

A Survey of Recursive and Recurrent Neural Networks

Jian-wei Liu^{a,*}, Bing-rong Xu^a, Zhi-yan Song^a

^a*Department of Automation, College of Artificial Intelligence, China University of Petroleum, Beijing, Beijing, China*

Abstract

Recursive and recurrent neural networks are the main realization forms of sequence models based on neural networks, which have been developed rapidly in recent years. Recurrent neural networks are basically standard processing methods for machine translation, question and answer, and sequence video analysis. It is also the mainstream modeling method for handwriting automatic synthesis, speech processing and image generation. In this paper, the branches of recursive and recurrent neural networks are classified in detail according to the network structure, training objective function and learning algorithm implementation. They are roughly divided into three categories: The first category is General Recursive and Recurrent Neural Networks, including Basic Recursive and Recurrent Neural Networks, Long Short Term Memory Recursive and Recurrent Neural Networks, Convolutional Recursive and Recurrent Neural Networks, Differential Recursive and Recurrent Neural Networks, One-Layer Recursive and Recurrent Neural Networks, High-Order Recursive and Recurrent Neural Networks, Highway Networks, Multidimensional Recursive and Recurrent Neural Networks, Bidirectional Recursive and Recurrent Neural Networks; the second category is Structured Recursive and Recurrent Neural Networks, including Grid Recursive and Recurrent Neural Networks, Graph Recursive and Recurrent Neural Networks, Temporal Recursive and Recurrent Neural Networks, Lattice Recursive and Recurrent Neural Networks, Hierarchical Recursive and Recurrent Neural Networks, Tree Recursive and Recurrent Neural Networks; the third category is Other Recursive and Recurrent Neural Networks, including Array Long Short Term Memory, Nested and Stacked Recursive and Recurrent Neural Networks, Memory Recursive and Recurrent Neural Networks. Various networks cross each other and even rely on each other to form a complex network of relationships. In the context of the development and convergence of various networks, many complex sequence, speech and image problems are solved. After a detailed description of the principle and structure of the above model and model deformation, the research progress and application of each model are described, and finally the recursive and recurrent neural network models are prospected and summarized.

Keywords: Recursive Neural Networks, Recurrent Neural Networks, General Recursive and Recurrent Neural Networks, Structured Recursive and Recurrent Neural Networks

*Corresponding author

Email address: liujw@cup.edu.cn (Jian-wei Liu)

1. Introduction

Recursive Neural Networks (RecursiveNNs) and Recurrent Neural Networks (RecurrentNNs) are two types of artificial neural networks (ANNs) structures. ANNs were originally developed as a mathematical model to simulate the information processing process of the biological brain. Although it is now clear that ANNs have almost no similarities with real biological neurons, ANNs have been widely used as pattern classifiers. The basic structure of ANNs is a network composed of single neurons or nodes. After neurons are layered, they are connected by connecting edges with weights to represent the interaction between neurons.

There are many types of ANNs, and their characteristics vary greatly. Generally, ANNs can be divided into recurrent connected ANNs and non-recurrent connected ANNs, that is, a cyclic graph structure and acyclic graph structure. Neural networks with recurrent connections are called RecurrentNNs or RecursiveNNs.

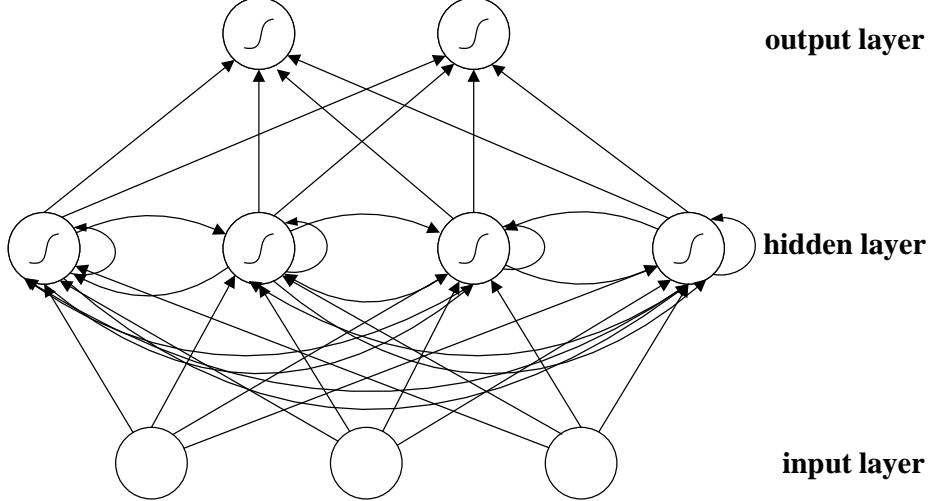


Figure 1: RNNs structure.

In this paper, RecursiveNNs and RecurrentNNs are collectively referred to as RNNs. Because RNNs are inspired by the periodic connections of neurons in the brain, they use recurrent iteration functions to store information. RNNs have several advantages: RNNs can well capture context information and realize temporal dependency learning (because RNNs can learn what to store and what to ignore); RNNs can learn different types of data representation; RNNs can represent sequences (such as words) order well. However, RNNs also have some shortcomings, which limit their wide application in practical problems. The most serious problem is that it is difficult to model large-scale temporal dependencies. Therefore, in order to solve more complex problems, many structural variants based on RNNs have appeared. The classification of RNNs summarized in this paper is shown in Fig. 2.

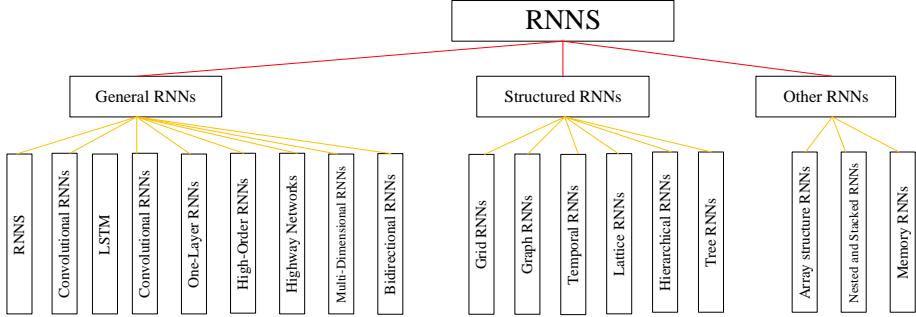


Figure 2: RNNs classification.

2. General RNNs

There are many types of General RNNs, but the most basic ones are RecursiveNNs and RecurrentNNs, so that the subsequent networks are all ordinary variants of it. It is generally applied to a series of problems such as handwriting recognition, speech recognition, natural language processing, and computer vision; Convolutional RNNs have good effects in image and audio, and are mainly used in the re-recognition or tracking process of characters or the representation of audio features learning and audio labeling tasks; LSTM is a relatively typical and mature structure among general RNNs structural variants, which can be applied to multiple phonemes, images and other issues; Differential RNNs learn different information through different orders for video processing and action recognition; One-layer RNNs are used to solve pseudo-convex optimization problems with equality and inequality constraints; High-order RNNs allow high-order interactions of neurons with strong storage capabilities, and are used in grammatical reasoning and target detection; the highway connection of the Highway Network allows information to cross different layers unimpeded, thereby alleviating the problem of gradient disappearance when building a deeper LSTM; Multi-dimensional RNNs replace the single recurrent link of RNNs with the spatio-temporal dimension in the data in dimension to solve local distortion; Bidirectional RNNs can use two separate hidden layers to process data in both directions to simultaneously utilize the previous context and the future context. The following is a detailed introduction to the above RNNs.

2.1. RecursiveNNs and RecurrentNNs

Next, the various network architectures of RecursiveNNs and RecurrentNNs will be described in detail.

2.1.1. RecurrentNNs

RecurrentNNs is an extension of traditional feedforward neural networks that can handle variable-length sequence inputs. It learns the hidden representation of the variable-length input sequence

through the internal recurrent hidden variables. The output of the activation function of the hidden variable at each time step depends on the output of the recurrent hidden variable activation function at the previous time step [1].

Given an input sequence $x = (x_1, x_2, \dots, x_T)$, the recurrent update process of RecurrentNNs hidden variables is as follows:

$$h_t = \begin{cases} 0, & t = 0 \\ \phi(h_{t-1}, x_t), & \text{otherwise} \end{cases} \quad (1)$$

where ϕ is a nonlinear activation function (such as the logistics sigmoid activation function pre-activated by the affine transformation). The output $y = (y_1, y_2, \dots, y_T)$ of RecurrentNNs may also be of variable length. So in general, the implementation of the hidden variable update process in Eq.(1) is as follows:

$$h_t = g(Wx_t + Uh_{t-1}) \quad (2)$$

where g is a smooth bounded function (such as logistics sigmoid function or hyperbolic tangent function). W is the weight matrix of the hidden variable input to this time step, and U is the weight matrix of the hidden variable from the previous time step to this time step.

Given the current state h_t , RecurrentNNs can be used to represent the joint probability distribution on the input sequence, that is, to explain the update process of RecurrentNNs from the viewpoint of a generative model: the update formula at each time step generates a conditional probability distribution, the product of the conditional probability distribution at all time steps to obtain the joint probability distribution, and RecurrentNNs can naturally represent the probability distribution on the variable-length sequence, and a special termination symbol can be introduced to detect the end position of the variable-length sequence. According to the Bayesian formula, the chain rule, the entire joint probability distribution of a random variable sequence $x = (x_1, x_2, \dots, x_T)$ with sequence length T can be expressed as:

$$p(x_1, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_T|x_1, \dots, x_{T-1}) \quad (3)$$

The last element is a special sequence termination symbol, and each conditional probability distribution in the sequence joint probability distribution is parameterized into a neural network form to obtain:

$$p(x_t|x_1, \dots, x_{t-1}) = g(h_t) \quad (4)$$

where h_t is from Eq.(1).

2.1.2. RecursiveNNs

RecursiveNNs operate on structured input, not just for sequence structures. RecursiveNNs have been applied to grammatical analysis, sentence-level sentiment analysis, paraphrase detection, morphology oriented word vector representation, machine question answering and political ideology detection. The essence of RecursiveNNs is the process of recurrently constructing a larger subtree by multiple subtrees of the same structure.

The architecture of RecursiveNNs [2] is: RecursiveNNs recursively apply the same set of weights along the length of the sequence. Given a location directed acyclic graph, it visits nodes in topological order and applies transformations recursively in the previously calculated sub representation to generate further representation. In fact, RecurrentNNs are just RecursiveNNs with a specific structure.

Given a binary tree structure with fixed leaf nodes, for example, a parse tree with word vectors on the leaves (see Fig. 3), RecursiveNNs calculates the representation of the internal node η as follows:

$$x_h = \sigma(W_L x_{l(h)} + W_R x_{r(h)} + b) \quad (5)$$

where $l(\eta)$ and $r(\eta)$ are the left and right child nodes of η , W_L and W_R are the weight matrices connecting the left and right child nodes to the parent node, and b is the bias vector. Assuming that W_L and W_R are square matrices, and do not distinguish whether $l(\eta)$ and $r(\eta)$ are leaves or internal nodes, there is an interesting explanation in this case: the initial representation of the leaf node and the non-terminal intermediate representation are in the same space. In the parse tree example, RecursiveNNs combines two sub-phrases to generate a longer phrase in the same space. Therefore, there is a final output layer at root ρ :

$$y = \gamma(Ux_\rho + c) \quad (6)$$

where U is the output weight matrix and c is the bias vector of the output layer. In supervised learning tasks, supervision occurs at this layer. Therefore, in the learning process, y will produce an initial error, and it will propagate back from the root to the leaf nodes.

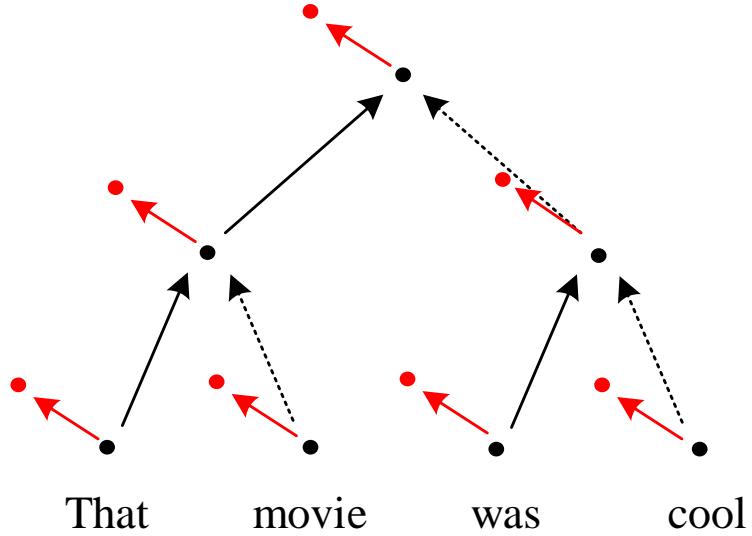


Figure 3: Recursive neural network with bottom-up propagation.

2.1.3. The Difference between RecursiveNNs and RecurrentNNs

RecurrentNNs are seen as unfolding along time, processing information of sequence structure. For example, Fig. 4 is a RecurrentNNs, which expands over time [3]. At each timestep, the input of the hidden layer includes the user's input and the output of the hidden layer at the previous timestep.

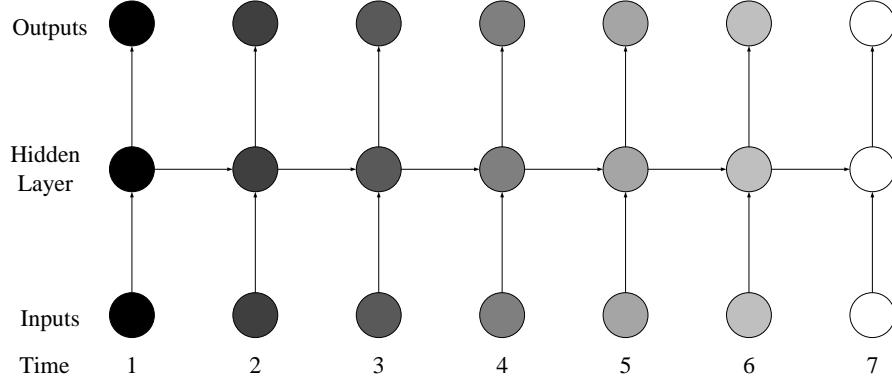


Figure 4: Unfold along time.

RecurrentNNs is a special case of RecursiveNNs. RecursiveNNs is more like a hierarchical network. In this network, the input sequence is actually not related to time, but the input must be processed hierarchically in the form of a tree. When RecurrentNNs encounters a sequence with a different

length from the sequence seen during training during the test, it cannot handle the position-related weights, so the weights are shared along the sequence length (the dimensionality remains the same). In RecursiveNNs, for the same reason, the weights on each node are also shared (the dimensionality remains the same).

Fig. 5 shows what a RecursiveNNs looks like:

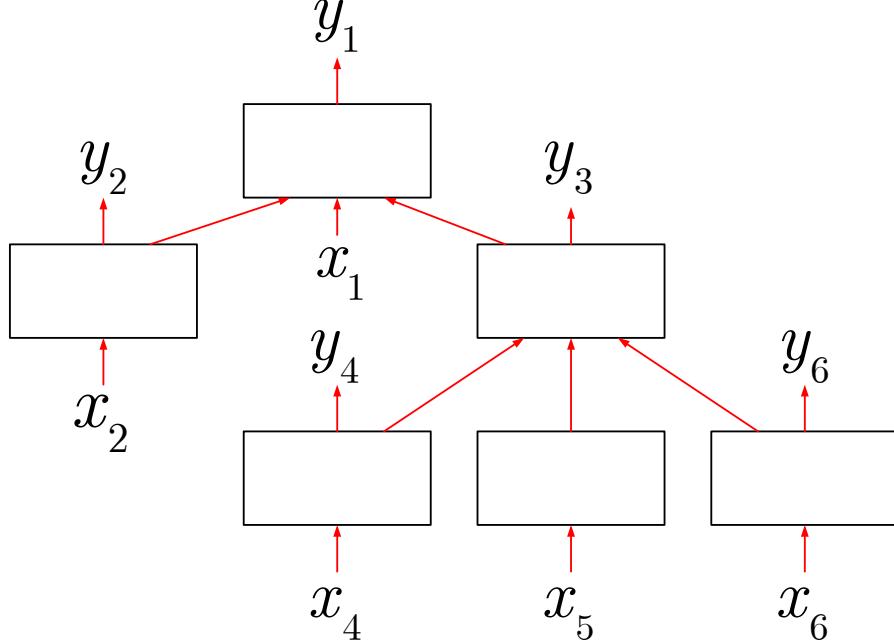


Figure 5: RecursiveNNs.

According to Fig. 5, it is easy to understand why it is called RecursiveNNs. The child node of each parent node is just a node similar to that node. Different neural networks are suitable for different scenarios. If you want to generate characters one by one, then a RecurrentNNs is very suitable. But if you want to generate a parse tree, it is better to use a RecursiveNNs because it helps to create a better hierarchical representation.

2.2. LSTM

Next, the various network architectures of LSTM will be described in detail.

2.2.1. Difficulties in Training RecurrentNNs

1) Training recurrent networks

Although RecurrentNNs is a simple and powerful model in theory, it is difficult to get good training in practice. One of the main reasons is the gradient disappearance and gradient explosion problem

described in [4] [5]. This question is discussed below.

At time t , assuming the input variable x_t and the state variable h_t , RecurrentNNs can be expressed as:

$$h_t = F(h_{t-1}, x_t, \theta) \quad (7)$$

The above formula can be parameterized into the following form:

$$h_t = W_{rec}\sigma(h_{t-1}) + W_{in}x_t + b \quad (8)$$

where W_{rec} is the recurrent weight matrix, b is the bias vector, and W_{in} is the input weight matrix. Let $\theta = \{W_{rec}, W_{in}, b\}$ denote the parameter set of RecurrentNNs. The initial value h_0 is given by the user, or another learning algorithm is used to learn the initial value of h_0 on the data. σ is the sigmoid function acting on the components of the pre-activation function.

One of the methods to calculate the gradient is the Back Propagation Through Time (BPTT), in which the recurrent model is represented by a multi-layer model (with an infinite number of layers), and the back propagation process is applied to the expanded model (see Fig. 6).

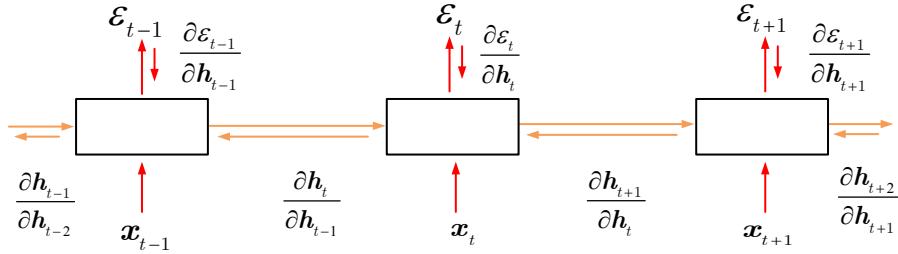


Figure 6: Unrolling recurrent neural networks.

By creating a copy of the model for each time step, the recurrent neural network is unfolded in time. Use h_t to represent the hidden state of the network at time t , x_t to represent the input of the network at time t , and ε_t to represent the error of the network output at time t .

Tomas Mikolov et al. deviated from the classic BPTT equation and rewritten the gradient to better highlight the explosion gradient problem:

$$\frac{\partial \varepsilon}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \varepsilon_t}{\partial \theta} \quad (9)$$

$$\frac{\partial \varepsilon_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \varepsilon_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial \theta} \right) \quad (10)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} W_{rec}^T diag(\sigma'(h_{i-1})) \quad (11)$$

These equations are obtained by writing the gradient as a sum of products. $\frac{\partial^+ h_k}{\partial \theta}$ is the immediate partial derivative of state h_k with respect to θ , and h_{k-1} is a constant with respect to θ . According to Eq.(8), the value of any i -th row of matrix $\left(\frac{\partial^+ h_k}{\partial W_{rec}} \right)$ is $\sigma(h_{k-1})$. Eq.(11) also provides a Jacobian matrix $\frac{\partial h_i}{\partial h_{i-1}}$ for the specific parameters in Eq.(8), where *diag* transforms the vector into a diagonal matrix, and σ' is the derivative of σ . Any gradient component $\frac{\partial \varepsilon_t}{\partial \theta}$ is also obtained by summation (see Eq.(10)), and its term is called temporal contribution or temporal component. It can be seen that each temporal contribution $\frac{\partial \varepsilon_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial \theta}$ can measure the influence of θ at step k on the cost at step $t > k$. The factor $\frac{\partial h_t}{\partial h_k}$ (Eq.(11)) can transmit the error transmitted from step t back to step k in time. Tomas Mikolov et al. further distinguish between long-term and short-term contributions. The long-term is defined as the temporal component at $k \ll t$ and the short-term refers to the remaining part.

2)Exploding and vanishing gradients

In the introduction of Bengio et al. [6], the gradient explosion problem refers to a substantial increase in the gradient norm during training. This is caused by the explosion of long-term components, which grow exponentially faster than short-term components. The vanishing gradient problem refers to the opposite process. When the exponentially increasing long-term component quickly approaches the norm 0, the model cannot learn the dependence between a wide range of events. The following analyzes the problem of gradient disappearance and explosion from a structural perspective.

Tomas Mikolov et al. extended the similar derivation of Bengio et al. and considered the case of a single hidden unit. If the linear form of the model is considered (i.e. σ is set to the identity function in equation(8)), the power iteration method can be used to analyze the product of the Jacobian matrix and obtain the strict conditions when the gradient explodes or disappears. Define ρ as the spectral radius of the recurrent weight matrix W_{rec} . When $\rho < 1$, the gradient will not disappear or explode, and when $\rho > 1$, the gradient will disappear or explode. This result is then extended to a nonlinear function σ through singular values, where $|\sigma'(x)|$ is bounded, $\|diag(\sigma'(x_k))\| \leq \gamma \in R$. First prove that when $\lambda_1 < \frac{1}{\gamma}$, λ_1 is the largest singular value of W_{rec} , and the disappearing gradient problem will occur. By reversing this proof, the necessary condition for gradient explosion can be obtained, that is, the maximum singular value λ_1 is greater than $\frac{1}{\gamma}$ (otherwise the long-term component will disappear instead of exploding). Let $\gamma = 1$ for the *tanh* function and $\gamma = \frac{1}{4}$ for the *sigmoid* function.

