidden sate vector for dimension j is

$$\mathbf{h}_j^{'} = LSTM(\mathbf{H}, \mathbf{m}_j, \mathbf{W}_{g^i}^j, \mathbf{W}_{g^f}^j, \mathbf{W}_{g^i}^o, \mathbf{W}_{g^c}^j), \tag{40}$$

where $LSTM(\cdot)$ is the standard LSTM procedure [57] and $\mathbf{H}$ is concatenation of input hidden state vectors defined as

$$\mathbf{H} = [\mathbf{h}_1, ..., \mathbf{h}_N]^T. \tag{41}$$

A two-dimension grid LSTM network adds LSTM cells along the spatial dimension to a stacked LSTM. A three or more dimensional LSTM is similar to MSLSTM, however, has added LSTM cells along the spatial depth and performs N-way interaction. More details on grid LSTM are provided in [57].

7) Differential Recurrent Neural Networks: While LSTMs have shown improved learning ability in understanding long term sequential dependencies, it has been argued that its gating mechanisms have no way of comprehensively discriminating between salient and non-salient information in a sequence [65]. Therefore, LSTMs fail to capture spatio-temporal dynamic patterns in tasks such as action recognition [65], in which sequences can often contain many non-salient frames. Differential recurrent neural networks (dRNNs) refer to detecting and capturing of important spatio-temporal sequences to learn dynamics of actions in input [65]. A LSTM gate in dRNNs monitors alternations in information gain of important motions between successive frames. This change of information is detectable by computing the derivative of hidden states (DoS). A large DoS reveals sudden change of actions state, which means the spatio-temporal structure contains informative dynamics. In this situation, the gates in Figure 10 allow flow of information to update the memory cell defined as

$$\mathbf{s}_t = \mathbf{g}_t^f \odot \mathbf{s}_{t-1} + \mathbf{g}_i^t \odot \mathbf{s}_{t-1/2} \tag{42}$$

where

$$\mathbf{s}_{t-1/2} = tanh(\mathbf{W}_{hs}\mathbf{h}_{t-1} + \mathbf{W}_{xs}\mathbf{X}_t + \mathbf{b}_s). \tag{43}$$

The DoS $d\mathbf{s}_t/d_t$ quantifies the change of information at each time t. Small DoS keeps the memory cell away from any

influence by the input. More specifically, the cell controls the input gate as

$$\mathbf{g}_t^i = \sigma(\sum_{r=0}^{R} \mathbf{W}_{dg^i}^{(r)} \frac{d^{(r)}\mathbf{s}_{t-1}}{dt^{(r)}} + \mathbf{W}_{hg^i}\mathbf{h}_{t-1} + \mathbf{W}_x g^i \mathbf{x}_t + \mathbf{b}_{g^i}), \tag{44}$$

the forget gate unit as

$$\mathbf{g}_t^f = \sigma(\sum_{r=0}^{R} \mathbf{W}_{dg^f}^{(r)} \frac{d^{(r)}\mathbf{s}_{t-1}}{dt^{(r)}} + \mathbf{W}_{hg^f}\mathbf{h}_{t-1} + \mathbf{W}_{xg^f}\mathbf{x}_t + \mathbf{b}_{g^f}), \tag{45}$$

and the output gate unit as

$$\mathbf{g}_t^o = \sigma(\sum_{r=0}^{R} \mathbf{W}_{dg^o}^{(r)} \frac{d^{(r)}\mathbf{s}_t}{dt^{(r)}} + \mathbf{W}_{hg^o}\mathbf{h}_{t-1} + \mathbf{W}_{xg^o}\mathbf{x}_t + \mathbf{b}_{g^o}), \tag{46}$$

where the DoS has an upper order limit of R. BPTT can train dRNNs. The 1-order and 2-order dRNN have better performance in training comparing with the simple LSTM; however, it has additional computational complexity.

8) Other LSTM Models: The local-global LSTM (LG-LSTM) architecture is initially proposed for semantic object parsing [66]. The objective is to improve exploitation of complex local (neighbourhood of a pixel) and global (whole image) contextual information on each position of an image. The current version of LG-LSTM has appended a stack of LSTM layers to intermediate convolutional layers. This technique directly enhances visual features and allows an end-to-end learning of network parameters [66]. Performance comparison of LG-LSTM with a variety of CNN models show high accuracy performance [66]. It is expected that this model can achieve more success by replacing all convolutional layers with LG-LSTM layers.

The matching LSTM (mLSTM) is initially proposed for natural language inference. The matching mechanism stores (remembers) the critical results for the final prediction and forgets the less important matchings [62]. The last hidden state of the mLSTM is useful to predict the relationship between the premise and the hypothesis. The difference with other methods is that instead of a whole sentence embeddings of the premise and the hypothesis, the sLSTM performs a word-by-word matching of the hypothesis with the premise [62].

The recurrence in both time and frequency for RNN, named F-T-LSTM, is proposed in [67]. This model generates a summary of the spectral information by scanning the frequency bands using a frequency LSTM. Then, it feeds the output layers activations as inputs to a LSTM. The formulation of frequency LSTM is similar to the time LSTM [67]. A convolutional LSTM (ConvLSTM) model with convolutional structures in both the input-to-state and state-to-state transitions for precipitation now-casting is proposed in [68]. This model uses a stack of multiple ConvLSTM layers to construct an end-to-end trainable model [68].

F. Gated Recurrent Unit

While LSTMs have shown to be a viable option for avoiding vanishing or exploding gradients, they have a higher memory requirement given multiple memory cells in their architecture.

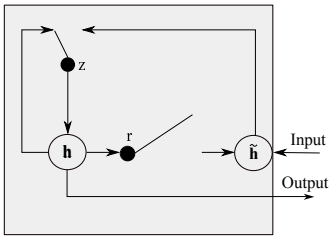


Fig. 11: A gated recurrent unit (GRU). The update gate z decides if the hidden state is to be updated with a new hidden state $\tilde{h}$. The reset gate r controls if the previous hidden state needs to be ignored.

Recurrent units adaptively capture dependencies of different time scales in gated recurrent units (GRUs) [69]. Similar to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having separate memory cells. In contrast to LSTM, the GRU exposes the whole state at each timestep [70] and computes a linear sum between the existing state and the newly computed state. The block diagram of a GRU is presented in Figure 11. The activation in a GRU is linearly modeled as

$$\mathbf{h}_t = (1 - z_t)\mathbf{h}_{t-1} + z_t\tilde{\mathbf{h}}_t, \tag{47}$$

where the update gate z_t controls update value of the activation, defined as

$$z_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}), \tag{48}$$

where $\mathbf{W}$ and $\mathbf{U}$ are weight matrices to be learned. The candidate activation is

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1})), \tag{49}$$

where $\mathbf{r}_t$ is a set of rest gates defined as

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}) \tag{50}$$

which allows the unit to forget the previous state by reading the first symbol of an input sequence. Several similarities and differences between GRU networks and LSTM networks are outlined in [69]. The study found that both models performed better than the other only in certain tasks, which suggests there cannot be a suggestion as to which model is better.

G. Memory Networks

Conventional RNNs have small memory size to store features from past inputs [71], [72]. Memory neural networks (MemNN) utilize successful learning methods for inference with a readable and writable memory component. A MemNN is an array of objects and consists of input, response, generalization, and output feature map components [71], [73]. It converts the input to an internal feature representation and then updates the memories based on the new input. Then, it uses the input and the updated memory to compute output features and decode them to produce an output [71]. This networks is not easy to train using BPTT and requires supervision at each layer [74]. A less supervision oriented version of MemNN is end-to-end MemNN, which can be trained end-to-end from input-output pairs [74]. It generates an output after a number of

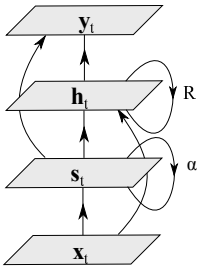


Fig. 12: Recurrent neural network with context features (longer memory).

timesteps and the intermediary steps use memory input/output operations to update the internal state [74].

Recurrent memory networks (RMN) take advantage of the LSTM as well as the MemNN [75]. The memory block in RMN takes the hidden state of the LSTM and compares it to the most recent inputs using an attention mechanism. The RMN algorithm analyses the attention weights of trained model and extracts knowledge from the retained information in the LSTM over time [75]. This model is developed for language modeling and is tested on three large datasets. The results show performance of the algorithm versus LSTM model, however, this model inherits the complexity of LSTM and RMN and needs further development.

Episodic memory is inspired from semantic and episodic memories, which are necessary for complex reasoning in the brain [73]. Episodic memory is named as the memory of the dynamic memory network framework, which remembers autobiographical details [73]. This memory refers to generated representation of stored experiencial facts. The facts are retrieved from the inputs conditioned on the question. This results in a final representation by reasoning on the facts. The module performs several passes over the facts, while focusing on different facts. The output of each pass is called an episode, which is summarized into the memory [73]. A relevant work to MemNN is the dynamic memory networks (DMN). An added memory component to the MemNN can boost its performance in learning long-term dependencies [71]. This approach has shown performance for natural language question answering application [73]. The generalization and output feature map parts of the MemNN have some similar functionalities with the episodic memory in DMS. The MemNN processes sentences independently [73], while the DMS processes sentences via a sequence model [73]. The performance results on the Facebook bAbI dataset show the DMN passes 18 tasks with accuracy of more than 95% while the MemNN passes 16 tasks with lower accuracy [73]. Several steps of Episodic memory are discussed in [73].

H. Structurally Constrained Recurrent Neural Network

Another model which aims to deal with the vanishing gradient problem is the structurally constrained recurrent neural network (SCRN). This network is based on the observation that the hidden states change rapidly during training, as presented in Figure 12, [6]. In this approach, the SCRN structure is extended by adding a specific recurrent matrix equal to identity longer term dependencies. The fully connected recurrent matrix (called hidden layer) produces a set of quickly changing hidden units, while the diagonal matrix (called context layer) supports slow change of the state of the context units [6]. In this way, state of the hidden layer stays static and changes are fed from external inputs. Although this model can prevent gradients of the recurrent matrix vanishing, it is not efficient in training [6]. In this model, for a dictionary of size d, $\mathbf{s}_t$ is the state of the context units defined as

$$\mathbf{s}_t = (1-\alpha)\mathbf{B}\mathbf{x}_t + \alpha\mathbf{s}_{t-1}, \tag{51}$$

where α is the context layer weight, normally set to 0.95, $\mathbf{B}_{d\times s}$ is the context embedding matrix, and $\mathbf{x}_t$ is the input. The hidden layer is defined as

$$\mathbf{h}_t = \sigma(\mathbf{P}\mathbf{s}_t + \mathbf{A}\mathbf{x}_t + \mathbf{R}\mathbf{h}_{t-1}), \tag{52}$$

where $\mathbf{A}_{d\times m}$ is the token embedding matrix, $\mathbf{P}_{p\times m}$ is the connection matrix between hidden and context layers, $\mathbf{R}_{m\times m}$ is the hidden layer $\mathbf{h}_{t-1}$ weights matrix, and $\sigma(\cdot)$ is the "sigmoid" activation function. Finally, the output $\mathbf{y}_t$ is defined as

$$\mathbf{y}_t = f(\mathbf{U}\mathbf{h}_t + \mathbf{V}\mathbf{s}_t), \tag{53}$$

where f is the "softmax" activation function, and $\mathbf{U}$ and $\mathbf{V}$ are the output weight matrices of hidden and context layers, respectively.