2.2.2. Long Short-Term Memory Unit

LSTM was proposed in [7] [8], and then improved and promoted in [9]. On many issues, LSTM has achieved considerable success and has been widely used. Each j -th unit of LSTM contains a memory unit c_t^j at time step t , then the output h_t^j (or activation) of the LSTM unit is expressed as [1]:

$$h_t^j = o_t^j \tanh(c_t^j) \quad (12)$$

where o_t^j is the output gate, which adjusts the amount of output content of the memory unit. The calculation process of the output gate is as follows:

$$o_t^j = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t)^j \quad (13)$$

where σ is a logistic sigmoid function, and V_o is a diagonal matrix.

Update the memory unit c_t^j by partially forgetting the existing memory and then adding a new memory unit \tilde{c}_t^j :

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \quad (14)$$

where the new memory unit is:

$$\tilde{c}_t^j = \tanh(W_c x_t + U_c h_{t-1})^j \quad (15)$$

The function of the forgetting gate f_t^j is to adjust the degree of forgetting of the existing memory, and the degree of adding content to the new memory unit is adjusted by the input gate i_t^j . The calculation process of the forget gate and input gate is as follows:

$$f_t^j = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1})^j \quad (16)$$

$$i_t^j = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1})^j \quad (17)$$

Note that V_f and V_i are diagonal matrices.

Unlike the traditional recurrent unit that covers content at each time step (see Eq.()), the LSTM unit can decide whether to keep the existing memory by introducing a gate. Intuitively speaking, if the LSTM unit can detect important features from the input sequence at an early stage, it can easily carry this information (the existence of features) over long distances to capture potential long-distance dependencies.

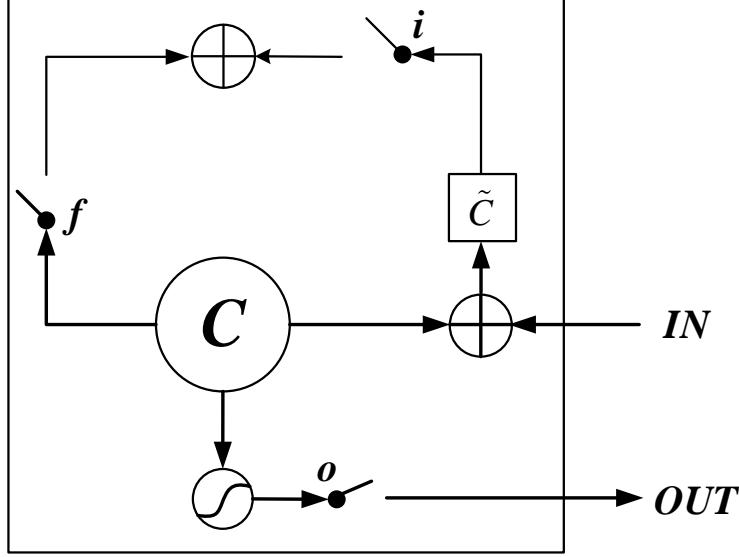


Figure 7: Long Short-Term Memory.

i , f and o are input gate, forget gate and output gate respectively, c and \tilde{c} represent memory unit and new memory unit.

2.2.3. Gated Recurrent Unit

Cho et al. [10] proposed a gated recurrent unit (GRU), which enables each recurrent unit to adaptively capture the dependencies of different time scales. Similar to the LSTM unit, the GRU has a gating unit to adjust the information flow inside the unit, but there is no separate memory unit.

The activation h_t^j of the GRU at time step t is the linear interpolation between the previously activated h_{t-1}^j and the pre-activated \tilde{h}_t^j :

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \quad (18)$$

where the update gate z_t^j determines the extent to which the unit updates its activation or content. The calculation method of the update gate is as follows:

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \quad (19)$$

The process of taking a linear sum between the existing state and the new calculation state is similar to the LSTM unit. However, GRU does not have any mechanism to control the extent of its state exposure, but exposes the entire state every time.

The calculation method of pre-activation \tilde{h}_t^j is similar to RecurrentNNs (see Eq.(2)):

$$\tilde{h}_t^j = \tanh(Wx_t + U(r_t \odot h_{t-1}))^j \quad (20)$$

where r_t is a set of reset gates and \odot is an element-wise multiplication. When r_t^j is close to 0, the reset gate effectively makes the unit work like reading the first symbol of the input sequence, making it forget the previously calculated state.

The calculation method of reset gate r_t^j is similar to update gate:

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})^j \quad (21)$$

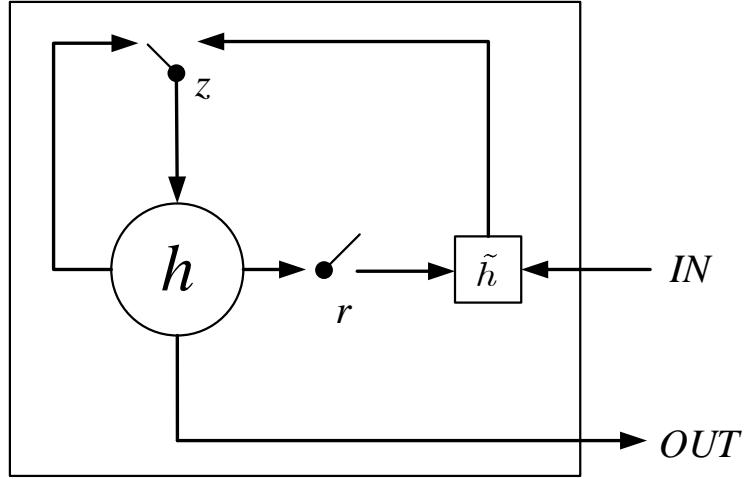


Figure 8: GRU.

2.2.4. The Depth-gated LSTM

Depth-Gated LSTM (DGLSTM) is shown in Fig. 9 [11]. It has a depth gate, which connects the memory cell $c_t^{(L+1)}$ in the lower layer $L+1$ and the memory cell c_t^L in the upper layer L . The depth gate controls the flow from the lower memory unit directly to the upper memory unit. The gate function at level $L+1$ at time step t is a logistic function as follows:

$$d_t^{(L+1)} = \sigma(b_d^{(L+1)} + W_{xd}^{(L+1)} x_t^{(L+1)} + w_{cd}^{(L+1)} \odot c_{t-1}^{(L+1)} + w_{ld}^{(L+1)} \odot c_t^{(L)}) \quad (22)$$

where $b_d^{(L+1)}$ is the bias, and $W_{xd}^{(L+1)}$ is the weight matrix associated with the depth gate and the input of this layer. The past memory unit generates correlation through the weight vector $w_{cd}^{(L+1)}$. Use the weight vector $w_{ld}^{(L+1)}$ to associate the lower memory. Note that if the lower and upper memory cells have different dimensions, $w_{ld}^{(L+1)}$ should be a matrix instead of a vector.

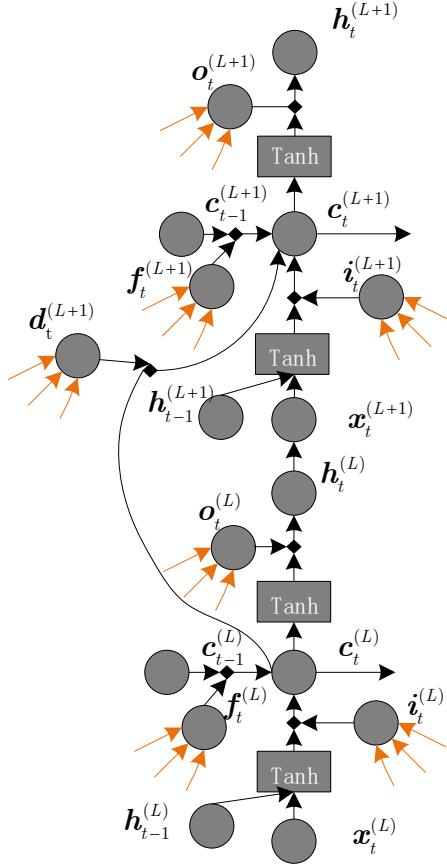


Figure 9: Schematic diagram of DGLSTM. Note the additional connections between the memory cells in the lower and upper layers in the DGLSTM.

Using the depth gate, DGLSTM calculates the memory cell process at level $L + 1$ as follows:

$$c_t^{(L+1)} = d_t^{(L+1)} \odot c_t^{(L)} + f_t^{(L+1)} \odot c_{t-1}^{(L+1)} + i_t^{(L+1)} \odot \tanh(W_{xc}^{(L+1)} x_t^{(L+1)} + W_{hc}^{(L+1)} h_{t-1}^{(L+1)}) \quad (23)$$

In DGLSTM, the general LSTM structure ((12),(13),(16),(17)) is used, but DGLSTM uses superscript $L + 1$ to represent the operation at layer $L + 1$.

The idea of gated linear correlation can also be used to connect the first layer memory unit $c_t^{(1)}$ with the feature observation vector $x_t^{(0)}$. In this case, when $L = 0$, the depth gate calculation process is as follows:

$$d_t^{(1)} = \sigma(b_d^{(1)} + W_{xd}^{(1)} x_t^{(1)} + w_{cd}^{(1)} \odot c_{t-1}^{(1)}) \quad (24)$$

and the memory cell is computed as:

$$c_i^{(1)} = d_t^{(1)} \odot (W_{xd}^{(1)} x_t^{(0)}) + f_t^{(1)} \odot c_{t-1}^{(1)} + i_t^{(1)} \odot \tanh(W_{xc}^{(1)} x_t^{(0)} + W_{hc}^{(1)} h_{t-1}^{(1)}) \quad (25)$$

2.2.5. Structure-evolving LSTM

Structure-evolving LSTM [12] [13] consists of five gates: input gate g^u , forget gate g^f , adaptive forget gate \bar{g}^f , memory gate g^c , output gate g^o and edge gate p . 1 is the indicator function. W^e represents the weight parameter of the recurrent edge gate. W^u, W^f, W^c, W^o are the recurrent gate weight matrices specified for the input features, and U^u, U^f, U^c, U^o are the recurrent gate weight matrices of the hidden state of each node. $U^{un}, U^{fn}, U^{cn}, U^{on}$ are the weight parameters specified for the state of neighboring nodes. The structure-evolving LSTM unit assigns different forgetting gates to different neighboring nodes through the effect of the input state of the current node and its hidden state, which are defined as $\bar{g}_{ij}^f, j \in \mathcal{N}(i)_{G^{(t)}}(i)$. Adjacent nodes have different effects on the updated memory state c_i^{t+1} and hidden state h_j^{t+1} . Use the weight matrix W^e to weight the adaptive forgetting gate \bar{g}_{ij}^f to calculate the merging probability p_{ij} of each pair of graph nodes $< i, j > \in \varepsilon^t$. Intuitively, the adaptive forgetting gate is used to identify the significant correlations of different node pairs. Therefore, an adaptive forget gate is used to estimate the merging probability of each pair of graphs. The new hidden state, memory state, and edge gate (i.e., the merging probability of each pair of connected nodes) in graph $G^{(t)}$ can be calculated as follows:

$$g_i^u = \delta(W^u x_i^t + U^u h_i^{t-1} + U^{un} \bar{h}_i^{t-1} + b^u)$$

$$\bar{g}_{ij}^f = \delta(W^f x_i^t + U^{fn} h_j^{t-1} + b^f)$$

$$g_i^f = \delta(W^f x_i^t + U^f h_i^{t-1} + b^f)$$

$$g_i^o = \delta(W^o x_i^t + U^o h_i^{t-1} + U^{on} \bar{h}_i^{t-1} + b^o)$$

$$g_i^c = \tanh(W^c x_i^t + U^c h_i^{t-1} + U^{cn} \bar{h}_i^{t-1} + b^c)$$

$$c_{i,t} = \frac{\sum_{j \in \mathcal{N}(i)_g(i)} (1(q_j = 1) \bar{g}_{ij}^f \odot c_j^t + 1(q_j = 0) \bar{g}_{ij}^f \odot c_j^{t-1})}{|\mathcal{N}(i)_{G^{(t)}}(i)|} + g_i^f \odot c_i^{t-1} + g_i^u \odot g_i^c$$

$$h_i^t = \tanh(g_i^o \odot c_i^t)$$

$$p_{ij}^t = \delta(W^e \bar{g}_{ij}^f) \quad (26)$$

where σ is the logistic sigmoid function, and \odot represents the point-wise product. q_j is a visit flag, used to indicate whether the graph node v_i has been updated, where q_j is set to 1 if it is updated, otherwise it is set to 0. $\mathcal{N}(i)_{G^{(t)}}(i)$ represents the number of adjacent graph nodes of node i . As a memory system, this model writes information into the memory state and records it in sequence by each graph node, and then uses it to communicate with subsequent graph nodes and the hidden state of the previous LSTM layer. Therefore, the merge probability $\{p_{ij}\}, \langle i, j \rangle \in \varepsilon^t$ can be effectively learned and applied in the $t+1$ -th layer to generate a new high-level graph structure $G^{(t+1)}$. During training, the merging probability of graph edges is supervised by approximating the final graph structure of a specific task, such as the connection of the final semantic region of image parsing. Use the back propagation method to train all weight matrices.

The advantages of this model include: (1) Multi-level graph representation of data and LSTM network parameters can be learned simultaneously in an end-to-end manner. (2) Realize that the underlying multilayer graph structure evolves with parameter learning.

Structure-evolving LSTM is also called graph LSTM, which can be classified in graph RNNs, so it won't be repeated in Chapter 3.2.

2.2.6. Layer Trajectory LSTM

Jinyu Li et al. proposed layer trajectory LSTM (ltLSTM) [14] [15]. Its output can satisfy both temporal modeling and senone classification. It uses all output layers of general LSTM to construct a layer LSTM (L-LSTM), as shown in Fig. 10. The weights are not shared between layers because sharing doesn't reduce computational complexity. The general LSTM is used for the purpose of temporal modeling via temporal recurrence, while the L-LSTM scans multiple outputs of general LSTM layers and then uses the summarized layer trajectory information to perform the final senone classification. The L-LSTM layer provides a gated path from the output layer to the bottom layer during operation, so the problem of gradient disappearance is reduced.

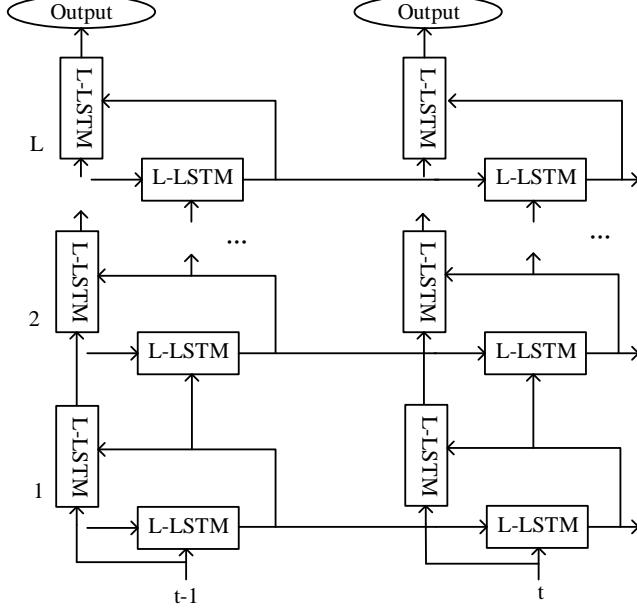


Figure 10: ItLSTM flowchart.

L-LSTM is used to scan the general LSTM output of all layers at the current time step to obtain summary layer trajectory information for senone classification. L-LSTM has no time recurrence, it only exists between general LSTMs at different time steps.

In ItLSTM, the general LSTM formula is still as shown in section 2.2.2. As shown in Fig. 10, there is no time recurrence between L-LSTMs at different time steps. Therefore, the formula of L-LSTM only has a cross-layer recurrence:

$$j_t^l = \sigma(U_{jh}^l h_t^l + U_{jg}^l g_t^{l-1} + q_j^l \odot c_t^{l-1} + d_j^l)$$

$$e_t^l = \sigma(U_{eh}^l h_t^l + U_{eg}^l g_t^{l-1} + q_e^l \odot c_t^{l-1} + d_e^l)$$

$$c_t^l = e_t^l \odot c_t^{l-1} + j_t^l \odot \phi(U_{sh}^l h_t^l + U_{sg}^l g_t^{l-1} + d_s^l)$$

$$v_t^l = \sigma(U_{vh}^l h_t^l + U_{vg}^l g_t^{l-1} + q_v^l \odot c_t^l + d_v^l)$$

$$g_t^l = v_t^l \odot \phi(c_t^l) \quad (27)$$

The vectors j_t^l , v_t^l , e_t^l , and m_t^l are the input gate, output gate, forget gate and memory unit of L-LSTM, respectively. g_t^l is the output of L-LSTM. The matrices U_{jh}^l and U_{jg}^l are the weight matrices

of input h_t^l and output g_t^l , respectively. d_t^l is the bias vector. q_j^l , q_v^l , q_e^l are vector parameters related to peephole connection.

The g_t^{l-1} -layer recurrence of L-LSTM in LtLSTM is the biggest difference from general LSTM. L-LSTM takes the general LSTM output h_t^l of the current layer as input, which is in contrast with the x_t^l in the general LSTM.

Advantages of LtLSTM: Since the forward propagation of the general LSTM at the next time step has nothing to do with the calculation of the L-LSTM layer at the current time step, the forward propagation of the general LSTM and L-LSTM can process two separate threads in parallel [16], and the network calculation time runs layer by layer and frame by frame. Another advantage of decoupling temporal and layer operations in LtLSTM is that batch processing [17] can be used to evaluate L-LSTM, which can improve the running time of feedforward deep neural networks by simultaneously evaluating the network scores of multiple time frames. But this batch processing cannot be applied to general LSTM.

2.2.7. Multiplicative LSTM

Since LSTM and multiplicative RNN [18] (mRNN) structures are complementary, Krause et al. proposed multiplicative LSTM (mLSTM) [19], which is a hybrid structure that combines the factorization of mRNN hidden variable to hidden variable conversion and LSTM gated framework. The structure of mRNN and LSTM can add connections from the intermediate state m_t of mRNN to each gate unit in LSTM, resulting in the following system:

$$\begin{aligned} m_t &= (W_{mx}x_t) \odot (W_{mh}h_{t-1}) \\ \hat{h}_t &= W_{hx}x_t + W_{hm}m_t \\ i_t &= \sigma(W_{ix}x_t + W_{im}m_t) \\ o_t &= \sigma(W_{ox}x_t + W_{om}m_t) \\ f_t &= \sigma(W_{fx}x_t + W_{fm}m_t) \end{aligned} \tag{28}$$

The goal of this structure is to transform the flexible input dependency of mRNN and combine it with the long time lag and information control of LSTM. The gated unit of LSTM can more easily control (or bypass) the complex conversion of the factorization hidden variable weight matrix. The additional sigmoid input and forget gate in the LSTM unit exhibit a more flexible input-dependent conversion function than the regular mRNN.

The advantages of this model include: (1) mLSTM does not use nonlinear recurrent depth, so it is easier to parallelize. (2) For each possible input, there is a different recurrent transfer function, so it can better express the autoregressive density estimation.

There are also many variants of LSTM, which are interspersed in different types of RNNs, so they won't be repeated in this section 2.2.

2.3. Convolutional RNNs

Next, the various network architectures of the Convolutional RNNs will be described in detail.

2.3.1. Recurrent Convolutional Network

In the recurrent convolutional network, each frame of the input video sequence is first processed by a convolutional neural network (CNN) to generate a feature vector representing the appearance of a person at a specific time step. Then, before using the temporal pooling to combine the output of all time steps, the recurrent layer is used to let the information flow between each time step. The temporal pooling can make the network use a feature vector to represent an arbitrarily long video sequence, and the recurrent layer can make the network make better use of the time information in the sequence. In order to train the feature extraction network for re-recognition, McLaughlin et al. used the Siamese network [20]. Therefore, the video re-identification method based on recurrent convolutional network [21] has higher accuracy. The structure of the recurrent convolutional network is detailed below. Let $s = s^{(1)}, \dots, s^{(T)}$ be a video sequence of length T , which consists of full-body images of a person, where $s^{(t)}$ is the image at time step t . Each image $s^{(t)}$ generates a vector $f^{(t)} = C(s^{(t)})$ through CNN, where $f^{(t)}$ is the vector representation of the activation map of the CNN output layer. Then the vector $f^{(t)}$ is passed to the recurrent layer, where it is projected to a low-dimensional feature space and combined with the information at the previous moment. One issue to note is that the parameters of CNN are shared at all time steps, which means that each input frame is processed by the same feature extraction network. Therefore, dropout is used between the CNN and the recurrent layer to reduce overfitting. Fig. 11 shows the complete details of the CNN and recurrent layer structure. Among them, $r^{(t)}$ is state variable of RNNs at time step t , and $o^{(t)}$ is the sequence output vector at time step t .

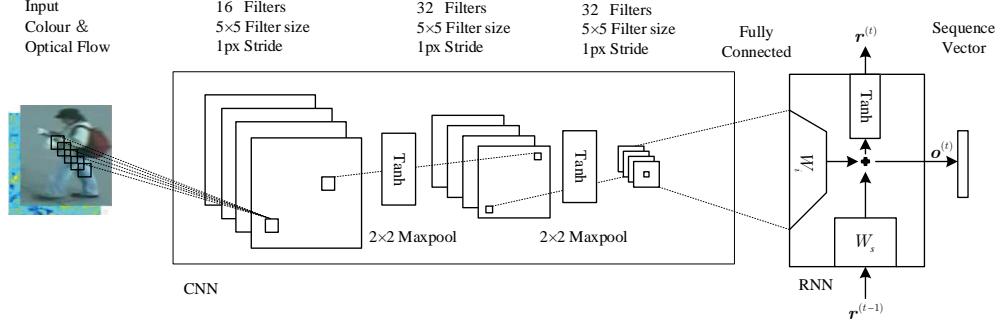


Figure 11: The structure of the proposed CNN and recurrent layer.