Analysis using adaptive context features, where the weights of the context layer are learned for each unit to capture context from different time delays, show that learning of the self-recurrent weights does not seem to be important, as long as one uses also the standard hidden layer in the model. This is while fixing the weights of the context layer to be constant, forces the hidden units to capture information on the same time scale. The SCRN model is evaluated on the Penn Treebank dataset. The presented results in [6] show that the SCRN method has bigger gains compared to the proposed model in [3]. Also, the learning longer memory model claims that it has similar performance, but with less complexity, comparing to the LSTM model [6].

While adding the simple constraint to the matrix results in lower computation compared to its gated counterparts, the model is not efficient in training. The analysis of using adaptive context features, where the weights of the context layer are learned for each unit to capture context from different time delays, shows that learning of the self-recurrent weights does not seem to be important, as long as one uses also the standard hidden layer in the model [6]. Thus, fixing the weights of the context layer to be constant forces the hidden units to capture information on the same time scale.

I. Unitary Recurrent Neural Networks

A simple approach to alleviating the vanishing and exploding gradients problem is to simply use unitary matrices in a RNN. The problem of vanishing or exploding gradients can be attributed to the eigenvalues of the hidden to hidden weight matrix, deviating from one [76]. Thus, to prevent these eigenvalues from deviating, unitary matrices can be used to replace the general matrices in the network.

Unitary matrices are orthogonal matrices in the complex domain [76]. They have absolute eigenvalues of exactly one, which preserves the norm of vector flows and the gradients to propagate through longer timesteps. This leads to preventing vanishing or exploding gradient problems from arising [77]. However, it has been argued that the ability to back propogate gradients without any vanishing could lead to the output being equally dependant on all inputs regardless of the time differences [77]. This also results in the network to waste memory due to storing redundant information.

Unitary RNNs have significant advantages over previous architectures, which have attempted to solve the vanishing gradient problem. An Unitary RNN architecture keeps the internal workings of a vanilla RNN without adding any additional memory requirements. Additionally, by maintaining the same architecture, Unitary RNNs do not noticeably increase the computational cost.

J. Gated Orthogonal Recurrent Unit

Thus far, implementations of RNNs have taken two separate approaches to tackle the issues of exploding and vanishing gradients. The first is implementation of additional gates to improve memory of the system, such as the LSTM and GRU architectures. The second is implementation of unitary matrices to maintain absolute eigenvalues of one.

The gated orthogonal recurrent unit replaces the hidden state loop matrix with an orthogonal matrix and introduces an augmentation of the ReLU activation function, which allows it to handle complex-value inputs [77]. This unit is capable of capturing long term dependencies of the data using unitary matrices, while also leverages forgetting mechanisms present in the GRU structure [77].

K. Hierarchical Subsampling Recurrent Neural Networks

It has been shown that RNNs particularly struggle with learning long sequences. While previous architectures have aimed to change the mechanics of the network to better learn long term dependencies, a simpler solution exists, shortening the sequences using methods such as subsampling. Hierarchical subsampling recurrent neural networks (HSRNNs) aim to better learn large sequences by performing subsampling at each level using a fixed window size [78]. Training this network follows the same process as training a regular RNN, with a few modifications based on the window sizes at each level.

HSRNNs can be extended to multidimensional networks by simply replacing the subsampling windows with multidimensional windows [78]. In multidirectional HSRNNs, each level consists of two recurrent layers scanning in two separate directions with a feedforward layer in between. However, in reducing the size of the sequences, the HSRNN becomes less robust to sequential distortions. This requires a lot of tuning of the network than other RNN models, since the optimal window size varies depending on the task [78]. HSRNNs have shown to be a viable option as a model for learning long sequences due to their lower computational costs when compared to their counterparts. RNNs, regardless of their internal architecture, are activated at each time step of the sequence. This can cause extremely high computational costs for the network to learn information in long sequences [78]. Additionally, information can be widely dispersed in a long sequence, making interdependencies harder to find.

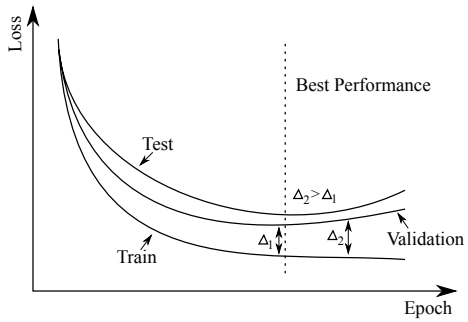


Fig. 13: Overfitting in training neural networks. To avoid overfitting, it is possible to early-stop the training at the "Best Performance" epoch, where the training loss is decreasing but the validation loss starts increasing.

V. Regularizing Recurrent Neural Networks

Regularization refers to controlling the capacity of the neural network by adding or removing information to prevent overfitting. For better training of a RNN, a portion of available data is considered as validation dataset. The validation set is used to watch the training procedure and prevent the network from underfitting and overfitting [79]. Overfitting refers to the gap between the training loss and the validation loss (including the test loss), which increases after a number of training epochs as the training loss decreases, as demonstrated in Figure 13. Successful training of RNNs requires good regularization [80]. This section aims to introduce common regularization methods in training RNNs.

A. L_1 and L_2

The L_1 and L_2 regularization methods add a regularization term to the loss function to penalize certain parameter configuration and prevent the coefficients from fitting so perfectly as to overfit. The loss function in Eq. (8) with added regularization term is

$$\mathcal{L}(\mathbf{y}, \mathbf{z}) = \mathcal{L}(\mathbf{y}, \mathbf{z}) + \eta \left\| \theta \right\|_p^p, \tag{54}$$

where θ is the set of network parameters (weights), η controls the relative importance of the regularization parameter, and

$$\left\| \theta \right\|_p = (\sum_{j=0}^{|\theta|} |\theta_j|^p)^{1/p}. \tag{55}$$

If $p = 1$ the regularizer is L_1 and if $p = 2$ the regularizer is L_2. The L_1 is the sum of the weights and L_2 is the sum of the square of the weights.

B. Dropout

In general, the dropout randomly omits a fraction of the connections between two layers of the network during training. For example, for the hidden layer outputs in Eq. (1) we have

$$\mathbf{h}_t = \mathbf{k} \odot \mathbf{h}_t, \tag{56}$$

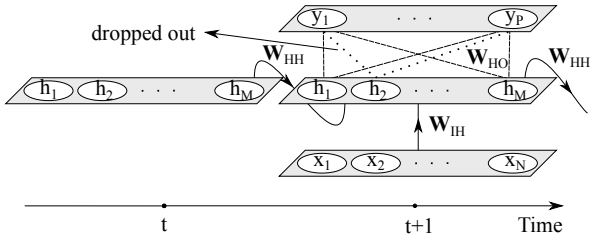


Fig. 14: Dropout applied to feed-forward connections in a RNN. The recurrent connections are shown as full connection with a solid line. The connection between hidden units and output units are shown in dashed lines. The dropped-out connection between the hidden units and output units are shown by dotted lines.

where $\mathbf{k}$ is a binary vector mask and $\odot$ is the element-wise product [81]. The mask can also follow a statistical pattern in applying the withdrawal. During testing, all units are retained and their activations may be weighted.

A dropout specifically tailored to RNNs is introduced in [82], called RNNDrop. This method generates a single dropout mask at the beginning of each training sequence and adjusts it over the duration of the sequence. This allows the network connections to remain constant through time. Other implementations of dropout for RNNs suggest simply dropping the previous hidden state of the network. A similar model to the RNNDrop is introduced in [83], where instead of dropout, it masks data samples at every input sequence per step. This small adjustment has competitive performance to the RNNDrop.

C. Activation Stabilization

Another recently proposed method of regularization involves stabilizing the activations of the RNNs [84]. The norm-stabilizer is an additional cost term to the loss function defined as

$$\mathcal{L}(\mathbf{y},\mathbf{z}) = \mathcal{L}(\mathbf{y},\mathbf{z}) + \beta\frac{1}{T}\sum_{t=1}^{T}(||\mathbf{h}_t||_2 - ||\mathbf{h}_{t-1}||_2)^2 \qquad (57)$$

where $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ are the vectors of the hidden activations at time t and $t-1$, respectively, and β controls the relative importance of the regularization. This additional term stabilizes the norms of the hidden vectors when generalizing long-term sequences.

Other implementations have been made to stabilize the hidden-to-hidden transition matrix such as the use of orthogonal matrices, however, inputs and nonlinearities can still affect the stability of the activation methods. Experiments on language modelling and phoneme recognition show state of the art performance of this approach [84].

D. Hidden Activation Preservation

The zoneout method is a very special case of dropout. It forces some units to keep their activation from the previous timestep (i.e., $\mathbf{h}_t = \mathbf{h}_{t-1}$) [85]. This approach injects stochasticity (by adding noise) into the network, which makes the network more robust to changes in the hidden state and help the network to avoid overfitting. Zoneout uses a Bernoulli mask $\mathbf{k}$ to modify dynamics of $\mathbf{h}_t$ as

$$\mathbf{h}_t = \mathbf{k} \odot \mathbf{h}_t + (1-\mathbf{k}) \odot 1 \qquad (58)$$

which improves the flow of information in the network [85]. Zoneout has slightly better performance than dropout. However, it can also work together with dropout and other regularization methods [85].

VI. Recurrent Neural Networks for Signal Processing

RNNs have various applications in different fields and a large number of research articles are published in that regard. In this section, we review different applications of RNNs in signal processing, particularly text, audio and speech, image, and video processing.

A. Text

RNNs are developed for various application in natural language processing and language modeling. RNNs can outperform *n*-gram models and are widely used as language modelers [86]. However, RNNs are computationally more expensive and challenging to train. A method based on factorization of output layer is proposed in [87], which can speedup the training of a RNN for language modeling up to 100 times. In this approach, words are assigned to specific categories based on their unigram frequency and only words belonging to the predicted class are evaluated in the output layer [86]. HF optimization is used in [5] to train RNNs for character-level language modeling. This model uses gated connections to allow the current input character to determine the transition matrix from one hidden state vector to the next [5]. LSTMs have improved RNN models for language modeling due to their due to their ability to learn long-term dependencies in a sequence better than a simple hidden state [88]. LSTMs are also used in [89] to generate complex text and online handwriting sequences with long-range structure, simply by predicting one data point at a time. RNNs are also used to capture poetic style in works of literature and generate lyrics, for example Rap lyric generation [90]–[92]. A variety of document classification tasks is proposed in the literture using RNNs. In [93], a GRU is adapted to perform document level sentiment analysis. In [94], RCNNs are used for text classification on several datasets. In such approaches, generally the words are mapped to a feature vector and the sequence of feature vectors are passed as input to the RNN model. The same sequence of feature vectors can also be represented as a feature matrix (i.e., an image) to be fed as input to a CNN. CNNs are used in [95] to classify radiology reports. The proposed model is particularly developed for chest pathology and mammogram reports. However, RNNs have not yet been examined for medical reports interpretation and can potentially result in very high classification performance.

B. Speech and Audio

Speech and audio signals continuously vary over time. The inherent sequential and time varying nature of audio signals make RNNs the ideal model to learn features in this field.