Join the recurrent connection after the CNN, as shown below:

$$o^{(t)} = W_i f^{(t)} + W_s r^{(t-1)} \quad (29)$$

$$r^{(t)} = \text{Tanh}(o^{(t)}) \quad (30)$$

The output $o^{(t)} \in \mathbb{R}^{e \times 1}$ is a linear combination of vectors at each time step, $f^{(t)} \in \mathbb{R}^{N \times 1}$ contains the information of the current input image, and $r^{(t-1)} \in \mathbb{R}^{e \times 1}$ contains the state information of the RNNs at the previous time step. Use the fully connected layers $W_i \in \mathbb{R}^{e \times N}$ and $W_s \in \mathbb{R}^{e \times e}$ to calculate the output, e is the dimension of the feature embedding space, and N is the dimension of the activation mapping vector output of the final layer of CNN. Note that the parameter matrix W_i is a non-square matrix, which means that the final layer activation map of the CNN is projected into a vector in the lower-dimensional feature embedding space. The state variable $r^{(t)}$ of RNNs is initialized to a zero vector at the first time step $r^{(0)}$, and is transferred between each time step through the Tanh nonlinear function. Add a temporal pool after the recurrent layer, so that information can be aggregated at any time step, long-term information existing in the sequence can be captured, and the feature vector representation of the sequence of any length can be learned.

2.3.2. Spatially Supervised Recurrent Convolutional Neural Networks

The human and object tracking process based on spatially supervised recurrent convolutional neural network is shown in Fig. 12 [22]. Guanghan Ning et al. use the YOLO [23] [24] (a framework for target detection problems) system to collect visual features and perform preliminary position inference; then use LSTM in the next stage. LSTM can capture a large range of spatial dependencies and is suitable for human and sequence processing of the object tracking process. This model belongs to a deep neural network. It takes the original video frame as input and returns the coordinates of the bounding box of

the tracked object in each frame. This model formalizes the complete tracking conditional probability distribution as:

$$p(B_1, B_2, \dots, B_T | X_1, X_2, \dots, X_T) = \prod_{t=1}^T p(B_t | B_{<t}, X_{\leq t}) \quad (31)$$

where B_t and X_t are the target and the position of the input frame at time step t , respectively. $B_{<t}$ is the history information of all previous positions before time step t , and $X_{\leq t}$ is the history information of the input frame up to time step t . The spatially supervised recurrent convolutional neural network can effectively process spatiotemporal information and infer the location of the region. The YOLO deep convolutional neural network is extended to the spatiotemporal domain by using the RecurrentNNs.

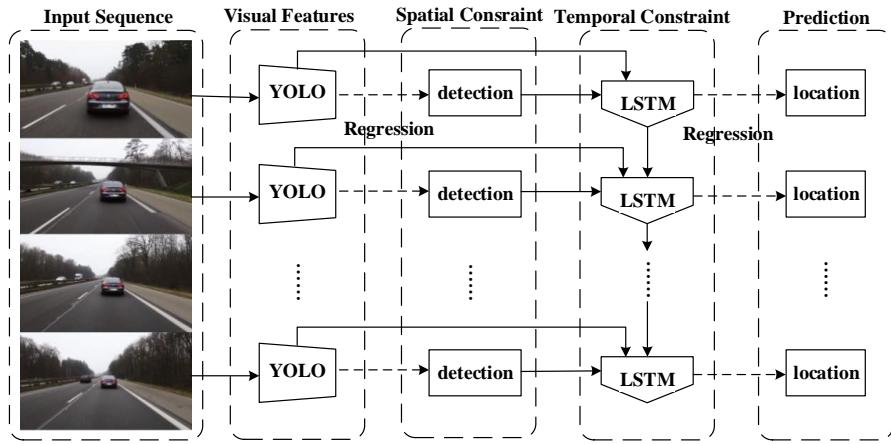


Figure 12: Simplified system overview and tracking procedures.

The advantages of this model include: (1) A modular neural network is introduced, which can be trained end-to-end through a gradient-based learning method. (2) Compared with the existing trackers based on convolutional networks, the framework proposed by Guanghan Ning et al. extends the neural network analysis to the spatio-temporal domain, which can achieve effective visual object tracking. (3) The proposed network is accurate and efficient, with low complexity.

2.3.3. Convolutional Gated Recurrent Neural Network

Fig. 13 shows the framework of a convolutional gated recurrent neural network applied to audio tagging [25] [26] [27] [28]. CNN is a feature extractor that can extract robust features against background noise, especially the original audio waveform, through a maximum pooling operation. Then, the extracted robust features are fed into Gated Recurrent Unit-RNN (GRU-RNN) to learn long-term audio mode. GRU-RNN can select relevant information from the long-term context of each audio event. In

order to use the information of the future, this system has designed a bidirectional GRU-RNN. Finally, under the condition of the sigmoid output activation function, the output of GRU-RNN is mapped to the posterior probability of the target audio event through a feedforward neural layer. The framework is flexible enough to be applied to any type of feature, especially raw audio waveforms. Since the original waveform has many values, it will cause high-dimensional problems. However, the proposed CNN can learn under short windows such as the Short-Time Fourier Transform (STFT) process, and similar Fast Fourier Transform (FFT) basis vector set or mel-like filters can also be used for learning the original waveform. Finally, a single-layer feedforward deep neural network (DNN) obtains the final GRU sequence to predict the posterior probability of the tag.

Yong Xu et al. chose binary cross entropy as the loss function, which is more effective than the mean square error [29]. The loss function is defined as follows:

$$E = - \sum_{n=1}^N \| T_n \log \hat{T}_n + (1 - T_n) \log (1 - \hat{T}_n) \| \quad (32)$$

$$\hat{T}_n = (1 + \exp(-O))^{-1} \quad (33)$$

where E is the binary cross entropy, \hat{T}_n and T_n represent the estimated value and reference tag vector at sample index n , respectively. Before using the sigmoid activation function, the linear output of DNN is defined as O .

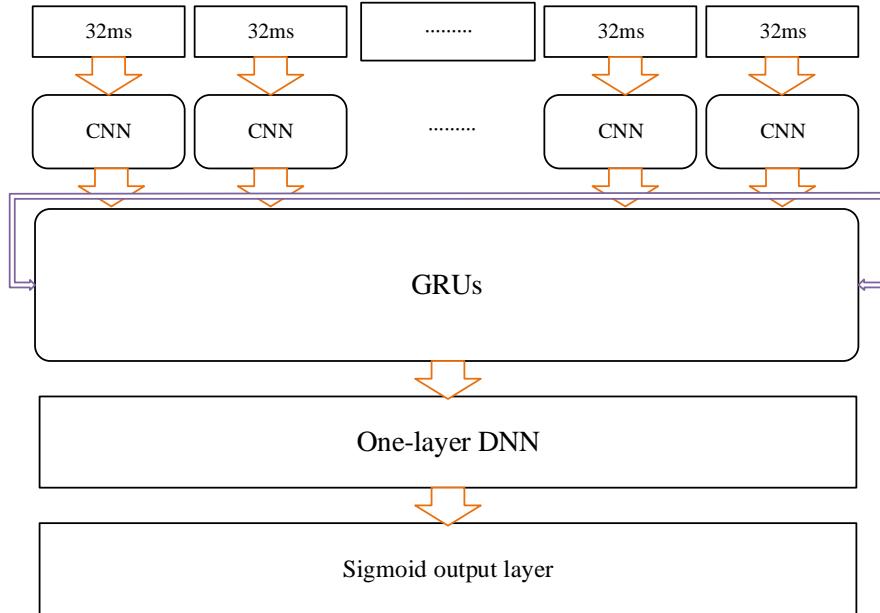


Figure 13: The framework of convolutional gated recurrent neural network for audio tagging.

2.3.4. Long-term Recurrent Convolutional Network (LRCN)

Long-term Recurrent Convolutional Network (LRCN) [30] [31] combines a deep hierarchical visual feature extractor (such as CNN) with a specific model that can recognize and synthesize sequential data (input or output), visual, language or other aspects of the temporal dynamic characteristics. Fig. 14 depicts the core of this model. The LRCN model generates a fixed-length vector representation $\phi_t \in \mathbb{R}^d$ by passing each visual input v_t (isolated image or video frame) to a feature transformation $\phi_V(v_t)$ parameterized by V (visual feature extractor parameters). After calculating the feature space representation of the visual input sequence $\langle \phi_1, \phi_2, \dots, \phi_T \rangle$, the sequence model can be substituted.

In the most general form, the sequence model parameterized by W (sequence model parameter) maps the input x_t and the hidden state h_{t-1} at the previous time step to the output z_t and updates the hidden state h_t . Therefore, the inference must be run sequentially (that is, run from top to bottom in the sequence learning box in Fig. 14), through the sequential calculation: $h_1 = f_W(x_1, h_0) = f_W(x_1, 0)$, $h_2 = f_W(x_2, h_1)$ and so on, until h_T .

The last step of predicting the distribution $P(y_t)$ at time step t is to perform softmax (normalized exponential function) processing on the output z_t of the sequence model, and generate the possible output distribution at each time step in the (in this example, finite and discrete) space C :

$$P(y_t = c) = \frac{\exp(W_{zc} z_{t,c} + b_c)}{\sum_{c' \in C} \exp(W_{zc'} z_{t,c'} + b_c)} \quad (34)$$

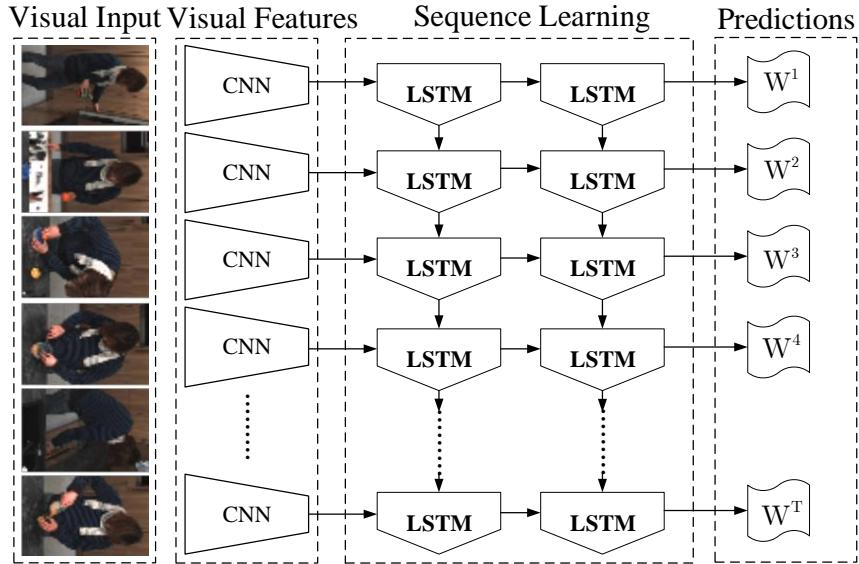


Figure 14: LRCN diagram. LRCN uses CNN (middle left) to process (possibly) variable-length visual input (left), and its output is fed to a bunch of recurrent sequence models (LSTM, middle right), which finally produces variable-length predictions (right).

The advantages of this model include: (1) The model can be flexibly applied to various visual tasks involving sequential input and output. (2) Suitable for large-scale visual learning, with end-to-end trainability. (3) It has advantages when the target concept is complex or/and training data is limited. (4) Time dynamics and convolutional perception representations can be learned at the same time.

2.3.5. Multimodal CNN-LSTM Structure

Xinyu Li et al. [32] introduce a system based on the multimodal CNN-LSTM structure, as shown in Fig. 15. The system can identify concurrent activities [33] [34] from real data captured by multiple sensors of different types. The recognition is done in two steps. First, extract spatial and temporal features from multimodal data, and input each data type into a CNN that extracts spatial features, and then an LSTM that extracts temporal information from sensor data. The extracted features are fused in the second step. Second, use a single classifier to realize concurrent activity recognition. This system is the first to use a single model to solve concurrent activity recognition of multi-sensory data. The model is scalable, easy to train and easy to deploy. This activity recognition system consists of four main modules connected in series: sensory data preprocessing, spatial feature extraction, temporal correlation and fusion, and coding layer. The multimodal CNN-LSTM structure is responsible for feature extraction and temporal association extraction in the system.

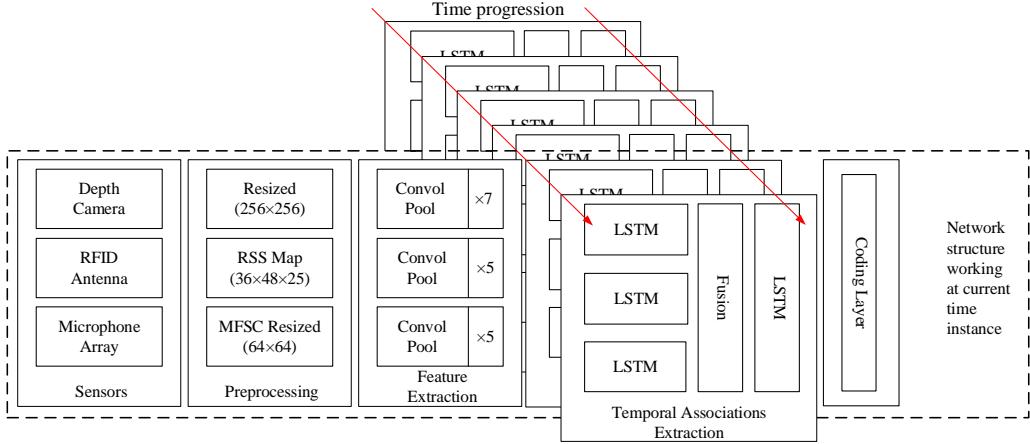


Figure 15: The structure of the concurrent activity recognition system running at a given time instance. The stacked part represents the evolution of the LSTM memory over time.

2.3.6. Diffusion Convolutional Recurrent Neural Network

Yaguang Li et al. introduced Diffusion Convolutional Recurrent Neural Network (DCRNN) [35], which is a deep learning framework for traffic prediction [36] [37], which combines the spatial and temporal dependencies of traffic flow. Specifically, DCRNN uses a bidirectional random walk on the graph to capture spatial dependence, and uses an encoder-decoder architecture with scheduled sampling

to capture temporal dependence. Through space and time modeling, Yaguang Li et al. established DCRNN. The model architecture of DCRNN is shown in Fig. 16. The entire network trains the entire network by maximizing the likelihood of generating the target future time series using backpropagation. DCRNN can capture the spatio-temporal dependence between time series and can be applied to various spatio-temporal prediction problems.

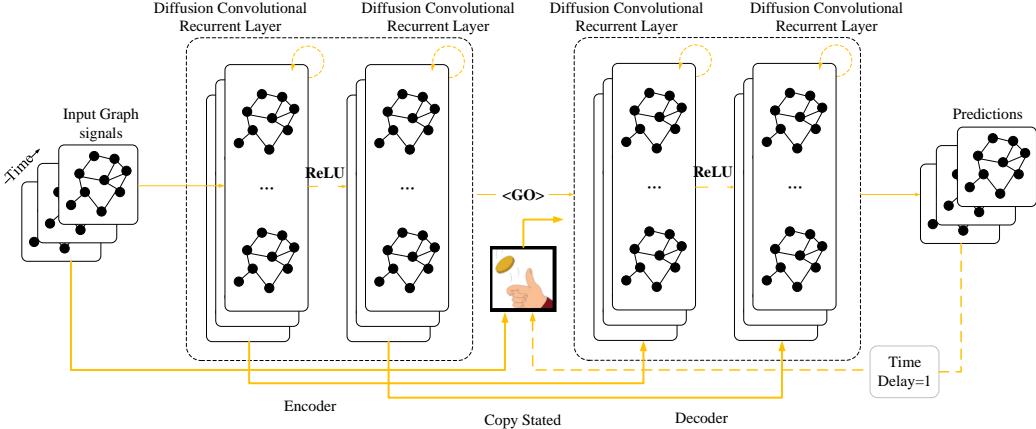


Figure 16: The system structure of the DCRNN for temporal and spatial traffic prediction. The historical time sequence is fed into the encoder, and its final state is used to initialize the decoder. The decoder makes predictions based on previous ground truth or model output.

2.4. Differential RNNs

Next, the various network architectures of Differential RNNs will be described in detail.

2.4.1. Differential RNN

For action recognition tasks [38] [39], not all video frames contain significant patterns to distinguish different types of actions. Many spatio-temporal models, such as 3D-SIFT [40] and HoGHoF [41], have been used to locate and encode significant spatiotemporal points. They detect and encode the spatio-temporal points in the video frame that are related to the significant action of the object, revealing important dynamic characteristics of the action.

Veeriah et al. proposed a new LSTM model [42], which automatically learns the dynamic characteristics of actions by detecting and integrating significant spatio-temporal sequences. Traditional LSTM may not be able to capture these remarkable dynamic patterns, because the gated unit does not explicitly consider the influence of the dynamic structure in the input sequence, which makes it difficult for the model to learn the evolution of behavioral states. To solve this problem, Veeriah et al. proposed a differential RNN (dRNN) model to learn and integrate the dynamic characteristics of actions.

The dRNN model proposed by Veeriah et al. is based on the following observation: the internal state of each memory unit contains accumulated information about the spatio-temporal structure, that is, it is a long short-term representation of the input sequence. Therefore, the derivative of states (DoS) $\frac{ds_t}{dt}$ quantifies the information change at each time step t . In other words, a large number of DoS is an indicator of a significant spatio-temporal structure, which contains information dynamics caused by sudden changes in the action state. In this case, the gated unit should allow more information to enter the memory unit to update its internal state, otherwise, when the DoS amplitude is very small, the input information should be gated out of the memory unit, so that the internal state will not be affected by the current input. Therefore, DoS should be used as a factor to control the flow of information in and out of the internal state of the memory unit.

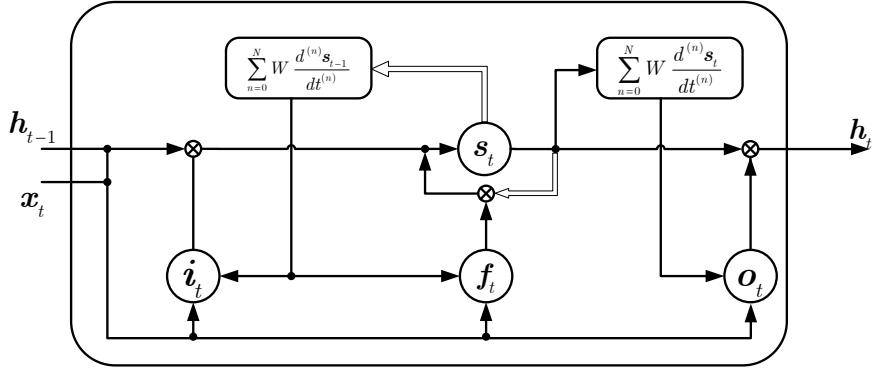


Figure 17: Architectures of dRNN at time t .

The dRNN has the following update equation:

$$i_t = \sigma\left(\sum_{n=0}^N W_{id}^{(n)} \frac{d^{(n)} s_{t-1}}{dt^{(n)}} + W_{ih} h_{t-1} + W_{ix} x_t + b_i\right)$$

$$f_t = \sigma\left(\sum_{n=0}^N W_{fd}^{(n)} \frac{d^{(n)} s_{t-1}}{dt^{(n)}} + W_{fh} h_{t-1} + W_{fx} x_t + b_f\right)$$

$$s_t = f_t \otimes s_{t-1} + i_t \otimes \tanh(W_{sh} h_{t-1} + W_{sx} x_t + b_s)$$

$$o_t = \sigma\left(\sum_{n=0}^N W_{od}^{(n)} \frac{d^{(n)} s_t}{dt^{(n)}} + W_{oh} h_{t-1} + W_{ox} x_t + b_o\right)$$

$$h_t = o_t \otimes \tanh(s_t) \quad (35)$$

where $\frac{d^{(n)} s_{t-1}}{dt^{(n)}}$ is the n-order DoS, and $W_{*d}^{(n)}$ are the corresponding mapping matrices. \otimes stands for element-wise product. In the memory cell, the input gate i_t and the forget gate f_t are controlled by DoS $\frac{d^{(n)} s_{t-1}}{dt^{(n)}}$ at time step $t - 1$, and the output gate o_t is controlled by the DoS $\frac{d^{(n)} s_t}{dt^{(n)}}$ at time step t .

The advantages of this model include: (1) High-order state derivatives can be used to learn complex time series. (2) It is more suitable to describe the short-term and long-term dynamics of behavior.

2.4.2. Deep Differential RNN

Naifan Zhuang et al. propose deep differential RNNs (d^2) [43], inspired by Alex Graves [44] to adjust the LSTM gate through an individual order of DoS. In the case of increasing DoS orders, multiple layers of LSTM units are stacked. Specifically, the first layer of d^2 uses zero-order DoS, which is similar to traditional LSTM units; the second layer uses LSTM units with first-order DoS; the third layer uses LSTM units with second-order DoS, and so on. Because it integrates the DoS idea from dRNN and the idea of deep stacked layers from deep RNN, the model is called deep differentiable RNN (d^2). Each layer of d^2 learns the change of information gain through an individual order of DoS. As the unit layer of d^2 becomes deeper, this model can learn higher-order and more complex dynamic patterns.

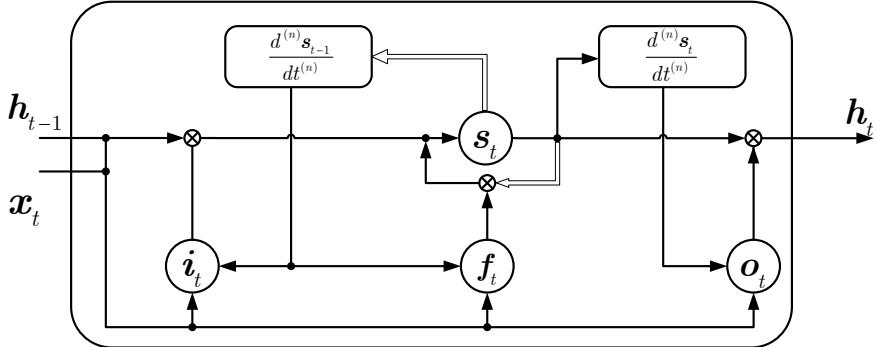


Figure 18: n -order (layer n) d^2 unit at time step t .

Fig. 18 shows the LSTM unit in the proposed d^2 model layer ($n + 1$). The hollow line represents the information flow of s_{t-1} . The following recurrent equation can be used to control the LSTM gate in the d^2 layer ($n + 1$).