Until recently, RNNs had limited contribution in labelling unsegmented speech data, primarily because this task requires pre-segmented data and postprocessing to produce

outputs [96]. Early models in speech recognition, such as time-delay neural networks, often try to make use of the sequential nature of the data by feeding an ANN a set of frames [97]. Given that both past and future sequential information can be of use in speech recognition predictions, the concept of BRNNs were introduced for speech recognition [98]. Later, RNNs were combined with hidden Markov models (HMM) in which the HMM acted as an acoustic model while the RNN acted as the language model [99]. With the introduction of the connectionist temporal classification (CTC) function, RNNs are capable of leveraging sequence learning on unsegmented speech data [96]. Since then, the popularity of RNNs in speech recognition has exploded. Developments in speech recognition then used the CTC function alongside newer recurrent network architectures, which were more robust to vanishing gradients to improve performance and perform recognition on larger vocabularies [100]–[102]. Iterations of the CTC model, such as the sequence transducer and neural transducer [89], [103] have incorporated a second RNN to act as a language model to tackle tasks such as online speech recognition. These augmentations allows the model to make predictions based on not only the linguistic features, but also on the previous transcriptions made.

Speech emotion recognition is very similar to speech recognition, such that a segment of speech must be classified as an emotion. Thus the development of speech emotion recognition followed the same path as that of speech recognition. HMMs were initially used for their wide presence in speech applications [104]. Later, Gaussian mixture models (GMMs) were adapted to the task for their lower training requirements and efficient multi-modal distribution modeling [104]. However these models often require hand crafted and feature engineered input data. Some examples are mel-drequency cepstral coefficients (MFCCs), perceptual linear prediction (PLP) coefficients, and supra-segmental features [105]. With the introduction of RNNs, the trend of input data began to shift from such feature engineering to simply feeding the raw signal as the input, since the networks themselves were able to learn these features on their own. Several RNN models have been introduced since then to perform speech emotion recognition. In [106], an LSTM network is shown to have better performance than support vector machines (SVMs) and conditional random fields (CRFs). This improved performance is attributed to the network's ability to capture emotions by better modeling long-range dependencies. In [107], a deep BLSTM is introduced for speech emotion recognition. Deep BLSTMs are able to capture more information by taking in larger number of frames while a feed-forward DNN simply uses the frame with the highest energy in a sequence [107]. However, comparisons to previous RNNs used for speech emotion recognition were not made. Given that this model used a different model than the LSTM model described prior, no comparisons could be found as to which architecture performs better. Recently, a deep convolutional LSTM is adapted in [105]. This model gives state-of-the-art performance when tested on the RECOLA dataset, as the convolutional layers learns to remove background noise and outline important features in the speech, while the LSTM models the temporal structure of the speech sequence.

Much like speech recognition, speech synthesis also requires long-term sequence learning. HMM-based models can often produce synthesized speech, which does not sound natural. This is due to the overly smooth trajectories produced by the model, as a result of statistical averaging during the training phase [108]. Recent advancementsin neural networks have shown that deep MLP neural networks can synthesize speech. However, these models take each frame as an independent entity from its neighbours and fail to take into account the sequential nature of speech [108]. RNNs were first used for speech synthesis to leverage these sequential dependencies [109], [110], and were then replaced with LSTM models to better learn long term sequential dependencies [111]. The BLSTM has been shown to perform very well in speech synthesis due to the ability to integrate the relationship with neighbouring frames in both future and past time steps [112], [113]. CNNs have shown to perform better than state of the art LSTM models, in particular the WaveNet model [114]. WaveNet is a newly introduced CNN capable of generating speech, using dilated convolutions. Through the use of dilated causal convolutions, WaveNet can model long-range temporal dependencies by increasing it's receptive field of input. WaveNet has shown better performance than LSTMs and HMMs [114].

The modelling of polyphonic music presents another task with inherent contextual dependencies. In [115], a RNN combined with a restricted Boltzmann machine (RBM) is introduced, which is capable of modeling temporal information in a music track. This model has a sequence of conditional RBMs, which are fed as parameters to a RNN, so that can learn harmonic and rhythmic probability rules from polyphonic music of varying complexity [115]. It has been shown that RNN models struggle to keep track of distant events that indicate the temporal structure of music [116]. LSTM models have since been adapted in music generation to better learn the long-term temporal structure of certain genres of music [116], [117].

C. Image

Learning the spatial dependencies is generally the main focus in machine vision. While CNNs have dominated most applications in computer vision and image processing, RNNs have also shown promising results such as image labeling, image modeling, and handwriting recognition.

Scene labeling refers to the task of associating every pixel in an image to a class. This inherently involves the classification of a pixel to be associated with the class of its neighbour pixels. However, models such as CNNs have not been completely successful in using these underlying dependencies in their model. These dependencies have been shown to be leveraged in numerous implementations of RNNs. A set of images are represented as undirected cyclic graphs (UCGs) in [118]. To feed these images into a RNN, the UCGs are decomposed into several directed acyclic graphs (DAGs) meant to approximate the original images. Each DAG image involves a convolutional layer to produce discriminative feature mapping, a DAG-RNN to model the contextual dependencies between pixels,

and a deconvolutional layer to up-sample the feature map to its original image size. This implementation has better performance than other state of the art models on popular datasets such as SiftFlow, CamVid, and Barcelona [118]. A similar implementation is shown in [49], where instead of decomposing the image into several DAGs, the image is first fed into a CNN to extract features for a local patch, which is then fed to a 2D-RNN. This 2D-RNN is similar to a simple RNN, except for its ability to store hidden states in two dimensions. The two hidden neurons flow in different directions towards the same neuron to create the hidden memory. To encode the entire image, multiple starting points are chosen to create the multiple hidden states of the same pixel. This architecture is developed further by introducing 2D-LSTM units to better retain long-term information [119].

Image modeling is the task of assigning a probability distribution to an image. RNNs are naturally the best choice for image modeling tasks given its inherent ability to be used as a generative model. The deep recurrent attentive writer (DRAW) combines a novel spatial attention mechanism that mimics the foveation of the human eye, with a sequential variational auto-encoding framework that allows iterative construction of complex images [120]. While most image generative models aim to generate scenes all at once, this causes all pixels to be modelled on a single latent distribution. The DRAW model generates images through first generating sections of the scene independently of each other before going through iterations of refinement. The recent introduction of PixelRNN, involving LSTMs and BLSTMs, has shown improvements in modelling natural images with scalability [121]. The PixelRNN uses up to 12 2-dimensional LSTM layers, each of which has an input-to-state component and a recurrent state-to-state component. These component then determine the gates inside each LSTM. To compute these states, masked convolutions are used to gather states along one of the dimensions of the image. This model has better log-likelihood scores than other state of the art models evaluated on MNIST and CIFAR-10 datasets. While PixelRNN has shown to perform better than DRAW on the MNIST dataset, there has been no comparison between the two models as to why this might be.

The task of handwriting recognition combines both image processing and sequence learning. This task can be divided into two types, online and offline recognition. RNNs perform well on this task, given the contextual dependencies in letter sequences [122]. For the task of online handwriting recognition, the position of the pen-tip is recorded at intervals and these positions are mapped to the sequence of words [122]. In [122], a BLSTM model is introduced for online handwriting recognition. It's performance is better than conventionally used HMM models due to its ability to make use of information in both past and future time steps. BLSTM perform well when combined with a probabilistic language model and trained with CTC. For offline handwriting recognition, only the image of the handwriting is available. To tackle this problem, an MDLSTM is used to convert the 2-dimensional inputs into a 1-dimensional sequence [52]. The data is then passed through a hierarchy of MDLSTMs, which incrementally decrease the size of the data. While such tasks are often implemented using CNNs, it is argued that due to the absence of recurrent connections in such networks, CNNs cannot be used for cursive handwriting recognition without first being pre-segmented [52]. The MDLSTM model proposed in [52] offers a simple solution which does not need segmented input and can learn the long term temporal dependencies.

Recurrent generative networks are developed in [123] to automatically recover images from compressed linear measurements. In this model, a novel proximal learning framework is developed, which adopts ResNets to model the proximals and a mixture of pixel-wise and perceptual costs are used for training. The deep convolutional generative adversarial networks are developed in [124] to generate artificial chest radiographs for automated abnormality detection in chest radiographs. This model can be extended to medical image modalities which have spatial and temporal dependencies, such as head magnetic resonance imaging (MRI), using RCNNs. Since RNNs can model non-linear dynamical systems, recent RNN architectures can potentially enhance performance of these models.

D. Video

A video is a sequence of images (i.e., frames) with temporal and spatial dependencies between frames and pixels in each frame, respectively. A video file has far more pixels in comparison to a single image, which results in a greater number of parameters and computational cost to process it. While different tasks have been performed on videos using RNNs, they are most prevalent in video description generation. This application involves components of both image processing and natural language processing. The method proposed in [125] combines a CNN for visual feature extraction with an LSTM model capable of decoding the features into a natural language string known as long-term recurrent convolutional networks (LRCNs). However, this model was not an end-to-end solution and required supervised intermediate representations of the features generated by the CNN. This model is built upon in [126], which introduces a solution capable of being trained end-to-end. This model utilizes an LSTM model, which directly connects to a deep CNN. This model was further improved in [127], in which a 3-dimensional convolutional architecture was introduced for feature extraction. These features were then fed to an LSTM model based on a soft-attention mechanism to dynamically control the flow of information from multiple video frames. RNNs has fewer advances in video processing, comparing with the other types of signals, which introduces new opportunities for temporal spatial machine learning.

VII. Conclusion and Potential directions

In this paper, we systematically review major and recent advancements of RNNs in the literature and introduce the challenging problems in training RNNs. RNN refer to a network of artificial neurons with recurrent connections among the units. The recurrent connections learn the dependencies among input sequential or time-series data. The ability to learn sequential dependencies has allowed RNNs to gain popularity

in applications such as speech recognition, speech synthesis, machine vision, and video description generation.

One of the main challenges is training RNNs is learning long-term dependencies in data. It occurs generally due to the large number of parameters that need to be optimized during training in RNN over long periods of time. This paper discusses several architectures and training methods that have been developed to tackle the problems associated with training of RNNs. The followings are some major opportunities and challenges in developing RNNs:

- The introduction of BPTT algorithm has facilitated efficient training of RNNs. However, this approaches introduces gradient vanishing and explodin problems. Recent advances in RNNs have since aimed at tackling this issue. However, these challenges are still the main bottleneck of training RNNs.
- Gating mechanisms have been a breakthrough in allowing RNNs to learn long-term sequential dependencies. Architectures such as the LSTM and GRU have shown significantly high performance in a variety of applications. However, these architecture introduce higher complexity and computation than simple RNNs. Reducing the internal complexity of these architectures can help reduce training time for the network.
- The unitary RNN has potentially solved the above issue by introducing a simple architecture capable of learning long-term dependencies. By replacing the internal weights with unitary matrices, the architecture keeps the same complexity of a simple RNN while providing stronger modeling ability. Further research into the use of unitary RNNs can help in validating its performance against its gated RNN counterparts.
- Several regularization methods such as dropout, activation stabilization, and activation preservation have been adapted for RNNs to avoid overfitting. While these methods have shown to improve performance, there is no standard for regularizing RNNs. Further research into RNNs regularization can help introduce potentially better regularization methods.
- RNNs have a great potential to learn features from 3-dimensional medical images, such as head MRI scans, lung computed tomography (CT), and abdominal MRI. In such modalities, the temporal dependency between images is very important, particularly for cancer detection and segmentation.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[3] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8624–8628.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[5] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[6] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, "Learning longer memory in recurrent neural networks," *arXiv preprint arXiv:1412.7753*, 2014.