1) Input gate:

$$i_t = \sigma(W_{id}^{(n)} \frac{d^{(n)} s_{t-1}}{dt^{(n)}} + W_{ih} h_{t-1} + W_{ix} x_t + b_i) \quad (36)$$

2) Forget gate:

$$f_t = \sigma(W_{fd}^{(n)} \frac{d^{(n)} s_{t-1}}{dt^{(n)}} + W_{fh} h_{t-1} + W_{fx} x_t + b_f) \quad (37)$$

3) Output gate:

$$o_t = \sigma(W_{fd}^{(n)} \frac{d^{(n)} s_t}{dt^{(n)}} + W_{ih} h_{t-1} + W_{ox} x_t + b_o) \quad (38)$$

The advantages of this model include: (1) It can simulate more complex dynamics than general LSTM. (2) It can detect significant spatio-temporal structure. (3) Reduce the problem of information distortion.

2.5. One-Layer RNN

Sitian Qin et al. proposed a one-layer recurrent neural network to solve the pseudoconvex optimization problem [45]. Consider the following nonsmooth pseudoconvex optimization problem [46] [47] [48]:

$$\text{minimize } f(x)$$

$$\text{subject to } g(x) \leq 0$$

$$Ax = b \quad (39)$$

where $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ is the vector form of decision variable, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is set to be local Lipschitz continuous, but it may be non-smooth, $g(x) = (g_1(x), g_2(x), \dots, g_p(x))^T : \mathbb{R}^n \rightarrow \mathbb{R}^p$ is a p -dimensional vector-valued function and g_i is convex function but may be non-smooth ($i = 1, 2, \dots, p$), $A \in \mathbb{R}^{m \times n}$ is a full row-rank matrix (i.e., $\text{rank}(A) = m \leq n$), and $b = (b_1, b_2, \dots, b_m)^T \in \mathbb{R}^m$. Without loss of generality, Sitian Qin et al. assume that the nonlinear pseudoconvex optimization problem (Eq.(39)) has at least an optimal solution. For convenience, suppose that

$$\Omega = \{x \in \mathbb{R}^n : Ax = b\} \quad (40)$$

$$S = \{x \in \mathbb{R}^n : g(x) \leq 0\} \quad (41)$$

$$X = \Omega \bigcap S = \{x \in \mathbb{R}^n : g(x) \leq 0, Ax = b\} \quad (42)$$

The following describes the model. First, assume that the following Assumption 1 holds.

Assumption 1: here exists $\hat{x} \in \mathbb{R}^n$, $r > 0$, such that $\hat{x} \in \text{int}(S) \bigcap \Omega$, $S \subseteq B(\hat{x}, r) = \{x \in \mathbb{R}^n : \|x - \hat{x}\| < r\}$.

Based on assumption 1, the inequality constraint set S defined as in Eq.(40)-(42) is bounded, that is, there is $R > 0$ such that

$$S \subset B(0, R) = \{x \in \mathbb{R}^n : \|x\| < R\} \quad (43)$$

Let

$$G_i(x) = g_i(x) + \max \{\|x\|, R\} - R, i = 1, \dots, p \quad (44)$$

$$G(x) = (G_1(x), \dots, G_p(x))^T \quad (45)$$

Similar to the proofs in M. Forti et al. [49], Q. Liu et al. [50] and W. Bian et al. [51], Sitian Qin et al. define a penalty function for the constraint set S as follows:

$$D(x) = \sum_{i=1}^p \max \{0, G_i(x)\} \quad (46)$$

Lemma 1: If assumption 1 holds, then:

1) The penalty function $D(x)$ is coercive, that is

$$\lim_{\|x\| \rightarrow +\infty} D(x) = +\infty \quad (47)$$

2) $\{x \in \mathbb{R}^n : g(x) \leq 0\} = \{x \in \mathbb{R}^n : G(x) \leq 0\}$

Lemma 2: Based on the assumption 1, the penalty function $D(x)$ is convex. In addition:

$$\partial D(x) = \begin{cases} \sum_{i \in I^0(x)} [0, 1] \partial G_i(x) & x \in bd(S) \\ \{0\} & x \in int(S) \\ \sum_{i \in I^+(x)} \partial G_i(x) + \sum_{i \in I^0(x)} [0, 1] \partial G_i(x) & x \notin S \end{cases} \quad (48)$$

The meaning of $x \in bd(S)$ is that x is on the outer edge of convex set S , and the meaning of $x \in int(S)$ is that x is inside convex set S . Where

$$I^0(x) = \{i \in \{1, 2, \dots, m\} : G_i(x) = 0\} \quad (49)$$

$$I^+(x) = \{i \in \{1, 2, \dots, m\} : G_i(x) > 0\} \quad (50)$$

In this section, assume that the objective function f in Eq.(39) satisfies the following assumptions.

Assumption 2:

- 1) The objective function f in Eq.(39) is pseudoconvex and regular on $X = \Omega \cap S$, while the local Lipschitz is continuous on $B(0, R) \cap \Omega$, where R comes from Eq.(43).
- 2) f is bounded below on the equality constraint set Ω , i.e., $\inf_{x \in \Omega} f(x) > -\infty$.

Remark 1: After simple row transformation of the matrix, the matrix A in Eq.(39) can be translated into a full row-rank matrix. Therefore, assume that $A \in \mathbb{R}^{m \times n}$ is a full row-rank matrix (i.e., $\text{rank}(A) = m \leq n$). In this case, AA^T is invertible.

Let

$$P = A^T(AA^T)^{-1}A \quad (51)$$

Based on the above results, Sitian Qin et al. propose a one-layer neural network for solving nonsmooth pseudoconvex optimization problem, which has the following dynamic characteristics:

$$\epsilon_0 \frac{d}{dt}x(t) \in -(I - P)[\partial f(x(t)) + \varepsilon(t)\partial D(x(t))] - A^T h(Ax(t) - b) \quad (52)$$

where ϵ_0 is a positive scaling constant, I is the identity matrix, $h(x) = (\tilde{h}(x_1), \tilde{h}(x_2), \dots, \tilde{h}(x_m))^T$ and its component is defined as

$$\tilde{h}(x_i) = \begin{cases} 1 & \text{if } x_i > 0 \\ [-1, 1] & \text{if } x_i = 0 \\ -1 & \text{if } x_i < 0 \end{cases} \quad (53)$$

and $\varepsilon : [0, +\infty) \rightarrow (0, +\infty)$ satisfies the following conditions:

- 1) $\varepsilon(t)$ is a non-decreasing differentiable function;
- 2) $\lim_{t \rightarrow +\infty} \varepsilon(t) = +\infty$;
- 3) $\varepsilon^{-1}(t)$ exists and differentiable;
- 4) $\int_0^{+\infty} \varepsilon^{-1}(t) dt = +\infty$

Remark 2: The function $\varepsilon(t)$ defined above is a Tikhonov regularization item, which serves as a penalty parameter.

The advantages of this model include: (1) The introduced penalty parameters do not need to determine specific parameters first, and have better convergence. (2) It is more convenient for design and implementation.

2.6. High-Order RNN

There can be two-way connections between the units of RecurrentNNs. This distinguishes them from feed-forward neural networks, in which the output of one unit is only connected to units in the next layer. In simple cases, the historical state of each unit or neuron is determined by a differential equation of the following form:

$$\dot{x}_i = -a_i x_i + b_i \sum_j w_{ij} y_j \quad (54)$$

where x_i is the state of the i -th neuron, a_i, b_i are constants, w_{ij} is the synaptic weight connecting the j -th input to the i -th neuron, and y_j is the j -th input connected to the above neuron. Each y_j is either an external input or a neuron state passed through a sigmoidal function, that is, $y_j = S(x_j)$, where $S(\cdot)$ is sigmoidal nonlinearity.

In a recurrent second-order neural network, the total input of a neuron is not only a linear combination of the input components y_j , but also a linear combination of their product $y_j y_k$. In addition, high-level interaction research including ternary $y_j y_k y_l$ and quaternary representations can also be carried out. This type of neural network forms a recurrent high-order neural network (RHONN) [52] [53].

Consider a RHONN consisting of n neurons and m inputs. The state of each neuron is controlled by the following differential equation:

$$\dot{x}_i = -a_i x_i + b_i \left[\sum_{k=1}^L w_{ik} \prod_{j \in I_k} y_j^{d_j(k)} \right] \quad (55)$$

where $\{I_1, I_2, \dots, I_L\}$ is the set of L disordered subsets of $\{1, 2, \dots, m+n\}$, a_i, b_i are real coefficients, w_{ik} is the (adjustable) synaptic weight of the neural network, and $d_j(k)$ is a non-negative integer. The state of the i -th neuron is again represented as x_i , $y = [y_1, y_2, \dots, y_{m+n}]^T$ is the vector composed of the input of each neuron, defined as follows:

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y_{n+1} \\ \vdots \\ y_{m+n} \end{bmatrix} = \begin{bmatrix} S(x_1) \\ \vdots \\ S(x_n) \\ u_1 \\ \vdots \\ u_m \end{bmatrix} \quad (56)$$

where $u = [u_1, u_2, \dots, u_m]^T$ is the external input vector of the network. The function $S(\cdot)$ is a monotonically increasing sigmoidal function with the following form:

$$S(x) = \alpha \frac{1}{1 + e^{-\beta x}} - \gamma \quad (57)$$

where α and β are positive real numbers, and γ is a real number. In special cases, let $\alpha = \beta = 1$ and $\gamma = 0$ to get the logistic function, and let $\alpha = \beta = 2$ and $\gamma = 1$ to get the hyperbolic tangent function; these are the most commonly used sigmoidal activation functions in neural network applications.

On this basis, the L -dimensional vector z is introduced, which is defined as follows:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_L \end{bmatrix} = \begin{bmatrix} \prod_{j \in I_1} y_j^{d_j(1)} \\ \prod_{j \in I_2} y_j^{d_j(2)} \\ \vdots \\ \prod_{j \in I_L} y_j^{d_j(L)} \end{bmatrix} \quad (58)$$

Therefore, the RHONN model is simplified to:

$$\dot{x}_i = -a_i x_i + b_i \sum_{k=1}^L w_{ik} z_k \quad (59)$$

In addition, if the adjustable parameter vector is defined as $\theta_i := b_i [w_{i1}, w_{i2}, \dots, w_{iL}]^T$, the above formula becomes:

$$\dot{x}_i = -a_i x_i + \theta_i^T z \quad (60)$$

The vector $\{\theta_i : i = 1, \dots, n\}$ represents the adjustable weight of the network, and the coefficient $\{a_i : i = 1, \dots, n\}$ is a part of the underlying network architecture, which is fixed during training. In order to ensure that the bounded-input bounded-output (BIBO) of each neuron x_i is stable, it is assumed that each a_i is positive. Kosmatopoulos et al. found that in the special case of the continuous-time Hopfield model [54], $a_i = 1/R_i C_i$, where $R_i > 0$ and $C_i > 0$ are the resistance and capacitance of the i -th of the network, respectively.

The dynamic behavior of the entire network is expressed as a vector:

$$\dot{x} = Ax + \Theta^T z \quad (61)$$

where $x = [x_1 \dots x_n]^T$, $\Theta = [\theta_1 \dots \theta_n] \in \mathbb{R}^{L \times n}$, $A := \text{diag}\{-a_1, -a_2, \dots, -a_n\}$ is a diagonal matrix of $n \times n$. Because each a_i is positive, A is a stable matrix. Although not written in the formula, the vector z is a function of the network state x and the external input u .

The advantages of this model include: (1) If enough high-order connections are allowed then this network is capable of approximating arbitrary dynamical systems. (2) It is a learning algorithm that can ensure the stability of the entire system and is effective.

2.7. Highway Networks

Next, the various network architectures of Highway Networks will be described in detail.

2.7.1. Highway Networks

Rupesh Kumar Srivastava et al. [55] took inspiration from the LSTM architecture and further expanded the Mixture of Units (MoU) architecture by adding additional units (implemented using

nonlinear transformation C) [56] [57]. In addition to these units within the layer, they can also learn to control the flow of information across layers:

$$y_l = H(x_l) \cdot T(x_l) + x_l \cdot C(x_l) \quad (62)$$

Since there are now two types of gating units, the gating conversion is named as T and C . T realizes the transform gate, which learns the contribution of the regular conversion H of the input x to the total output y of the layer. C implements a carry gate, which learns how many inputs are brought to the output. These gates are similar to the input and forget gates in the LSTM architecture, and a constant-weight skip connection (weight 1) from input to output is similar to the recurrent self-connection used by the LSTM unit. Networks with this architecture are called highway networks because they allow information to flow unimpeded across layers.

The conversion realized by the highway layer is much more flexible than the multilayer perceptron. Pay special attention to the specific output of T and C :

$$y_l = \begin{cases} x_l, & \text{if } T(x_l) = 0, C(x_l) = 1 \\ H(x_l), & \text{if } T(x_l) = 1, C(x_l) = 0 \end{cases} \quad (63)$$

Similarly, for the derivative of the layer outputs with respect to its inputs,

$$\frac{dy_l}{dx_l} = \begin{cases} I, & \text{if } T(x_l) = 0, C(x_l) = 1 \\ H'(x)_l, & \text{if } T(x_l) = 1, C(x_l) = 0 \end{cases} \quad (64)$$

Therefore, according to the output of the transition and carry gate, the highway layer can change its behavior between the identity layer (a layer that simply passes its input) and their unit combination. This design allows for credit assignment at a greater depth, because during back propagation, the gradient can flow in the reverse direction without reducing the path through which the carry gate output is non-zero.

The advantages of this model include: (1) It can solve the difficult problem of deep network training. (2) Effectively alleviate the gradient problem.

2.7.2. Recurrent Highway Networks

Georg Zilly et al. proposed the Recurrent Highway Network (RHN) layer [58], in which one or more highway layers are in a recurrent state transition (equal to the desired recurrence depth). Formally, let $W_{H,T,C} \in \mathbb{R}^{n \times m}$ and $R_{H_l,T_l,C_l} \in \mathbb{R}^{n \times n}$ denote the weight matrix of the nonlinear transformation of H and the weight matrix of the gates T and C at layer $l \in \{1, \dots, L\}$. The bias is denoted as $b_{H_l,T_l,C_l} \in \mathbb{R}^n$, s_l denotes the intermediate output of layer l , and $s_0^{[t]} = y^{[t-1]}$. The following describes a RHN layer with a recurrence depth of L :

$$s_l^{[t]} = h_l^{[t]} \cdot t_l^{[t]} + s_{l-1}^{[t]} \cdot c_l^{[t]} \quad (65)$$

$$h_l^{[t]} = \tanh(W_H x^{[t]} \mathbb{I}_{\{l=1\}} + R_{H_l} s_{l-1}^{[t]} + b_{H_l}) \quad (66)$$

$$t_l^{[t]} = \sigma(W_T x^{[t]} \mathbb{I}_{\{l=1\}} + R_{T_l} s_{l-1}^{[t]} + b_{T_l}) \quad (67)$$

$$c_l^{[t]} = \sigma(W_C x^{[t]} \mathbb{I}_{\{l=1\}} + R_{C_l} s_{l-1}^{[t]} + b_{C_l}) \quad (68)$$

and $\mathbb{I}_{\{\cdot\}}$ is the indicator function.

The schematic diagram of RHN calculation is shown in Fig.(19). The output of the RHN layer is the output of the L -th layer of the highway layer, that is, $y^{[t]} = s_L^{[t]}$.

Note that in the recurrent transition, $x^{[t]}$ is only directly converted by the first highway layer ($l = 1$). For this layer, $s_{l-1}^{[t]}$ is the output of the RHN layer at the previous time step. The subsequent highway layer only processes the output of the previous layer. The vertical dashed line in Fig.(19) separates the multiple highway layers in the recurrent transition.

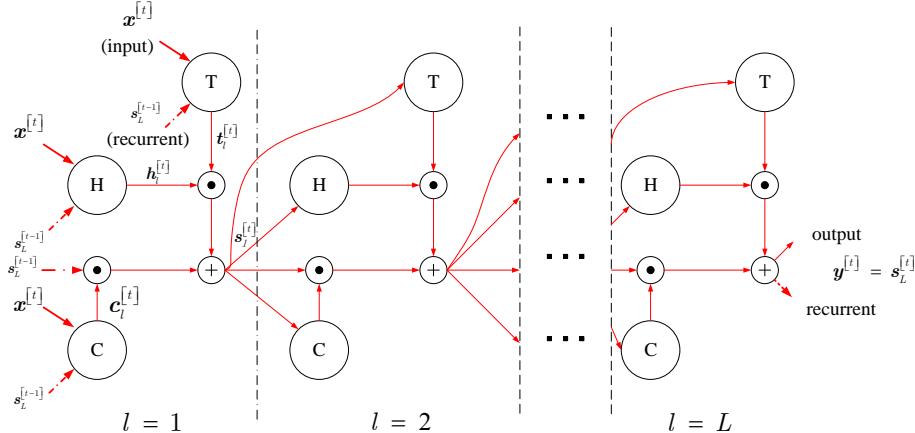


Figure 19: The figure shows the calculation of the RHN layer in the recurrent loop. Vertical dashed lines delimit stacked highway layers. The horizontal dashed line means to extend the recurrent depth by stacking more layers.

Conceptually, the RHN layer with $L = 1$ is basically a basic variant of the LSTM layer. Like other variants [59] [60] [61], it retains the basic essential components of LSTM—the multiplier gating unit, which can control information flow through self-connection. The RHN layer will naturally expand to $L > 1$, thereby expanding the LSTM to simulate more complex state transitions.

The simpler RHN layer formula can be analyzed similar to the standard RNN based on the Geršgorin circle theorem (GCT). Omit the input and biases, the temporal Jacobian $A = \partial y^{[t]} / \partial y^{[t-1]}$ of the RHN layer recurrence depth of 1 ($y^{[t]} = h^{[t]} \cdot t^{[t]} + y^{[t-1]} \cdot c^{[t]}$) is given by the following formula:

$$A = \text{diag}(c^{[t]}) + H' \text{diag}(t^{[t]}) + C' \text{diag}(y^{[t-1]}) + T' \text{diag}(h^{[t]}) \quad (69)$$

$$H' = R_H^\top \text{diag} \left[\tanh'(R_H y^{[t-1]}) \right] \quad (70)$$

$$T' = R_T^\top \text{diag} \left[\sigma'(R_T y^{[t-1]}) \right] \quad (71)$$

$$C' = R_C^\top \text{diag} \left[\sigma'(R_C y^{[t-1]}) \right] \quad (72)$$

and has a spectrum of:

$$\text{spec}(A) \subset \bigcup_{i \in \{1, \dots, n\}} \{ \lambda \in \mathbb{C} \mid \left\| \lambda - c_i^{[t]} - H'_{ii} t_i^{[t]} - C'_{ii} y_i^{[t-1]} - T'_{ii} h_i^{[t]} \right\|_{\mathbb{C}} \leq \sum_{j=1, j \neq i}^n \left| H'_{ij} t_i^{[t]} + C'_{ij} y_i^{[t-1]} + T'_{ij} h_i^{[t]} \right| \} \quad (73)$$

Eq.(73) shows the influence of gating on the eigenvalues A . Compared with RecurrentNNs, the RHN layer has more flexibility in adjusting the center and radius of Geršgorin circle, and it is especially able to notice two restrictions. If all the carry gates are fully opened and the transform gates are fully closed, then $c = 1_n$, $t = 0_n$, $T' = C' = 0_{n \times n}$ (because σ is saturated).

This leads to $c = 1_n$, $t = 0_n \Rightarrow \lambda_i = 1 \forall i \in \{1, \dots, n\}$.

Since the radius of the Geršgorin circle is reduced to 0, and each diagonal element is set to $c_i = 1$, all eigenvalues are set to 1. In another extreme case, if $c = 0_n$ and $t = 1_n$, then the eigenvalue is the eigenvalue of H' . When the gate changes between 0 and 1, each characteristic value of A can be dynamically adjusted to any combination of the above restriction behaviors.

The advantages of this model are: (1) It is a powerful new model designed to take advantage of increased depth in the recurrent transition while retaining the ease of training of LSTMs. (2) Solve sequential processing tasks.

2.7.3. Highway LSTM RNNs

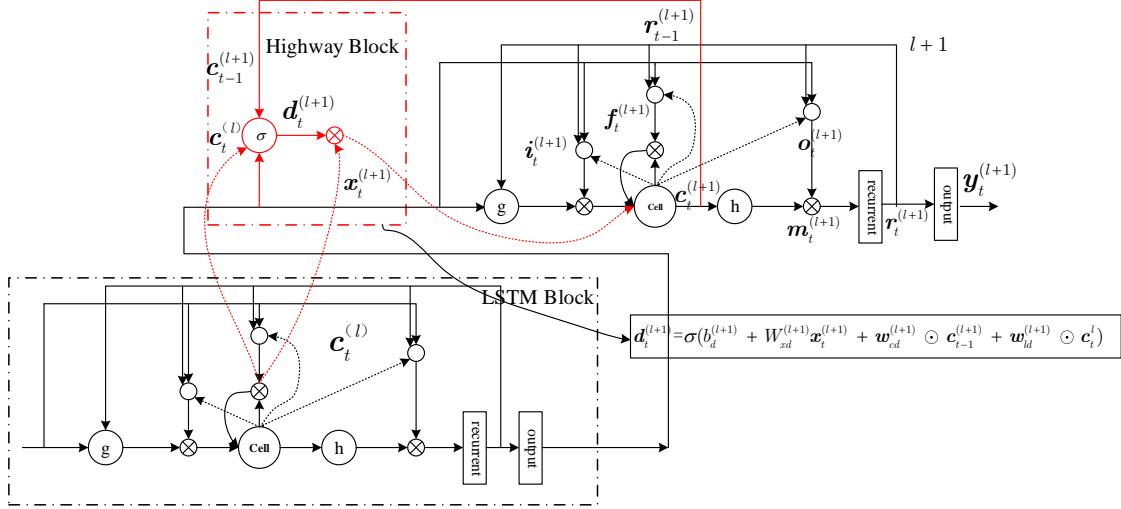


Figure 20: Highway LSTM RNNs

The highway LSTM (HLSTM) RNNs [62] proposed by Yu Zhang et al. are shown in Fig.(20). It has a direct gated connection between the memory cell c_t^l of the lower layer l and the memory cell c_t^{l+1} of the upper layer $l+1$ (in the red block in Fig.(20)). The carry gate controls the amount of information that flows directly from the lower unit to the upper unit. The gate function of layer $l+1$ at time step t is:

$$d_t^{(l+1)} = \sigma(b_d^{(l+1)} + W_{xd}^{(l+1)}x_t^{(l+1)} + w_{cd}^{(l+1)} \odot c_{t-1}^{(l+1)} + w_{ld}^{(l+1)} \odot c_t^l) \quad (74)$$

where $b_d^{(l+1)}$ is the bias and $W_{xd}^{(l+1)}$ is the weight matrix connecting the carry gate to the input of this layer. $w_{cd}^{(l+1)}$ is the weight vector from the carry gate to the past unit state in the current layer. $w_{ld}^{(l+1)}$ is the weight vector connecting the carry gate to the lower memory unit. $d_t^{(l+1)}$ is the activation vector of the carry gate on layer $l+1$. Using the carry gate, HLSTM [63] [64] RNN calculates the state of the unit on layer $l+1$ according to the following formula:

$$c_t^{(l+1)} = d_t^{(l+1)} \odot c_t^l + f_t^{(l+1)} \odot c_{t-1}^{(l+1)} + i_t^{(l+1)} \odot \tanh(W_{xc}^{(l+1)}x_t^{(l+1)} + W_{hc}^{(l+1)}m_{t-1}^{(l+1)} + b_c) \quad (75)$$

while other equations are the same as those in general LSTM.