[7] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[8] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

[9] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[10] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] I. Sutskever, "Training recurrent neural networks," *University of Toronto, Toronto, Ont., Canada*, 2013.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[14] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[15] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks," *IEEE Transactions on neural networks*, vol. 5, no. 2, pp. 279–297, 1994.

[16] S. Ma and C. Ji, "A unified approach on fast training of feedforward and recurrent networks using em algorithm," *IEEE transactions on signal processing*, vol. 46, no. 8, pp. 2270–2274, 1998.

[17] L.-W. Chan and C.-C. Szeto, "Training recurrent network with block-diagonal approximated levenberg-marquardt algorithm," in *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, vol. 3. IEEE, 1999, pp. 1521–1526.

[18] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[20] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[21] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[22] J. A. Pérez-Ortiz, F. A. Gers, D. Eck, and J. Schmidhuber, "Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets," *Neural Networks*, vol. 16, no. 2, pp. 241–250, 2003.

[23] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky, "Subword language modeling with neural networks," *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[24] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model." *SLT*, vol. 12, pp. 234–239, 2012.

[25] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[26] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[27] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods," in *Advances in neural information processing systems*, 2011, pp. 1647–1655.

[28] J. Martens and I. Sutskever, "Training deep and recurrent networks with hessian-free optimization," *Neural networks: Tricks of the trade*, pp. 479–535, 2012.

[29] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[30] L. Bottou, "Stochastic learning," in *Advanced lectures on machine learning*. Springer, 2004, pp. 146–168.

[31] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] S. S. Haykin *et al.*, *Kalman filtering and neural networks*. Wiley Online Library, 2001.

[33] R. J. Williams, "Training recurrent networks using the extended kalman filter," in *Neural Networks, 1992. IJCNN., International Joint Conference on*, vol. 4. IEEE, 1992, pp. 241–246.

[34] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 735–742.

[35] D. T. Mirikitani and N. Nikolaev, "Recursive bayesian recurrent neural networks for time-series modeling," *IEEE Transactions on Neural Networks*, vol. 21, no. 2, pp. 262–274, 2010.

[36] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1033–1040.

[37] H. Salehinejad, S. Rahnamayan, and H. R. Tizhoosh, "Micro-differential evolution: Diversity enhancement and a comparative study," *Applied Soft Computing*, vol. 52, pp. 812–833, 2017.

[38] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[39] K. Unnikrishnan and K. P. Venugopal, "Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks," *Neural Computation*, vol. 6, no. 3, pp. 469–490, 1994.

[40] C. Smith and Y. Jin, "Evolutionary multi-objective generation of recurrent neural network ensembles for time series prediction," *Neurocomputing*, vol. 143, pp. 302–311, 2014.

[41] T. Tanaka, T. Moriya, T. Shinozaki, S. Watanabe, T. Hori, and K. Duh, "Evolutionary optimization of long short-term memory neural network language model," *The Journal of the Acoustical Society of America*, vol. 140, no. 4, pp. 3062–3062, 2016.

[42] H. Salehinejad, S. Rahnamayan, H. R. Tizhoosh, and S. Y. Chen, "Micro-differential evolution with vectorized random mutation factor," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 2055–2062.

[43] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[44] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[45] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[46] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[47] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3367–3375.

[48] P. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *International Conference on Machine Learning*, 2014, pp. 82–90.

[49] B. Shuai, Z. Zuo, and G. Wang, "Quaddirectional 2d-recurrent neural networks for image labeling," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1990–1994, 2015.

[50] P. Baldi and G. Pollastri, "The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem," *Journal of Machine Learning Research*, vol. 4, no. Sep, pp. 575–602, 2003.

[51] A. Graves, S. Fernández, and J. Schmidhuber, "Multi-dimensional recurrent neural networks," 2007. [Online]. Available: http://arxiv.org/abs/0705.2011

[52] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, 2009, pp. 545–552.

[53] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio, "Renet: A recurrent neural network based alternative to convolutional networks," *arXiv preprint arXiv:1505.00393*, 2015.

[54] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[55] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[56] X. Zhu, P. Sobihani, and H. Guo, "Long short-term memory over recursive structures," in *International Conference on Machine Learning*, 2015, pp. 1604–1612.

[57] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," *arXiv preprint arXiv:1507.01526*, 2015.

[58] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated lstm," *arXiv preprint arXiv:1508.03790*, 2015.

[59] M. Ballesteros, C. Dyer, and N. A. Smith, "Improved transition-based parsing by modeling characters instead of words with lstms," *arXiv preprint arXiv:1508.00657*, 2015.

[60] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4277–4280.

[61] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[62] S. Wang and J. Jiang, "Learning natural language inference with lstm," *arXiv preprint arXiv:1512.08849*, 2015.

[63] A. Graves, S. Fernandez, and J. Schmidhuber, "Multi-dimensional recurrent neural networks," *arXiv preprint arXiv:0705.2011v1*, 2007.

[64] L. Theis and M. Bethge, "Generative image modeling using spatial lstms," in *Advances in Neural Information Processing Systems*, 2015, pp. 1918–1926.

[65] V. Veeriah, N. Zhuang, and G.-J. Qi, "Differential recurrent neural networks for action recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4041–4049.

[66] X. Liang, X. Shen, D. Xiang, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with local-global long short-term memory," *arXiv preprint arXiv:1511.04510*, 2015.

[67] J. Li, A. Mohamed, G. Zweig, and Y. Gong, "Lstm time and frequency recurrence for automatic speech recognition," 2015.

[68] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *arXiv preprint arXiv:1506.04214*, 2015.

[69] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[70] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[71] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv preprint arXiv:1410.3916*, 2014.

[72] J. Weston, A. Bordes, S. Chopra, and T. Mikolov, "Towards ai-complete question answering: a set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[73] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[74] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2431–2439.