Therefore, due to the output of the carry gate, the highway connection can smoothly change its behavior between LSTM layers, or simply pass its unit memory from the upper layer. When training a deeper LSTM, the highway connections between units of different layers make the influence between units of one layer to another layer more direct and can alleviate the problem of gradient disappearance.

2.7.4. Bidirectional Highway LSTM RNNs

The unidirectional LSTM RNNs described in the previous section can only use past history. However, in speech recognition, the future environment will also carry information, and it should be used to further enhance the acoustic model. Bidirectional RNNs process data by using two independent hidden layers in both directions, thereby taking advantage of past and future contexts. The results of A. Graves [44] et al., N. Jaitly et al. [65], and K. Chen et al. [66] showed that bidirectional LSTM RNNs can indeed improve the effect of speech recognition. Therefore, HLSTM RNNs are expanded from unidirection to bidirection. Note that the reverse layer follows the same equation used in the forward layer, except that $t - 1$ is replaced by $t + 1$ to take advantage of future frames, and the model runs from $t = T$ to 1. The output of the reverse layer and the forward layer are connected to form the input of the next layer.

2.7.5. Highway Convolutional Recurrent Deep Neural Network

The Highway Convolutional Recurrent Deep Neural Network (Highway CLDNN) structure proposed by Wei-Ning Hsu et al. [67] follows the design of T. N. Sainath et al. [68]. Fig.(21) illustrates the architecture of the model. Since the recurrent layer can capture the temporal relationship, at each time step, the frame input x_t without context is passed to the network. Wei-Ning Hsu et al. used filter bank features and pitch features to represent each frame x_t .

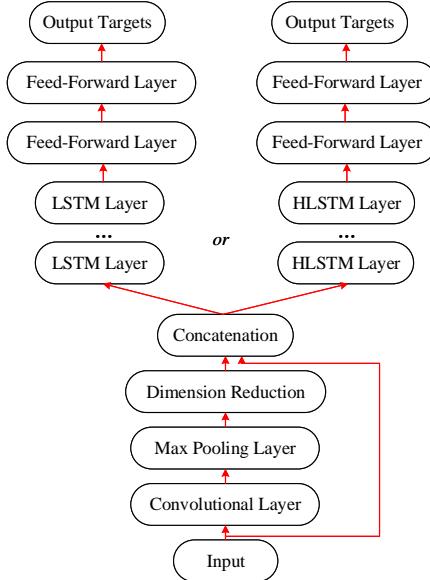


Figure 21: The left is CLDNN, and the right is Highway CLDNN.

In the field of speech recognition, in order to solve the speaker normalization problem, the input features are first passed to the convolutional layer. This layer generates 256 feature maps, and its core

is composed of 1×8 receptive field. Immediately after this convolutional layer, a pooling window and a non-overlapping maximum pooling with a step size of 3 are applied. Since the output dimension is still large (i.e., $[83/3] \times 256$), a 256-dimensional projection layer is added on top of the maximum pooling layer. Subsequently, in order to model the temporal relationship, the output of the projection layer is fed to the recurrent layer. In addition, according to the suggestion of T. N. Sainath, et al. [68], Wei-Ning Hsu et al. also pass the original input features x_t to the recurrent layer to provide input representations from different levels. Since the speech signal has information of different time scales [69], it is hoped that the recurrent layer can also capture the temporal relationship on different scales, and Deep RNN can capture this relationship [70]. Therefore, in order to utilize the deep recurrent structure, Highway LSTM is used for the recurrent layer. Specifically, 1024 units are applied on a 5-layer LSTM, and 512 dimensions are projected on each layer. Finally, the output of the last recurrent layer is fed to the fully connected feedforward layer, which can better distinguish the output targets. In this way, two layers of 1024 hidden unit searches are applied, and each layer has a corrected linear activation function.

2.8. Multidimensional RNNs

Next, the various network architectures of Multidimensional RNNs will be described in detail.

2.8.1. Two-dimensional LSTM

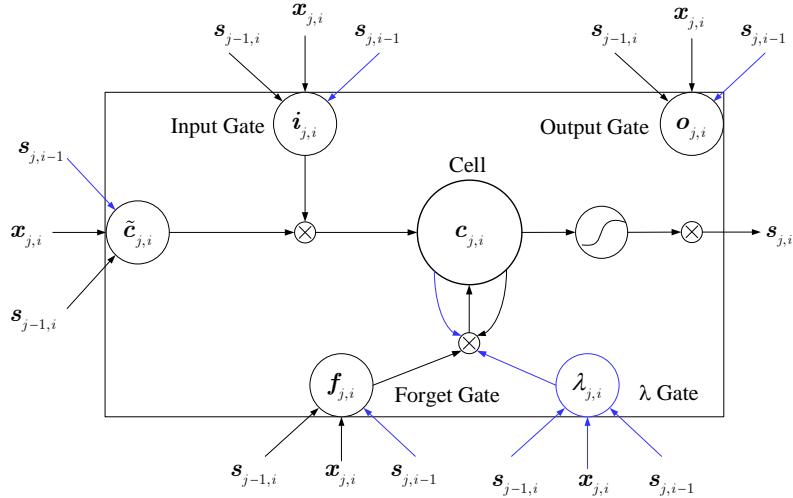


Figure 22: Two-dimensional LSTM unit. Additional connections and general LSTM are marked in blue.

Two-dimensional LSTM (2DLSTM) [71] was introduced by Graves [3] as a generalization of general LSTM. Fig.(22) illustrates a stable variant proposed by Leifert [72]. The 2DLSTM unit processes two-dimensional sequence data $x \in \mathbb{R}^{J \times I}$ of arbitrary length (J and I). At time step (j, i) , the calculation

of the unit depends on the vertical hidden state $s_{j,i-1}$ and the horizontal hidden state $s_{i,j-1}$ (see Eq.(76)–(80)). Similar to the LSTM unit, the 2DLSTM retains some state information in an internal unit state $c_{j,i}$. In addition to the input gate $i_{j,i}$, the forget gate $f_{j,i}$, and the output gate $o_{j,i}$ to control the information flows, the 2DLSTM also uses an additional λ gate $\lambda_{j,i}$. As written in Eq.(80), the calculation of λ gate activation is similar to other gates. Before passing the forgetting gate, use λ gate to weight the two predecessor unit $c_{j-1,i}$ and $c_{j,i-1}$ (Eq.(81)). g and σ are the tanh and sigmoid functions. V_S , W_S and U_S are weight matrices.

To train a 2DLSTM unit, back propagation through time (BPTT) is performed in two dimensions. Therefore, the gradient passes backward from time step (J, I) to $(1, 1)$, which is the origin.

$$i_{j,i} = \sigma(W_1 x_{j,i} + U_1 s_{j-1,i} + V_1 s_{j,i-1}) \quad (76)$$

$$f_{j,i} = \sigma(W_2 x_{j,i} + U_2 s_{j-1,i} + V_2 s_{j,i-1}) \quad (77)$$

$$o_{j,i} = \sigma(W_3 x_{j,i} + U_3 s_{j-1,i} + V_3 s_{j,i-1}) \quad (78)$$

$$\tilde{c}_{j,i} = g(W_4 x_{j,i} + U_4 s_{j-1,i} + V_4 s_{j,i-1}) \quad (79)$$

$$\lambda_{j,i} = \sigma(W_5 x_{j,i} + U_5 s_{j-1,i} + V_5 s_{j,i-1}) \quad (80)$$

$$c_{j,i} = f_{j,i} \circ [\lambda_{j,i} \circ c_{j-1,i} + (1 - \lambda_{j,i}) \circ c_{j,i-1}] + \tilde{c}_{j,i} \circ i_{j,i} \quad (81)$$

$$s_{j,i} = g(c_{j,i}) \circ o_{j,i} \quad (82)$$

2.8.2. Multidimensional RNNs

The basic idea of multidimensional RNN (MDRNN) [73] [74] is to replace the single recurrent connection in RecurrentNNs with the spatio-temporal dimensions in the data. These connections allow the network to create a flexible internal representation of the surrounding context, which is robust to local distortion.

The hidden layer of MDRNN scans the input in the one-dimensional strips and stores the activation in the buffer. The strips are sorted in such a way that at each point, the layer has visited these points one step back along each dimension. The hidden activations of these previous points are fed to the current point through recurrent connections along with the input. The two-dimensional structure is shown in Fig. 23.

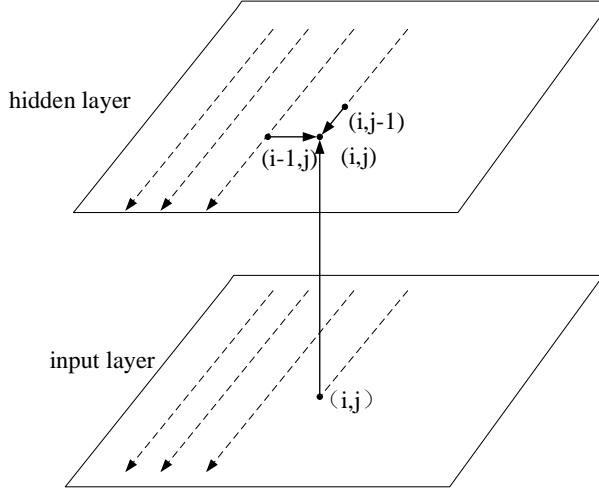


Figure 23: Two-dimensional RNN. The thick lines indicate the connections with the current point (i, j) . The connections in the hidden layer plane are recurrent. The dashed lines indicate the scanning strips along which previous points were visited, starting at the top left corner.

One such layer is sufficient to enable the network to access all contexts from the direction of the current point scan (e.g. to the top and left of (i, j) in Fig. 23). However, it is usually desirable to have surrounding contexts in all directions. The problem also exists in one-dimensional networks, so it is very useful to obtain information about the future and the past. The typical one-dimensional solution is a bidirectional recurrent network, with two independent hidden layers scanning the input forward and backward. The generalization of bidirectional network to n -dimensional requires 2^n hidden layers, starting from each corner in the n -dimensional hypercube and scanning in the opposite directions. For example, a two-dimensional network has four layers, and one layer starts from the top left corner and scans down and right, one layer starts from the bottom left and scans up and right, etc. All hidden layers are connected to a single output layer, so they can receive all the surrounding information context.

Uses the n -dimensional expansion of backpropagation over time to calculate the error gradient of the MDRNN. As in the one-dimensional case, the order of data processing is opposite to that of the forward process. Each hidden layer receives the output derivative and its own n ‘future’ derivatives at each time step.

Let a_j^p and b_j^p be respectively the input and activation of unit j at point $p = (p_1, \dots, p_n)$ in n -dimensional input sequence x with dimensions (D_1, \dots, D_n) . Let $p_d^- = (p_1, \dots, p_d - 1, \dots, p_n), p_d^+ = (p_1, \dots, p_d + 1, \dots, p_n)$. Let w_{ij} and w_{ij}^d be the feedforward connection weight from unit i to unit j and the recurrent connection weight from i to j along dimension d , respectively. Let θ_h be the activation function of the hidden unit h , for some unit j and some differentiable objective function O ,

let $\delta_j^p = \frac{\partial O}{\partial a_j^p}$. Then, the forward and backward equations of an n -dimensional MDRNN with I input unit, K output units and H hidden summation units are as follows:

Forward Pass:

$$a_h^p = \sum_{i=1}^I x_i^p w_{ih} + \sum_{\substack{d=1: \\ p_d > 0}}^n \sum_{\hat{h}=1}^H b_{\hat{h}}^{p_d^-} w_{\hat{h}h}^d \quad (83)$$

$$b_h^p = \theta_h(a_h^p) \quad (84)$$

Backward Pass:

$$\delta_h^p = \theta'_h(a_h^p) \left(\sum_{k=1}^K \delta_k^p w_{hk} + \sum_{\substack{d=1: \\ p_d < D_d - 1}}^n \sum_{\hat{h}=1}^H \delta_{\hat{h}}^{p_d^+} w_{\hat{h}h}^d \right) \quad (85)$$

The advantages of this model include: (1) Robust to any combination of input dimensions. (2) The multidimensional context can be modeled in a flexible way.

2.9. Bidirectional RNNs

Next, the various network architectures of Bidirectional RNNs will be described in detail.

2.9.1. Bidirectional Recurrent Neural Networks

One disadvantage of RecurrentNNs is that it can only use the previous context. In speech recognition, the entire sentence is transcribed at once, so the future context must be used. Bidirectional RNNs (BRecurrentNNs) [75] [76] use two separate hidden layers to process data in both directions to achieve it, and then forward these hidden layers to the same output layer.

The idea of BRecurrentNNs is to divide the state neurons of RecurrentNNs into the part responsible for the positive time direction (forward states) and the negative time direction (backward states). The outputs of the forward states are not connected to the input of the backward states, and vice versa. The above structure can be seen in Fig. 24, which unfolds along three time steps. Note that if there is no backward states, the structure is simplified to RecurrentNNs. If the forward states is removed, RecurrentNNs will be generated opposite to the time axis. When processing two time directions in the same network, the past and future input information of the current evaluation time frame can be directly used to minimize the objective function, without including the delay of future information.

The advantages of this model include: (1) It can be trained to a preset future frame without being restricted by input information. (2) Have faster development speed and better results.

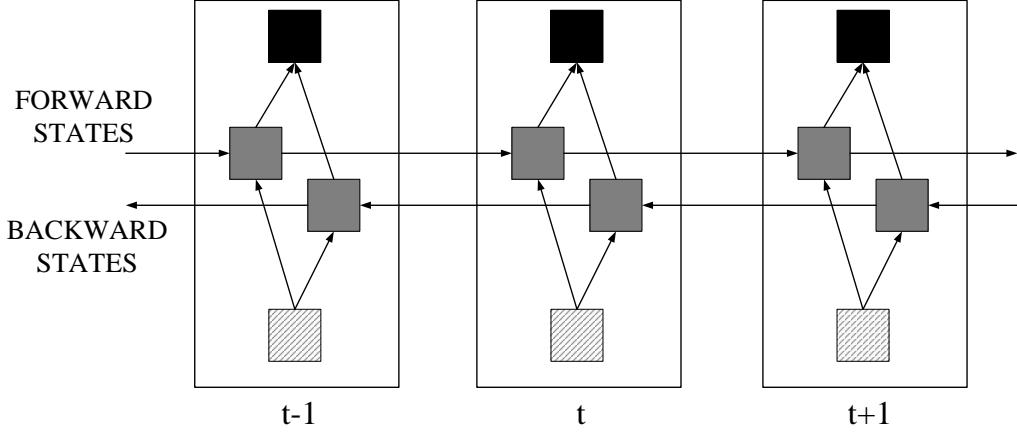


Figure 24: General structure of the BRecurrentNNs shown unfolded in time for three time steps.

2.9.2. Bidirectional Recursive Neural Networks

Extend the definition of the RecursiveNNs mentioned earlier to spread the information of the rest of the tree to each leaf node through the structure. In this way, decisions can be made on the leaf nodes based on the summary of the surrounding structure.

First, the upward level in the tree is represented as:

$$x_{\eta}^{\uparrow} = \sigma(W_L^{\uparrow}x_{l(\eta)}^{\uparrow} + W_R^{\uparrow}x_{r(\eta)}^{\uparrow} + b^{\uparrow}) \quad (86)$$

Note that if η is a leaf node, x_{η}^{\uparrow} is the initial representation x_{η} , which is similar to RecurrentNNs.

Next, add a downward layer on top of this upward layer:

$$x_{\eta}^{\downarrow} = \begin{cases} \sigma(W_L^{\downarrow}x_{p(\eta)}^{\downarrow} + V^{\downarrow}x_{\eta}^{\uparrow} + b^{\downarrow}), & \text{if } \eta \text{ is a left child} \\ \sigma(W_R^{\downarrow}x_{p(\eta)}^{\downarrow} + V^{\downarrow}x_{\eta}^{\uparrow} + b^{\downarrow}), & \text{if } \eta \text{ is a right child} \\ \sigma(V^{\downarrow}x_{\eta}^{\uparrow} + b^{\downarrow}), & \text{if } \eta \text{ is root} (\eta = \rho) \end{cases} \quad (87)$$

$p(\eta)$ is the parent node of η , W_L^{\downarrow} and W_R^{\downarrow} are connection weight matrices, respectively representing the downward connection of the parent node and the left and right child nodes, V^{\downarrow} is the weight matrix, which connects any node upward and downward, and b^{\downarrow} is the bias vector of the downward layer. Figure 25 depicts a RecursiveNNs [2] [77].

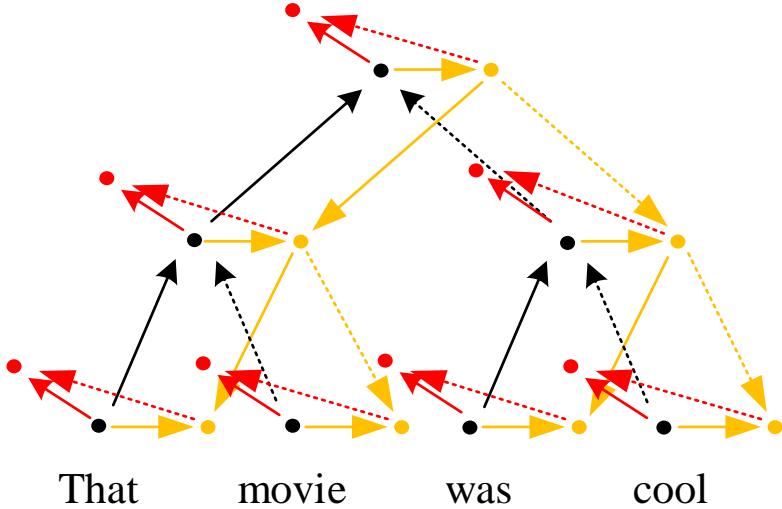


Figure 25: RecursiveNNs with bottom-up and top-down propagation. Black, orange and red indicate bottom-up, top-down, and output layers, respectively. The connection weights of the same color and style are shared.

For any node, x_η^\uparrow contains the subtree information rooted at η , and x_η^\downarrow contains other information of the tree, so each node of the tree contributes to the calculation of x_η^\downarrow . Therefore, x_η^\uparrow and x_η^\downarrow can be considered as a complete information summary of the surrounding structure of η . At the leaf or any node in general, use the output layer to make the final decision:

$$y_n = \gamma(U_\downarrow x_\eta^\downarrow + U_\uparrow x_\eta^\uparrow + c) \quad (88)$$

where U_\downarrow and U_\uparrow are the output weight matrices and c is the output bias vector.

In the supervised task, supervision occurs in the output layer. Then, in the training process, the structure back propagation method [78] is used to propagate the error upward, downward, downward, and upward. If necessary, the initial representation x can be updated using the backpropagation error, which can be fine-tuned in the settings for the word vector representation.

Note that this definition is similar in structure to the expanded recursive autoencoder [79]. However, the goals of these two structures are different. The expanded recursive autoencoder also has a downward propagation process, but the purpose is to reconstruct the initial representation. On the other hand, it is hoped that downward representation x^\downarrow is as different as upward representation x^\uparrow as possible, since the purpose is to capture information about the rest of the tree rather than a specific subtree under investigation. Therefore, the expanded recursive autoencoder does not use x^\uparrow (except at the root) when calculating x^\downarrow , but the RecursiveNNs does. .

3. Structured RNNs

Structured RNNs are divided into six categories. Grid RNNs form LSTM blocks into a multi-dimensional network. Each network contains a dimension, which not only calculates the time dimension but also calculates the depth dimension; Graph RNNs combine LSTM and CRNN to identify spatial structure learning dynamic patterns, not limited to image processing; Temporal RNNs are applied to basic time series classification tasks and label prediction problems; Lattice RNNs are mainly used for name entity recognition; Hierarchical RNNs mainly used in speech bandwidth expansion; Tree RNNs consider long and short distance interactions at different levels, and are used in language and image analysis.

3.1. Grid RNNs

Next, the various network architectures of Grid RNNs will be described in detail.

3.1.1. Grid LSTM

Grid LSTM [80] [81] was proposed by Kalchbrenner et al. [82]. Different from the general LSTM model that composes LSTM blocks into a temporal chain, the grid LSTM model composes LSTM blocks into a multi-dimensional grid, so each grid contains a set of LSTM blocks in each dimension (including the depth dimension). This architecture introduces the gated linear dependence of each dimensional between the states of adjacent units, alleviating the problem of vanishing gradients in all dimensions.

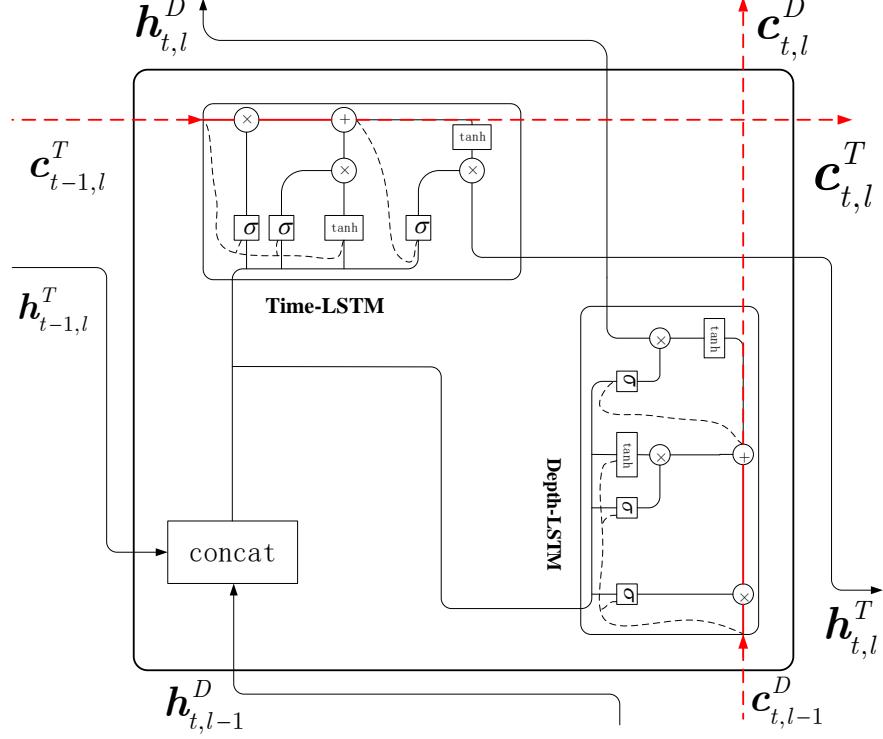


Figure 26: Grid LSTM

Here, Wei-Ning Hsu et al. consider a two-dimensional grid LSTM model, which are the time and depth dimensions, as shown in Fig. 26. The calculations in each grid are defined as follows:

$$x_{t,l} = [h_{t,l-1}^D; h_{t-1,l}^T] \quad (89)$$

$$(h_{t,l}^T, c_{t,l}^T) = \text{TIME-LSTM}(x_{t,l}, c_{t-1,l}^T, \Theta^T) \quad (90)$$

$$(h_{t,l}^D, c_{t,l}^D) = \text{DEPTH-LSTM}(x_{t,l}, c_{t,l-1}^D, \Theta^D) \quad (91)$$

Note that subscripts are used to denote time and depth, and superscripts are used to denote a collection of specific LSTM blocks. $c_{t,l}^i$ and $h_{t,l}^i$ are the unit state and unit output at time t and layer l of the i -LSTM, respectively, and Θ^i represents all the parameters of the i -LSTM. The unit output $h_{t,L}^D$ of the last layer of DEPTH-LSTM is passed to the softmax layer for classification.

The last thing that needs to be solved is $c_{t,0}^D$, its value is uncertain. The simplest solution is to set the value to zero, which provides flat initialization of the cell state regardless of the input value. Instead, a linear transformation can be applied:

$$c_{t,0}^D = V h_{t,0}^D \quad (92)$$

which has better performance in practice.

The advantages of this model include: (1) The depth can be extended in any dimension of the network. (2) The problem of vanishing gradients in all dimensions is alleviated.

3.1.2. Prioritized Grid LSTM

In the Eq.(39) in the previous section, the unit output of $TIME - LSTM$ is not classified at the current moment. In other words, the depth dimension should know the output of other dimensions in the current grid, making it implicitly deeper in the number of transformations before classification. Slightly modify the equation in the previous section to get:

$$x_{t,l}^T = [h_{t,l-1}^D; h_{t-1,l}^T]$$

$$(h_{t,l}^T, c_{t,l}^T) = TIME - LSTM(x_{t,l}^T, c_{t-1,l}^T, \Theta^T)$$

$$x_{t,l}^D = [h_{t,l-1}^D; h_{t,l}^T]$$

$$(h_{t,l}^D, c_{t,l}^D) = DEPTH - LSTM(x_{t,l}^D, c_{t,l-1}^D, \Theta^D) \quad (93)$$

After processing the $TIME - LSTM$ of the same grid, the input of $DEPTH - LSTM$ is updated. Wei-Ning Hsu et al. call this model a prioritized grid LSTM [80] and illustrate it in Fig. 27.

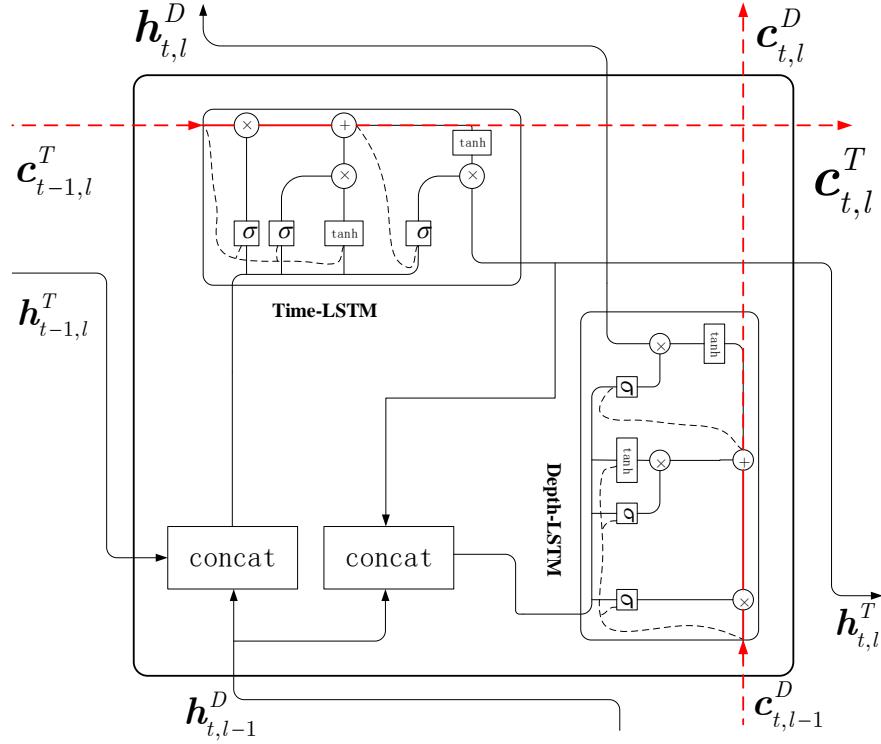


Figure 27: Prioritized Grid LSTM.

The advantages of this model include: (1) It can be calculated not only in the time dimension, but also in the depth dimension. (2) Prioritize the depth dimension in order to provide more updated information for the depth dimension. (3) Alleviate the problem of gradient disappearance.

3.1.3. Frequency Dependent Grid-RNN

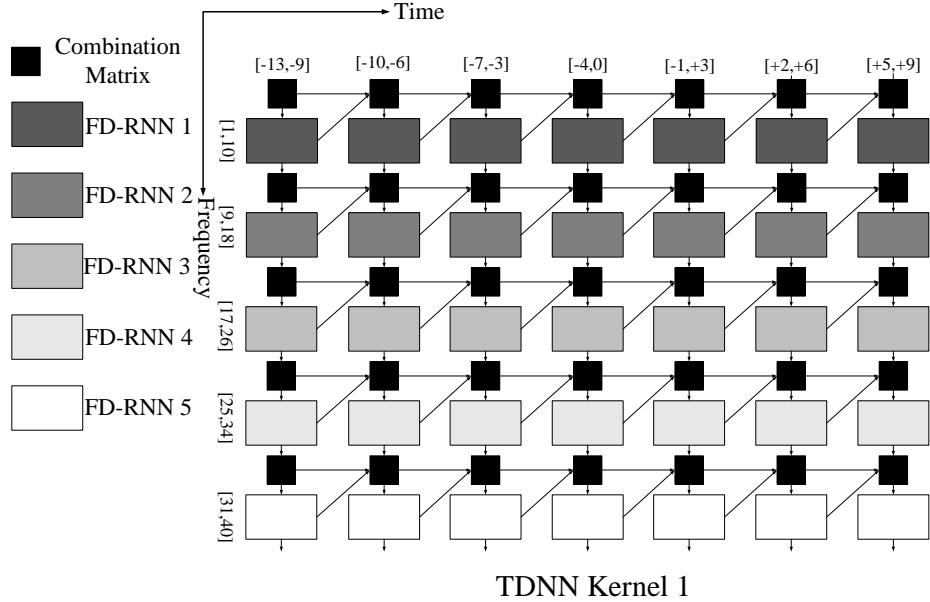


Figure 28: Frequency Dependent Grid-RNN Time Delay Neural Networks.

Kreyssing et al. proposed a frequency-dependent grid RNN architecture [83], as shown in Fig. 28. It uses RecurrentNNs and divides the input window ($t \in [-13, 9]$) into the same seven time periods. Therefore, it can be cleverly combined with the time delay neural network structure. The frequency axis is divided into five blocks of size 10, as shown in Fig. 28. The grid RNN is composed of two RNNs, one with $\sigma(\cdot)$ activation function and the other with linear activation function. $\sigma - RNN(h_{t,k}^F)$ performs the feature extraction, and C (Combination Matrix in Fig. 28) models the information flow between tanh instantiations. The linear activation function is used to improve the information flow of the $\sigma(\cdot)$ activation function. The structure is trained in the expanded form shown by G. Saon et al. [84].

The advantages of this model include: (1) *Linear – RNN* provides a linear path through the network. (2) It is helpful to untie the parameters of the $\sigma - RNN$ along the frequency axis.

3.2. Graph RNNs

Next, the various network architectures of Graph RNNs will be described in detail.

3.2.1. Graph Convolutional Recurrent Network

Seo et al. [85] proposed two graph convolutional recurrent network (GCRN) [86] [87] [88] architectures.

1)Model 1.The most straightforward definition is to stack the graph CNN for feature extraction and the LSTM for sequence learning;

$$x_t^{CNN} = CNN_{\mathcal{G}}(x_t)$$

$$i = \sigma(W_{xi}x_t^{CNN} + W_{hi}h_{t-1} + w_{ci} \odot c_{t-1} + b_i)$$

$$f = \sigma(W_{xf}x_t^{CNN} + W_{hf}h_{t-1} + w_{cf} \odot c_{t-1} + b_f)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t^{CNN} + W_{hc}h_{t-1} + b_c)$$

$$o = \sigma(W_{xo}x_t^{CNN} + W_{ho}h_{t-1} + w_{co} \odot c_t + b_o)$$

$$h_t = o \odot \tanh(c_t) \tag{94}$$

In this model, the input matrix $x_t \in \mathbb{R}^{n \times d_x}$ represents the observed value measured by the dynamic system network d_x in the graph \mathcal{G} at time t . x_t^{CNN} is the output of the graph CNN gate. In order to prove the concept, Seo et al. choose $x_t^{CNN} = W^{CNN} *_{\mathcal{G}} x_t$, where $W^{CNN} \in \mathbb{R}^{K \times d_x \times d_x}$ is the Chebyshev coefficient of the support vector K graph convolution kernel. The model gives the hidden variables of the spatial distribution and the unit state of size d_h given by the matrix c_t and $h_t \in \mathbb{R}^{n \times d_h}$. The peephole is controlled by $w_c \in \mathbb{R}^{n \times d_h}$. The weights $W_h \in \mathbb{R}^{d_h \times d_h}$ and $W_x \in \mathbb{R}^{d_h \times d_x}$ are the parameters of the fully connected layers. Architectures such as the above may be sufficient to capture the data distribution by using local stationarity, combined properties, and dynamic properties.

2)Model 2.In order to extend the convolutional LSTM model to the graph network, Seo et al. replaced Euclidean two-dimensional convolution $*$ with graph convolution $*_{\mathcal{G}}$:

$$i = \sigma(W_{xi}*_{\mathcal{G}}x_t + W_{hi}*_{\mathcal{G}}h_{t-1} + w_{ci} \odot c_{t-1} + b_i)$$

$$f = \sigma(W_{xf}*_{\mathcal{G}}x_t + W_{hf}*_{\mathcal{G}}h_{t-1} + w_{cf} \odot c_{t-1} + b_f)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}*_{\mathcal{G}}x_t + W_{hc}*_{\mathcal{G}}h_{t-1} + b_c)$$

$$o = \sigma(W_{xo}*_{\mathcal{G}}x_t + W_{ho}*_{\mathcal{G}}h_{t-1} + w_{co} \odot c_t + b_o)$$

$$h_t = o \odot \tanh(c_t) \quad (95)$$

In this model, the graph convolution kernel support vector K defined by the Chebyshev coefficient $W_h \in \mathbb{R}^{K \times d_h \times d_h}$ and $W_x \in \mathbb{R}^{K \times d_h \times d_x}$ determines the number of parameters, which is independent of the number n of nodes. In order to keep the symbol simple, Seo et al. used $W_{xi} *_{\mathcal{G}} x_t$ to represent the graph convolution of x_t and $d_h d_x$ filters, which is a function of Laplacian L parameterized by Chebyshev coefficient K . In a distributed computing setting, K controls the communication overhead, that is, the number of nodes that any given node should exchange in order to calculate its local state.

The advantages of this model include: (1) The isotropic filter can outperform the classic two-dimensional filter with fewer parameters. (2) The combination of graphs with CNN and RNNs is a multi-functional method to introduce and utilize secondary information by constructing a data matrix. (3) Simultaneous use of graphic spatial information and dynamic data information can improve learning accuracy and learning speed.

3.2.2. Temporal Dynamic Graph LSTM

Yuan Yuan et al. established a fully differentiable temporal dynamic graph LSTM (TD-Graph LSTM) [89] framework for action-driven video target detection tasks. Fig. 29 gives an overview of TD-Graph LSTM [90] [91] [92] [93]. Each frame in the input video first passes through a spatial convolution network to obtain the spatial visual features of the region proposals. Based on visual features, similar regions in two consecutive frames are found and correlated to represent the same object across temporal domains. Connect all semantically similar regions in two consecutive frames to construct a time graph structure, where graph nodes are represented by region proposals. Then, use the TD-Graph LSTM unit to recurrently circulate the information on the entire time graph, where the LSTM unit takes the spatial visual features as the input state. Thanks to graph topology, TD-Graph LSTM can incorporate temporal motion patterns for participating objects in actions in a more effective and meaningful way. TD-Graph LSTM outputs enhanced temporal-aware features of all regions. Then the region-level classification is used to generate classification confidence. Finally, these region-level predictions can be aggregated to generate frame-level object category predictions, which are supervised by the object category in the action label. The action-driven object classification loss can therefore be back propagated to all regions in the video as a whole, where the prediction of each frame can benefit each other.

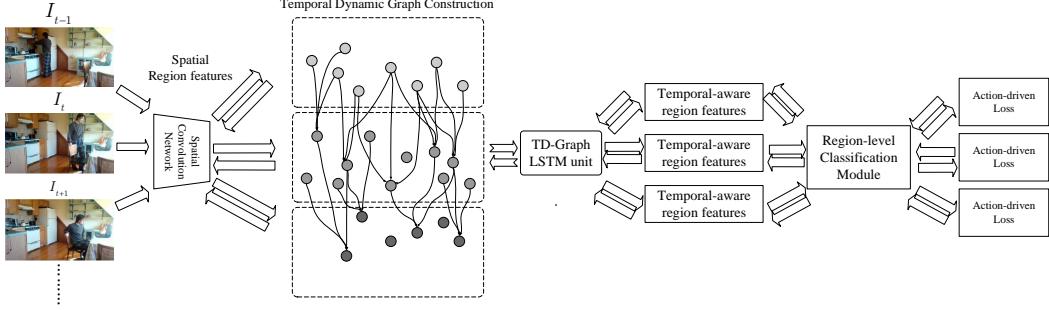


Figure 29: TD-Graph LSTM.

The advantages of this model include: (1) Global temporal reasoning can be realized. (2) Alleviate the label missing problem of each individual frame.

3.3. Temporal RNNs

Next, the various network architectures of Temporal RNNs will be described in detail.

3.3.1. Connectionist Temporal Classification

For tasks such as speech recognition, if the correspondence between input and label is unknown, RecurrentNNs has so far been limited to auxiliary roles. The problem is that standard training methods require that each input correspond to a separate output, which is usually impossible. To solve this problem, the traditional solution is to use the hidden Markov model to generate the target of the RNN, and then invert the output of the RNN to provide the observation probability.

Connectionist Temporal Classification (CTC) [94] is specifically used for temporal series classification tasks [95], that is, for sequence data label prediction problems where the correspondence between input and target labels is unknown. The CTC output layer uses the discriminant loss function to train the RecurrentNNs network to connect the weight matrix and bias, predict the class label of the unknown sequence, and does not use the hidden Markov model.

Given a voice input sequence x , the speech output sequence is y , and the target output sequence is z . The length of x is T , the length of z is U , and the number of possible phonemes is K . CTC uses the softmax layer to define a separate output distribution $Pr(k|t)$ at each time step t along the input sequence. This distribution covers K phonemes plus an additional blank symbol ϕ , ϕ represents non-output (so the size of the softmax layer is $K + 1$). Therefore, the network can decide whether to emit any tags or not to emit tags at every time step. In short, this defines the alignment distribution between the input sequence and the target sequence. Then, CTC uses a forward-backward algorithm to sum all possible alignments and determines the normalized probability $Pr(z|x)$ of the target sequence given the input sequence. Similar procedures have been used elsewhere in speech and handwriting

recognition to integrate possible segmentation; however, the difference with CTC is that it completely ignores segmentation, but adds up the single-time step label results.

RNNs trained with CTC are usually bidirectional to ensure that each $Pr(k|t)$ depends on the entire input sequence, not just from input to t . According to the deep bidirectional network, $Pr(k|t)$ is defined as follows:

$$y_t = W_{\vec{h}^N_y} \vec{h}_t^N + W_{\overleftarrow{h}^N_y} \overleftarrow{h}_t^N + b_y \quad (96)$$

$$Pr(k|t) = \frac{\exp(y_t[k])}{\sum_{k'=1}^K \exp(y_t[k'])} \quad (97)$$

Here, $y_t[k]$ is the k -th element of the irregular output vector y_t of length $K + 1$, and N is the number of bidirectional levels.

CTC uses a neural network to model all aspects of the sequence, and does not need to combine the network with a hidden Markov model, nor does it need to preprocess the training data, and it does not need to extract labels sequence from the network output through external post-processing. The practice of speech and handwriting recognition shows that a bidirectional LSTM network with a CTC output layer can effectively predict sequence labels, usually better than standard hidden Markov models.

3.3.2. Spatio-Temporal LSTM

RecurrentNNs have shown their advantages in modeling the complex dynamics of human activities as time series data, and have achieved good performance in the recognition of human behavior based on skeleton [96] [97] [98]. In the existing literature, RecurrentNNs are mainly used in the temporal domain to discover the discriminative dynamic characteristics and action patterns of action recognition. However, there is also different spatial information in the joint position and posture configuration of each video frame, and the sequence nature of body joints can apply RecurrentNNs-based modeling to the spatial domain.

Different from the existing method of connecting joint information as a whole, Jun Liu et al. extended the recurrent analysis to the spatial domain by discovering the spatial dependence pattern between different body joints, so they proposed a spatio-temporal LSTM (ST-LSTM) network [99], which can simultaneously simulate the temporal dependence between different frames and the spatial dependence of different joints at the same frame. Each ST-LSTM unit corresponds to a body joint, and it receives the hidden representation of its own joint at the previous time step, as well as the hidden representation of the previous joint in the current frame. The pattern of this model is shown in Fig. 30.

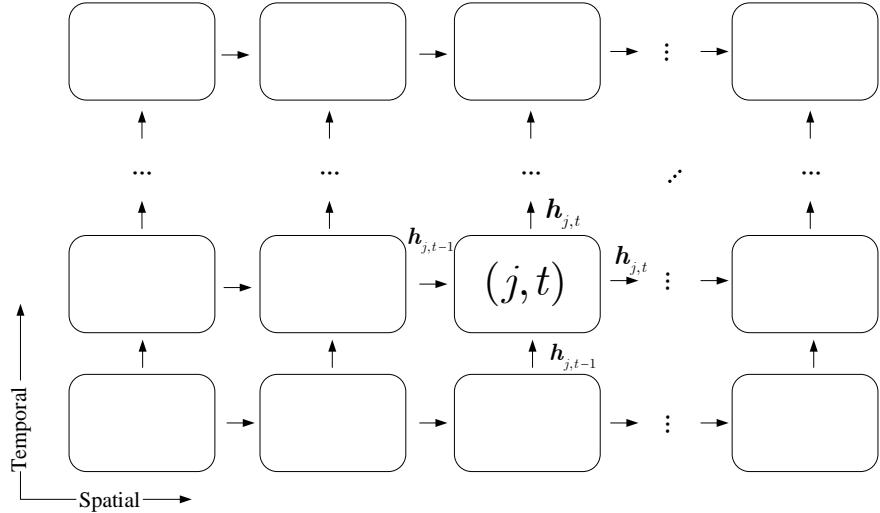


Figure 30: Illustration of the spatio-temporal LSTM network. In the temporal dimension, the corresponding body joints are fed to the frame. In the spatial dimension, the skeleton joints in each frame are fed sequentially. Each unit receives the hidden representation of the previous joint and the hidden representation of the same joint in the previous frame.