[75] K. Tran, A. Bisazza, and C. Monz, "Recurrent memory network for language modeling," *arXiv preprint arXiv:1601.01272*, 2016.

[76] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *International Conference on Machine Learning*, 2016, pp. 1120–1128.

[77] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljačić, and Y. Bengio, "Gated orthogonal recurrent units: On learning to forget," *arXiv preprint arXiv:1706.02761*, 2017.

[78] A. Graves, *Supervised sequence labelling with recurrent neural networks*. Springer Science & Business Media, 2012, vol. 385.

[79] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[80] N. Srivastava, "Improving neural networks with dropout," *University of Toronto*, vol. 182, 2013.

[81] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014, pp. 285–290.

[82] T. Moon, H. Choi, H. Lee, and I. Song, "Rnndrop: A novel dropout for rnns in asr," in *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 2015, pp. 65–70.

[83] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," *arXiv preprint arXiv:1603.05118*, 2016.

[84] D. Krueger and R. Memisevic, "Regularizing rnns by stabilizing activations," *arXiv preprint arXiv:1511.08400*, 2015.

[85] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, H. Larochelle, A. Courville *et al.*, "Zoneout:

Regularizing rnns by randomly preserving hidden activations," *arXiv preprint arXiv:1606.01305*, 2016.

[86] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černockỳ, "Empirical evaluation and combination of advanced language modeling techniques," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[87] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5528–5531.

[88] M. Sundermeyer, R. Schl"uter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[89] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[90] X. Zhang and M. Lapata, "Chinese poetry generation with recurrent neural networks." in *EMNLP*, 2014, pp. 670– 680.

[91] P. Potash, A. Romanov, and A. Rumshisky, "Ghostwriter: using an lstm for automatic rap lyric generation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1919– 1924.

[92] M. Ghazvininejad, X. Shi, Y. Choi, and K. Knight, "Generating topical poetry." in *EMNLP*, 2016, pp. 1183– 1191.

[93] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification." in *EMNLP*, 2015, pp. 1422– 1432.

[94] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification." in *AAAI*, vol. 333, 2015, pp. 2267–2273.

[95] H. Salehinejad, J. Barfett, S. Valaee, E. Colak, A. Mnatzakanian, and T. Dowdell, "Interpretation of mammogram and chest radiograph reports using deep neural networks-preliminary results," *arXiv preprint arXiv:1708.09254*, 2017.

[96] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[97] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[98] M. Schuster, "Bi-directional recurrent neural networks for speech recognition," Technical report, Tech. Rep., 1996.

[99] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Springer Science & Business Media, 2012, vol. 247.

[100] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1764–1772.

[101] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[102] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4945–4949.

[103] N. Jaitly, D. Sussillo, Q. V. Le, O. Vinyals, I. Sutskever, and S. Bengio, "A neural transducer," *arXiv preprint arXiv:1511.04868*, 2015.

[104] M. El Ayadi, M. S. Kamel, and F. Karray, "Survey on speech emotion recognition: Features, classification schemes, and databases," *Pattern Recognition*, vol. 44, no. 3, pp. 572–587, 2011.

[105] G. Trigeorgis, F. Ringeval, R. Brueckner, E. Marchi, M. A. Nicolaou, B. Schuller, and S. Zafeiriou, "Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5200–5204.

[106] M. Wöllmer, F. Eyben, S. Reiter, B. Schuller, C. Cox, E. Douglas-Cowie, and R. Cowie, "Abandoning emotion classes-towards continuous emotion recognition with modelling of long-range dependencies," in *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[107] J. Lee and I. Tashev, "High-level feature representation using recurrent neural network for speech emotion recognition." in *INTERSPEECH*, 2015, pp. 1537–1540.

[108] K. Fan, Z. Wang, J. Beck, J. Kwok, and K. A. Heller, "Fast second order stochastic backpropagation for variational inference," in *Advances in Neural Information Processing Systems*, 2015, pp. 1387–1395.

[109] O. Karaali, G. Corrigan, I. Gerson, and N. Massey, "Text-to-speech conversion with neural networks: A recurrent tdnn approach," *arXiv preprint cs/9811032*, 1998.

[110] C. Tuerk and T. Robinson, "Speech synthesis using artificial neural networks trained on cepstral coefficients." in *EUROSPEECH*, 1993.

[111] H. Zen and H. Sak, "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4470–4474.

[112] Y. Fan, Y. Qian, F.-L. Xie, and F. K. Soong, "Tts synthesis with bidirectional lstm based recurrent neural networks," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[113] R. Fernandez, A. Rendel, B. Ramabhadran, and R. Hoory, "Prosody contour prediction with long short-term memory, bi-directional, deep recurrent neural networks." in *Interspeech*, 2014, pp. 2268–2272.

[114] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[115] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[116] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, vol. 103, 2002.

[117] ——, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*. IEEE, 2002, pp. 747–756.

[118] B. Shuai, Z. Zuo, B. Wang, and G. Wang, "Dag-recurrent neural networks for scene labeling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3620–3629.

[119] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3547–3555.

[120] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[121] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[122] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2008, pp. 577–584.

[123] M. Mardani, H. Monajemi, V. Papyan, S. Vasanawala, D. Donoho, and J. Pauly, "Recurrent generative adversarial networks for proximal learning and automated compressive image recovery," *arXiv preprint arXiv:1711.10046*, 2017.

[124] H. Salehinejad, S. Valaee, T. Dowdell, E. Colak, and J. Barfett, "Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks," *arXiv preprint arXiv:1712.01636*, 2017.

[125] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[126] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating videos to natural language using deep recurrent neural networks," *arXiv preprint arXiv:1412.4729*, 2014.

[127] L. Yao, A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle, and A. Courville, "Video description generation incorporating spatio-temporal features and a soft-attention mechanism," *arXiv preprint arXiv:1502.08029*, 2015.