First, suppose the joints are arranged in a simple chain sequence. Use j and t to represent joints and frames respectively, $j \in \{1, \dots, J\}, t \in \{1, \dots, T\}$. Each ST-LSTM unit is represented by the input ($x_{j,t}$, the information of the corresponding joint at the current time step), the previous joint hidden ($h_{j-1,t}$) at the current time step and the same joint hidden ($h_{j,t-1}$) at the previous time step. As shown in Fig. 31, each unit also has two forget gates, $f_{j,t}^T$ and $f_{j,t}^S$, which deal with contextual information sources in the temporal and spatial dimensions, respectively. $u_{j,t}$ is the modulated input. The ST-LSTM transition equation is as follows:

$$\begin{pmatrix} i_{j,t} \\ f_{j,t}^S \\ f_{j,t}^T \\ o_{j,t} \\ u_{j,t} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left(M \begin{pmatrix} x_{j,t} \\ h_{j-1,t} \\ h_{j,t-1} \end{pmatrix} \right) \quad (98)$$

$$c_{j,t} = i_{j,t} \odot u_{j,t} + f_{j,t}^S \odot c_{j-1,t} + f_{j,t}^T \odot c_{j,t-1} \quad (99)$$

$$h_{j,t} = o_{j,t} \odot \tanh(c_{j,t}) \quad (100)$$

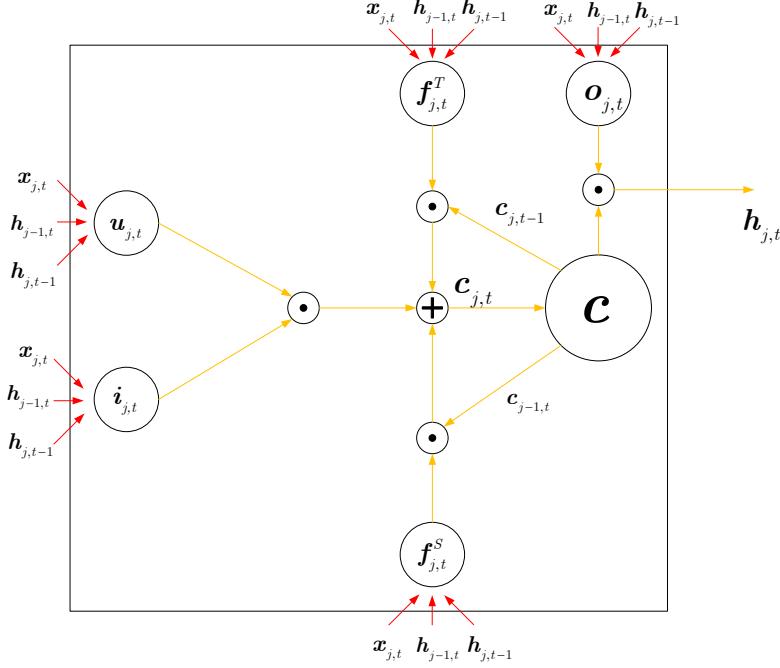


Figure 31: Illustration of the proposed ST-LSTM with one unit.

The advantages of this model include: (1) Better simultaneous analysis of information hidden sources related to actions in human skeleton sequences. (2) The reliability of the sequence data can be learned, and the influence of the input data on the long-term context representation update process stored in the unit memory unit can be adjusted accordingly.

3.3.3. Temporal Attention Model (TAM) for Feature Learning

In order to selectively focus on the most relevant images, the attention mechanism is applied to explore the temporal structure of a given image sequence. The entire process of TAM [100] is shown in Fig. 32.

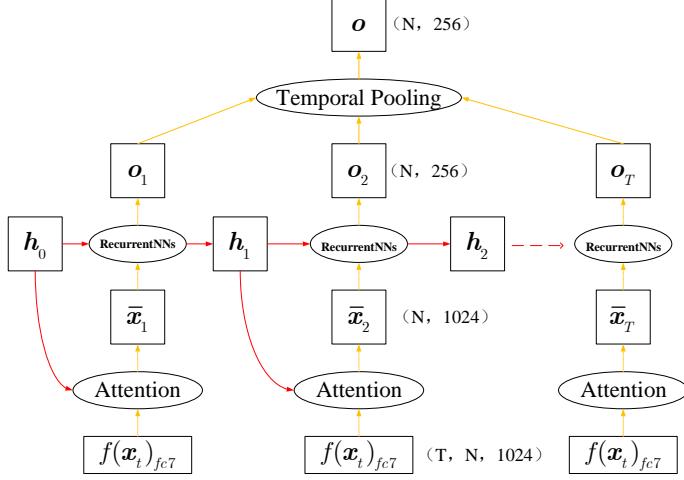


Figure 32: The structure of TAM. The input is T feature maps of the $fc7$ -th layer of the image sequence x . N is the data size of one batch. After the attention subnet, the weighted average \bar{x}_t of the feature T is obtained. Then \bar{x}_t is fed into the RecurrentNNs, and the RecurrentNNs outputs the feature o_t at each time step, and the final representation of x is the temporal average pool of $\{o_t\}_{t=1}^T$.

Assuming that each image sequence is represented as $x = \{x_t \mid x_t \in \mathbb{R}^D\}_{t=1}^T$, T is the length of the image sequence, and D is the dimension of the image. Let $f(x)$ denote the CNN. This model consists of two parts, namely attention unit and RecurrentNNs unit. At each time step t , the attention unit accepts $\{f(x_t)_{fc7}\}_{t=1}^T$ as input and generates a weighted average of these features, namely:

$$\bar{x}_t = \sum_{i=1}^T w_{t,i} f(x_t)_{fc7} \quad (101)$$

where $\{w_{t,i}\}$ is learned by the subnet, as shown in Fig. 33. h_{t-1} is the hidden state of RecurrentNNs at time step $t - 1$. $Uf(x_t)_{fc7}$, Vh_{t-1} and Wz_{t-1} are obtained through the fully connected layer. The softmax operation is used to guarantee $\sum_i w_{t,i} = 1$.

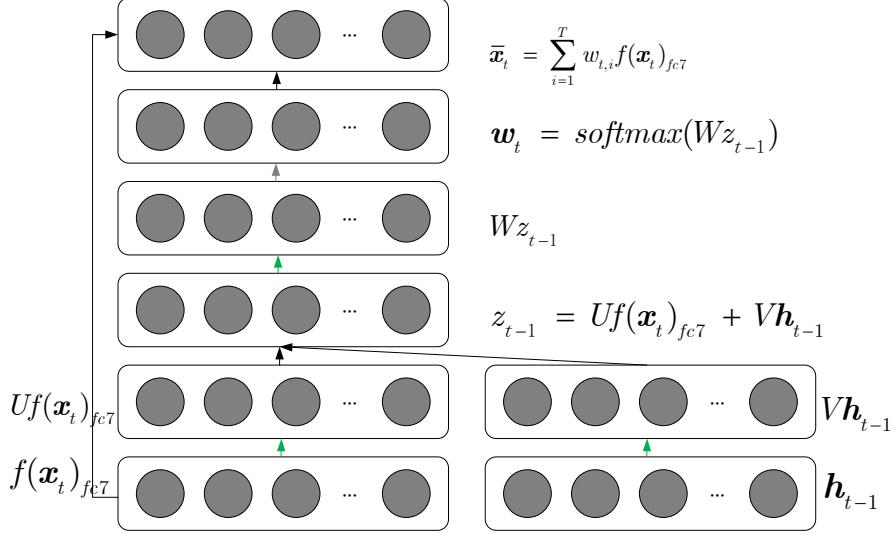


Figure 33: The subnet used to learn the correlation of each image in the image sequence. The subnet is represented by $w_t = \{w_{t,i}\}_{i=1}^T$. The green line indicates that they are fully connected, the black line indicates the sum or inner product of the elements, and the gray line indicates the softmax operation.

Then \bar{x}_t is fed to RecurrentNNs, which adopts the LSTM structure [101], which can summarize useful information in long-distance sequences. The final representation of the image sequence is the temporal average pool of each output [102]:

The advantages of this model include: (1) Through a temporal attention model, the most discriminative frame in a given video is automatically extracted. (2) The temporal attention model is conducive to feature learning, and the spatial recurrent model is conducive to metric learning.

3.4. Lattice RNNs

Next, the various network architectures of Lattice RNNs will be described in detail.

3.4.1. Character Lattice Model

The overall structure of the character lattice model is shown in Fig. 34 [103] [104] [105]. It can be seen as an extension of the character model, integrating character units and additional gates that control the flow of information.

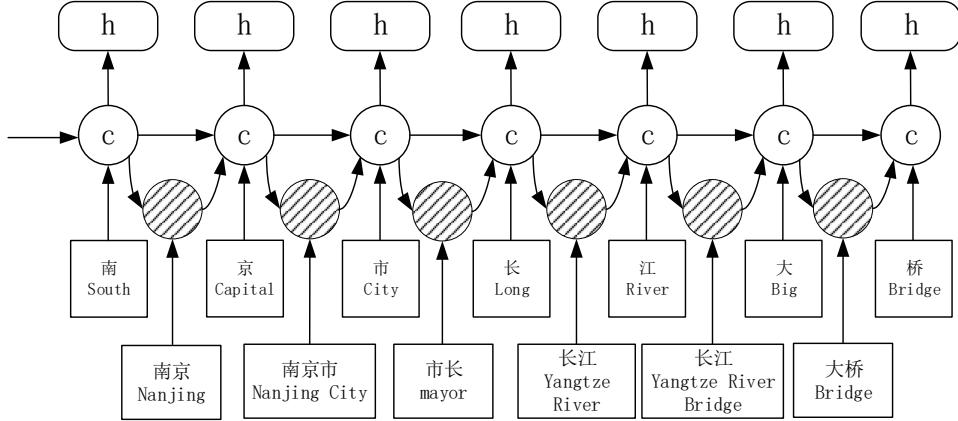


Figure 34: Lattice LSTM structure.

The input of the model is the character sequence c_1, c_2, \dots, c_m , and all character subsequences matching words in dictionary \mathbb{D} , where c_j represents the j -th character. The model involves four types of vectors: input vector, output hidden vector, unit vector and gate vector. In the character-based model, the character input vector is used to represent each character c_j in the character-based model:

$$x_j^c = e^c(c_j) \quad (102)$$

e^c represents the same character embedding lookup table. Use character unit vector c_j^c and hidden vector h_j^c on each c_j to construct the basic recurrent structure of the model, where c_j^c is used to record the recurrent information flow from the beginning of the sentence to c_j , and h_j^c is used for conditional random field sequence marking.

However, unlike character-based models, the calculation of c_j^c now considers the dictionary subsequence $w_{b,e}^d$ in the sentence. Each subsequence $w_{b,e}^d$ is expressed as follows:

$$x_{b,e}^w = e^w(w_{b,e}^d) \quad (103)$$

where e^w represents the same character embedding lookup table.

In addition, the word unit $c_{b,e}^w$ is used to represent the recurrent state of $x_{b,e}^w$ from the beginning of the sentence. The calculation formula of $c_{b,e}^w$ is as follows:

$$\begin{bmatrix} i_{b,e}^w \\ f_{b,e}^w \\ \tilde{c}_{b,e}^w \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \tanh \end{bmatrix} (W^{w\top} \begin{bmatrix} x_{b,e}^w \\ h_b^c \end{bmatrix} + b^w) \quad (104)$$

$$c_{b,e}^w = f_{b,e}^w \odot c_b^c + i_{b,e}^w \odot \tilde{c}_{b,e}^w \quad (105)$$

where $i_{b,e}^w$ and $f_{b,e}^w$ are the set of input and forget gates. There are no output gates for word units because tags are only executed at the character level.

With $c_{b,e}^w$, there are more recurrent paths to flow information into each c_j^c . For example, in Fig. 34, the input sources of c_7^c include x_7^c (桥Bridge), $c_{6,7}^w$ (大桥Bridge) and $c_{4,7}^w$ (长江大桥Yangtze River Bridge). Connect all $c_{b,e}^w$ ($b \in \{b' \mid w_{b',e}^d \in \mathbb{D}\}$) to unit c_e^c , and then use an additional gate $i_{b,e}^c$ for each sub-sequence unit $c_{b,e}^w$ to control its contribution to $c_{b,e}^c$:

$$i_{b,e}^c = \sigma(W^{l\top} \begin{bmatrix} x_e^c \\ c_{b,e}^w \end{bmatrix} + b^l) \quad (106)$$

Therefore, the word unit value c_j^c is calculated as follows:

$$c_j^c = \sum_{b \in \{b' \mid w_{b',j}^d \in \mathbb{D}\}} \alpha_{b,j}^c \odot c_{b,j}^w + \alpha_j^c \odot \tilde{c}_j^c \quad (107)$$

In the above formula, by setting the sum to 1, the gate value $i_{b,j}^c$ and i_j^c are normalized to $\alpha_{b,j}^c$ and α_j^c .

$$\alpha_{b,j}^c = \frac{\exp(i_{b,j}^c)}{\exp(i_j^c) + \sum_{b' \in \{b'' \mid w_{b'',j}^d \in \mathbb{D}\}} \exp(i_{b',j}^c)} \quad (108)$$

$$\alpha_j^c = \frac{\exp(i_j^c)}{\exp(i_j^c) + \sum_{b' \in \{b'' \mid w_{b'',j}^d \in \mathbb{D}\}} \exp(i_{b',j}^c)} \quad (109)$$

The final hidden vector h_j^c is calculated in the same way as the general LSTM. During the named entity recognition (NER) [106] training period, the loss value is back propagated to the parameters W^c , b^c , W^w , b^w , W^l and b^l making the model dynamically focus on more relevant words during NER marking.

The advantages of this model include: (1) Free choice of words in the context for disambiguation. (2) Use word information more effectively.

3.4.2. Lattice Language Models

Jacob Buckman et al. [107] defined some terms, using the term "token" represented by x_i to describe any indivisible items in the vocabulary that do not have other vocabulary items as components. The term "component" represented by k_i or x_i^j is used to describe a sequence of one or more tokens. This component represents a part of the complete string X and contains unit tokens from x_i to x_j : $x_i^j = [x_i, x_{i+1}, \dots, x_j]$, which also defines a "token vocabulary" (contains only a subset of the token vocabulary) and "component vocabulary" (similarly includes all components).

All segments of X can be expressed as the edge of the lattice path on the label prefix of X : $x_{<1}, x_{<2}, \dots, X$. The infimum is the empty prefix $x_{<1}$; the supremum is X ; if and only if x_i^j exists in

the component vocabulary, there is an edge from prefix $x_{<i}$ to the prefix $x_{<j}$, make $[x_{<i}; x_i^j] = x_{<j}$. As shown in Fig. 35, all paths through the lattice from $x_{<i}$ to X are segments from X to the token list on the traversed edge.

The probability of a particular prefix $p(x_{<j})$ is calculated by marginalizing all segments before x_{j-1} :

$$p(x_{<j}) = \sum_{S \in S(x_{<j})} \prod_{t=1}^{|S|} p(x_{s_{t-1}}^{s_t-1} | x_{<s_{t-1}}) \quad (110)$$

which defines $s_{|S|} = j$. The key to effectively calculating this probability is that this is a recursive formula, so it can be marginalized on the previous edge in lattice A_j instead of marginalizing all segments. Each item in A_j is a position $i (= s_{t-1})$, which indicates that there is an edge between the prefix $x_{<i}$ and the prefix $x_{<j}$ corresponding to the token x_i^j in the lattice. Therefore, $p(x_{<j})$ can be calculated as

$$p(x_{<j}) = \sum_{i \in A_j} p(x_{<i}) p(x_i^j | x_{<i}) \quad (111)$$

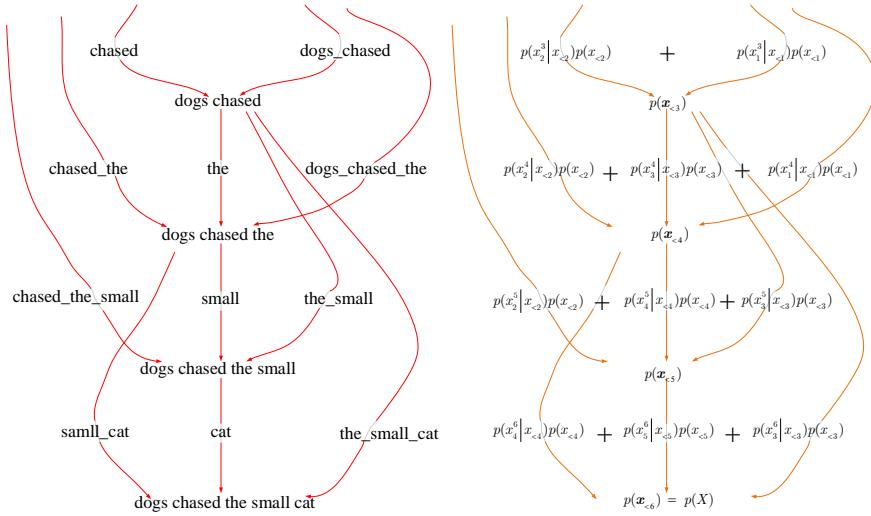


Figure 35: The lattice decomposition of a sentence and its corresponding lattice language model probability calculation.

Since X is the supremum prefix node, $p(X)$ can be calculated by using the formula $j = |X|$. To do this, it is necessary to calculate the probability of each of its $|X|$ predecessors. Each one needs to calculate $|X|$, which means that the calculation of $p(X)$ can be completed in $O(|X|^2)$. If it can be guaranteed that each node has the maximum number of input edges D , then there is $|A_j| \leq D$ for all j , and the boundary can be reduced to $O(D|X|)$ time.

The model proposed [108] above is agnostic to the shape of the lattice. Fig. 36 illustrates several potential lattice variants. Depending on how the lattice is constructed, this method is useful in many different contexts.

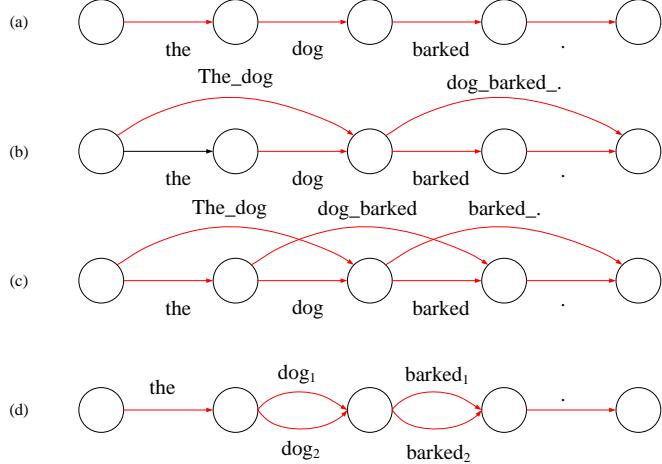


Figure 36: Examples of (a) single-path lattice, (b) sparse lattice, (c) dense lattice with $D = 2$ and (d) multiple lattices with $D = 2$ in the sentence "dog barking".

3.4.3. Neural Lattice Language Models

There is still a lack of part in the process of trying to apply the neural language model to the lattice. In the overall probability of equation(111), the probability $P(t|p)$ of the next segment must be calculated based on historical records. However, considering that there may be multiple paths in the lattice leading to p , this is not as simple as the case where only one segment is possible. Previous work on lattice-based language models [109] [110] [111] used a count-based n -gram model that only relied on each step in a limited historical context, so that the marginal probability is calculated in an accurate and effective way through dynamic programming. On the other hand, recurrent neural models rely on the entire context, causing them to lack this ability. Therefore, the main technical contribution of Jacob Buckman et al. is to describe the merging of lattices into a neural framework with infinite context by providing a method to approximate the hidden state of a RecurrentNNs. Technical methods currently include: direct approximation, Monte Carlo approximation, marginal approximation, Gumbel-Softmax interpolation.

The advantages of this model include: (1) All segments of the sentence are marginalized in an end-to-end manner. (2) Whether it is used to merge multiple phrases or embed multiple words, the neural lattice language model is better than the LSTM-based baseline in language modeling tasks.

3.4.4. Lattice Recurrent Unit

The Lattice Recurrent Unit (LRU) [112] model aims to have different information flows in time and depth dimensions. The LRU model can be seen as an extension of the GRU model.

The first LRU model is called Projected State LRU (PS-LRU), which will only decouple the projected state of each dimension. The second model is called Reset Gate LRU (RG-LRU), and it will further decouple the reset gate. The last one is called Update Gate LRU (UG-LRU or LRU), which can decouple all three components, including the update gate.

(1) PS-LRU

As the first model PS-LRU, the projection state \hat{h} is decoupled to create two new projection states \hat{h}_1 and \hat{h}_2 . Each is used to calculate a separate output state: h' and h'_2 . Formally, use the following update function to define PS-LRU:

$$\hat{h}_1 = \tanh \left([W_1^h \ U_1^h] \begin{bmatrix} h_1 \\ r \odot h_2 \end{bmatrix} \right) \quad (112)$$

$$\hat{h}_2 = \tanh \left([W_2^h \ U_2^h] \begin{bmatrix} r \odot h_1 \\ h_2 \end{bmatrix} \right) \quad (113)$$

$$h'_1 = z \odot h_1 + (1 - z) \odot \hat{h}_2 \quad (114)$$

$$h'_2 = z \odot h_2 + (1 - z) \odot \hat{h}_1 \quad (115)$$

The PS-LRU model uses the same update and reset gates for the two output states. Note that since there are two different outputs in this model, the hidden state of GRU output is decomposed into equation(114) and (115).

(2) RG-LRU

The reset gate is further decoupled, and two new gates r_1 and r_2 are obtained. This model is called RG-LRU and is defined as follows:

$$r_1 = \sigma \left([W_1^r \ U_1^r] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right) \quad (116)$$

$$r_2 = \sigma \left([W_2^r \ U_2^r] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right) \quad (117)$$

$$\hat{h}_1 = \tanh \left([W_1^h \ U_1^h] \begin{bmatrix} h_1 \\ r_2 \odot h_2 \end{bmatrix} \right) \quad (118)$$

$$\hat{h}_2 = \tanh \left([W_2^h \quad U_2^h] \begin{bmatrix} r_1 \odot h_1 \\ h_2 \end{bmatrix} \right) \quad (119)$$

Note that since r is now decoupled, the hidden state is decomposed into equation (118) and (119).

(3)UG-LRU In the final model UG-LRU, all three main components of the GRU are decoupled, including the update gate, and the z_1 and z_2 gate are defined as:

$$z_1 = \sigma \left([W_1^z \quad U_1^z] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right) \quad (120)$$

$$z_2 = \sigma \left([W_2^z \quad U_2^z] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \right) \quad (121)$$

$$h'_1 = z_1 \odot \hat{h}_2 + (1 - z_1) \odot h_1 \quad (122)$$

$$h'_2 = z_2 \odot \hat{h}_1 + (1 - z_2) \odot h_2 \quad (123)$$

Note that equation(114) and (115) need to be modified slightly because z_t^T and z_t^D are now decoupled.

The advantages of this model include: (1) When dealing with small data sets, it has the best accuracy, convergence speed and statistical efficiency among all baseline models. (2) Realize the goal of multi-layer network with limited learning resources by creating information flow along time and depth.

3.5. Hierarchical RNNs

Next, the various network architectures of Hierarchical RNNs will be described in detail.

3.5.1. Hierarchical RNNs

The hierarchical RNN (HRNN) [113] [114] [115] structure of speech bandwidth extension (BWE) is shown in Fig. 37. HRNN is composed of LSTM layer and feed-forward (FF) layer. These LSTM and FF layers in HRNN form a multi-layered hierarchical structure, and each layer works with a specific time resolution. The bottom layer (i.e., layer 1 in Figure 37) processes a single sample and outputs sample-level predictions. Each higher layer operates at a lower temporal resolution (i.e., processes more samples each time step). Except for the top layer, each layer sets conditions on the layer above it. The model structure is similar to the sampleRNN [116], the main difference is that the original sampleRNN model is an unconditional audio generator, which uses the history of the output waveform as the network input and generates the output waveform in an autoregressive way. However, the HRNN model shown in Fig. 37 directly describes the mapping relationship between the two waveform

sequences without considering the autoregressive characteristics of the output waveform. This HRNN structure is specifically designed for BWE, because the narrowband waveform is used as input in this task. Removal of autoregressive connections helps reduce computational complexity and facilitate parallel calculations during generation.

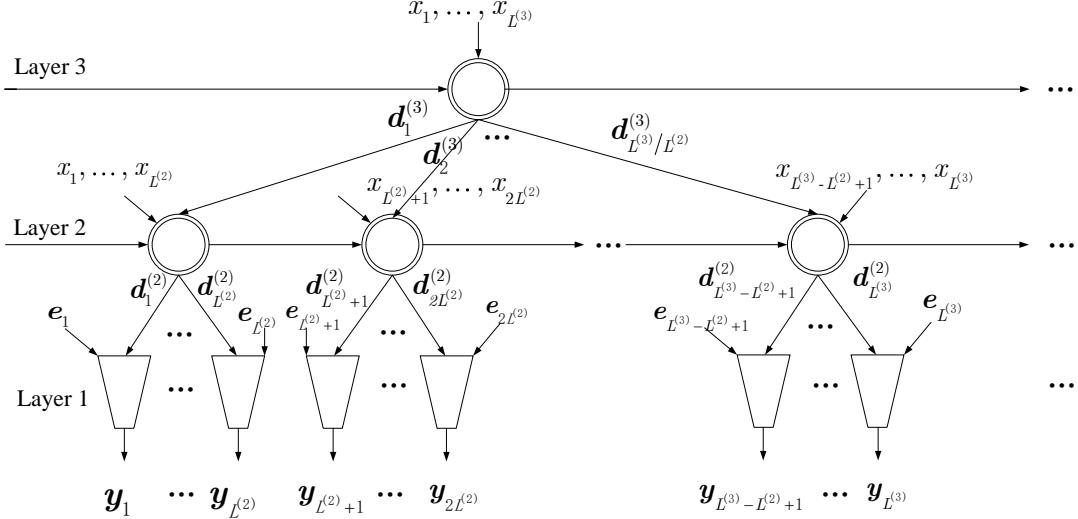


Figure 37: The structure of HRNNs for BWE..

The concentric circles in the figure above represent the LSTM layer, and the inverted trapezoid represents the FF layer.

Assume that the HRNN has K layers in total (for example, $K = 3$ in Fig. 37). The first layer is at the sample level, and the other $K - 1$ layers are at the frame level, and their working time resolution is lower than the samples.

3.5.2. Conditional Hierarchical RNNs

Some frame-level auxiliary features extracted from input narrowband waveforms, such as bottleneck (BN) features [117], have shown that they are very effective in improving the performance of vocoder-based BWE [118]. In order to combine these auxiliary inputs with the HRNN model, a conditional HRNN structure is designed, as shown in Fig. 38.

Compared with HRNNs, conditional HRNNs [113] [119] [120] add a conditional layer at the top. The input feature of the conditional layer is the frame-level auxiliary feature vector extracted from the input waveform, not the waveform sample. Assume that the total number of layers in the condition HRNN is K (for example, $K = 4$ in Fig. 38), and let $L^{(K)}$ represent the frame shift of the auxiliary input feature.

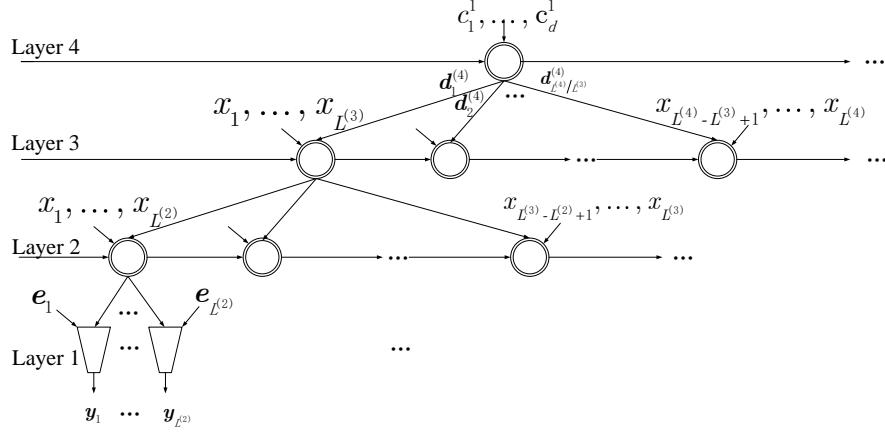


Figure 38: The structure of conditional HRNNs for BWE.

3.5.3. Hierarchical LSTM

Hierarchical LSTM (H-LSTM) [121] [122] [123] contains two coupled sub-networks: Pixel LSTM (P-LSTM) and Multi-scale Super-pixel LSTM (MS-LSTM) for processing surface labeling and relationship prediction. These two subnets provide complementary information to develop hierarchical scene contexts, and they are jointly optimized to improve performance.

As shown in Fig. 39, the input image first passes through a stack of convolutional layers and pooling layers to generate a set of convolutional feature maps. Then P-LSTM and MS-LSTM take these feature maps as input in the sharing mode, and output the pixel-wise geometric surface labeling and the interaction relationship between adjacent regions.

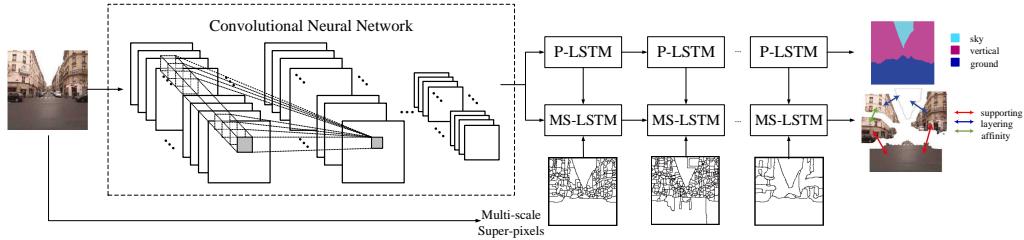


Figure 39: The proposed recurrent framework for geometric scene analysis. Each still image is first fed into several convolutional layers. Then these feature maps are respectively transferred to the stacked P-LSTM layer and MS-LSTM to generate the geometric surface labeling of each pixel and the interaction relationship between the regions.

Each LSTM unit in the i -th layer receives input x_i from the previous state and determines the current state composed of the hidden unit $h_{i+1} \in \mathbb{R}^d$ and the memory unit $c_{i+1} \in \mathbb{R}^d$ where d is the dimension of the network output. Similar to the work in Graves et al. [44], Zhanglin Peng et al. use g^u , g^f , g^c , and g^o to denote input, forget, memory and output gates respectively. Define W^u , W^f ,

W^c , W^o as the corresponding recurrent gate weight. Therefore, the hidden unit and memory unit of the current state can be calculated in the following way:

$$g^u = \phi(W^u * H_i) \quad (124)$$

$$g^f = \phi(W^f * H_i) \quad (125)$$

$$g^o = \phi(W^o * H_i) \quad (126)$$

$$g^c = \tanh(W^c * H_i) \quad (127)$$

$$c_{i+1} = g^f \odot c_i + g^u \odot g^c \quad (128)$$

$$h_{i+1} = \tanh(g^o \odot c_i) \quad (129)$$

where H_i represents the concatenation of input x_i and previous state h_i . ϕ is the sigmoid function in the form of $\phi(t) = 1/(1 + e^{-t})$ represents the element-wise product. According to Kalchbrenner et al. [82], the expressions (124)-(129) can be simplified to:

$$(c_{i+1}, h_{i+1}) = LSTM(H_i, c_i, W) \quad (130)$$

where W is the concatenation of four different types of recurrent gate weights.

The advantages of this model include: (1) Capturing different levels of context and better performing local reasoning. (2) A new recurrent neural network model for geometric scene analysis is proposed, which combines geometric surface labeling and relationship prediction.

3.5.4. Hierarchical Recurrent Attention Network

Chen Xing et al. proposed Hierarchical Recurrent Attention Network (HRAN) [124] [125] [126] to model the generation probability $p(y_1, \dots, y_T | U)$. U represents the context of the conversation and R represents the response. Fig. 40 shows the architecture of HRAN. Roughly speaking, before generating probabilities, HRAN uses a word-level encoder to encode the information of each utterance in the context into a hidden vector. Then, when generating each word, the hierarchical attention mechanism will focus on the important parts within and between the utterances with word-level attention and utterance-level attention, respectively. With these two attention levels, HRAN works in a bottom-up

way: word-level attention processes the hidden vector of the utterance and uploads it to the utterance-level encoder to form the hidden vector of the context. Utterance-level attention further processes the hidden vectors of the context into context vectors and uploads them to the decoder to generate words.

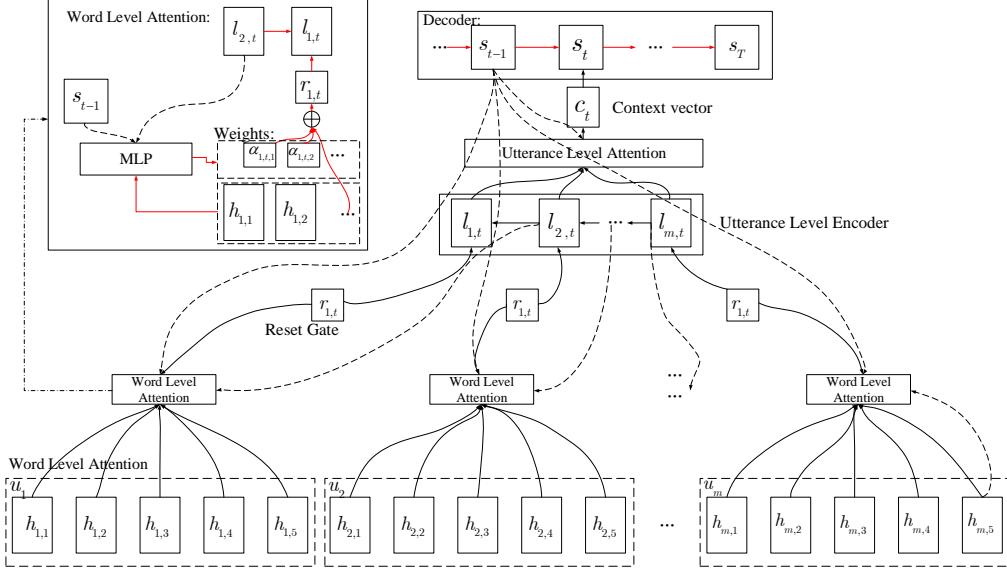


Figure 40: HRAN.

3.5.5. Hierarchical multiscale LSTM

The key element of the model is the introduction of a parameterized boundary detector, which outputs a binary value in each layer of the stacked RNN and learns when segmentation should end in a way that optimizes the overall goal. Whenever the boundary detector is turned on at layer l (i.e., when the boundary state is 1), the model regards it as the end of the segment corresponding to the potential abstraction level (such as a word or phrase) of the layer and send the summary representation of the detected segment to the upper layer ($l + 1$). Using boundary states, at each time step, each layer selects one of the following operations: UPDATE, COPY or FLUSH. The choice depends on (1) the boundary states at the current time step in the layer below z_t^{l-1} and (2) the boundary state at the previous time step in the same layer z_{t-1}^l .

Later, Junyoung Chung et al. described Hierarchical Multiscale Recurrent Neural Network (HM-RNN) based on LSTM update rules, and called this model Hierarchical Multiscale LSTM (HM-LSTM) [127] [128]. Consider the HM-LSTM model of L layer ($l = 1, \dots, L$), which performs the following updates at time step t at each layer l :

$$h_t^l, c_t^l, z_t^l = f_{HM-LSTM}^l (c_{t-1}^l, h_{t-1}^l, h_t^{l-1}, h_{t-1}^{l+1}, z_{t-1}^l, z_t^{l-1}) \quad (131)$$

Here, h and c represent the hidden state and the cell state, respectively. The implementation of

function $f_{HM-LSTM}^l$ is as follows. First, using the two boundary states z_{t-1}^l and z_t^{l-1} , the unit state is updated in the following way:

$$c_t^l = \begin{cases} f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l & \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 1 (\text{UPDATE}) \\ c_{t-1}^l & \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 0 (\text{COPY}) \\ i_t^l \odot g_t^l & \text{if } z_{t-1}^l = 1 (\text{FLUSH}) \end{cases} \quad (132)$$

Then get the hidden state in the following way:

$$h_t^l = \begin{cases} h_{t-1}^l & \text{if COPY} \\ o_t^l \odot \tanh(c_t^l) & \text{otherwise} \end{cases} \quad (133)$$

Among them, (f, i, o) are forget, input, and output gates, and g is the unit proposal vector. Note that, unlike LSTM, it is not necessary to calculate these gates and cell proposal values at every time step. For example, in the case of a COPY operation, these values do not need to be calculated, so the amount of calculation can be saved.

The advantages of this model include: (1) Capture the potential hierarchical structure in the sequence by encoding the temporal dependence of different time scales. (2) Effective hierarchical multi-scale representation can be learned to obtain better generalization performance.

3.6. Tree RNNs

Next, the various network architectures of Tree RNNs will be described in detail.

3.6.1. Tree LSTM

Tai KS et al. proposed a tree LSTM [129] [130] [131], which is mainly used in language analysis [132]. It can fully capture the characteristics of the vessel tree structure to solve the problem of anatomical labeling. In the tree LSTM, each LSTM unit can merge information from all subunit. The tree LSTM unit at node t includes input gate i_t , output gate o_t , memory gate m_t , memory unit c_t and hidden state h_t . The update of the gate vector and memory unit vector depends on the state of all subunits. The tree LSTM unit has a forgetting gate f_l for each child node l of node t . This allows the tree LSTM unit to selectively retain information from each child node. Therefore, the hidden state and memory unit of the node in the tree LSTM can be updated as:

$$H_t = \sum_{l \in C_t} h_l \quad (134)$$

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}H_t + b^{(i)}) \quad (135)$$

$$f_{tl} = \sigma(W^{(f)}x_t + U^{(f)}H_l + b^{(f)}) \quad (136)$$

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}H_t + b^{(o)}) \quad (137)$$

$$m_t = \tanh(W^{(m)}x_t + U^{(m)}H_t + b^{(m)}) \quad (138)$$

$$c_t = i_t \otimes m_t + \sum_{l \in C_t} f_{tl} \otimes c_l \quad (139)$$

$$h_t = o_t \otimes \tanh(c_t) \quad (140)$$

where x_t is the input at node t , and $W^{(i)}$, $W^{(f)}$, $W^{(o)}$ and $W^{(m)}$ are the weight matrices of input, forget, output and memory gate respectively. c_t is a subset of node t , σ represents the sigmoid function, and \otimes represents element wise multiplication.

3.6.2. Bidirectional Tree LSTM

Dan Wu extended the tree LSTM in both directions by adding an additional set of hidden state vectors in the Up-to-Down (UTD) direction [133]. Different from the Down-to-Up (DTU) direction, each hidden state in UTD-LSTM has only one parent node. In fact, the path from the root node to any node forms a sequential LSTM. At each node t , h_t calculation follows the bidirectional sequential LSTM model [134]. Denote the DTU hidden state vector as $h_t^{(\uparrow)}$, the UTD hidden state vector as $h_t^{(\downarrow)}$, and concatenate the final representation \tilde{h}_t :

$$\tilde{h}_t = h_t^{(\uparrow)} \oplus h_t^{(\downarrow)} \quad (141)$$

The softmax function is used to predict the probability \hat{p}_{y_j} of the segment label j from \tilde{h} . In the training stage, given the training set D , the training classifier maximizes the conditional log likelihood of the true class label y_j of the training sample in each vessel segment. The training objectives are defined as follows:

$$L_\Theta = -\frac{1}{N} \sum_{k=1}^N \log \hat{p}_{y_j} + \frac{\lambda}{2} (\|Z\|^2 + \|V\|^2) \quad (142)$$

where N is the number of vessel segments labeled in the training set, $\Theta = (Z, V)$ is the network weight, and λ is the regularization parameter. Z and V are the parameters of MLP and bidirectional tree LSTM respectively.

The advantages of this model include: (1) It can effectively learn the spatial and topological dependence of blood vessels, thereby capturing the characteristics of the tree structure. (2) Competitive performance can be produced on large data sets.

3.6.3. Attention with Tree Memory Network

For video question and answer questions, parse trees for different questions can have different topological structures. Hongyang Xue [135] and others used the StanfordParser [136] tool to construct grammar trees for sentences. Given the problem q , a parse tree can be constructed for it (for example, see Fig. 41). The corresponding tree structure memory network is shown in Fig. 42. The nodes of the memory network can be divided into three types: leaves, intermediate nodes and roots. Denote the n -dimensional state vector of node c_i as $h_i \in \mathbb{R}^n$, h_i is calculated from all its child nodes, and the child nodes are denoted as $C(i)$. Now describe how to calculate the states of each node in a bottom-up manner.

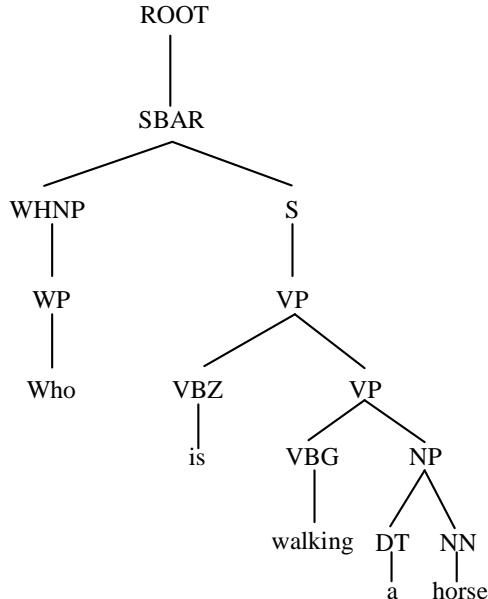


Figure 41: Stanford StanfordParser question-the original parse tree of who is walking a horse

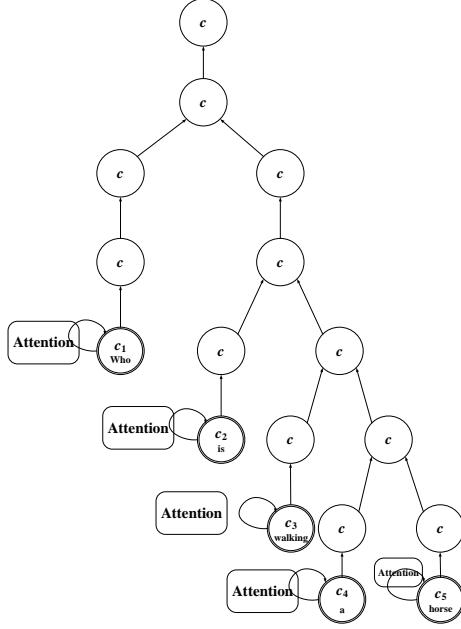


Figure 42: The Tree Memory Network (TreeMN) corresponds to the question in Fig. 41. Double-circle nodes are leaf nodes, and they are the words in the question. Each leaf node needs to calculate the attention to the video frame. The encoded features propagate from the leaf to the root.

Given a video $v = \{v_1, \dots, v_T\}$ and a question $q = \{q_1, \dots, q_N\}$, where v_i is the m -dimensional visual feature of the frame and q_i is the n -dimensional embedding of the mark, the purpose of the task is to generate the answer from the answer list A . In order to avoid symbol confusion, V is used to represent the matrix in $\mathbb{R}^{T \times m}$, where the i -th row of V is the eigenvector of the i -th frame.

1) Attention module: Attention module is similar to E-SA model [137] (a video question answering model). First, the video frame features are processed through bidirectional LSTM. The processed feature is $\{v_1, \dots, v_T\}$, where T is the number of frames. Given state h_i , the attention of the video frame is calculated as

$$\begin{aligned}
 h_{A,j} &= \tanh(W_Q h_i + W_V v_j + b_V) \\
 p &= \text{softmax}(W_p h_A) \\
 att_i &= \text{Attention}(h_i, V) = \sum_{k=1}^T p_k v_k
 \end{aligned} \tag{143}$$

Among them, $h_{A,j}$ is the j -th component of h_A , p_k is the k -th component of p , and $W_Q \in \mathbb{R}^{n \times n}$, $W_V \in \mathbb{R}^{n \times m}$ and $b_V \in \mathbb{R}^n$ are parameters that convert visual and text features into joint space. $W_p \in \mathbb{R}^{T \times n}$ is linear transformation.

2) Leaf nodes: Leaf nodes correspond to words in interrogative sentences. The state of a leaf node does not come from its child nodes (it has no child nodes), but is calculated based on the word

embedding and the attention of the video frame. Initialize the state h_j of leaf node c_j as $h_j = q_j$, and its new state is calculated by the following formula:

$$h_j = W_A \text{Attention}(h_j, V) + b_A + h_j \quad (144)$$

where $\text{Attention}(h_j, V)$ is the attention of the video frame guided by h_j , and $W_A \in \mathbb{R}^{n \times n}$, $b_A \in \mathbb{R}^n$ are the linear transformations shared by all leaf nodes. Denote the output of node c_j as o_j , and the output of leaf nodes as $o_j = \tanh(h_j)$.

3) Intermediate node: For an intermediate node corresponding to the intermediate representation of the sentence (such as NP , VP and NN), calculate the state h_i of the intermediate node c_i from its child node $C(i)$:

$$h_i = \sum_{j \in C(i)} W_B o_j + b_B \quad (145)$$

where $W_B \in \mathbb{R}^{n \times n}$ and $b_B \in \mathbb{R}^n$ are linear layers shared by all intermediate nodes. Its output is $o_i = \tanh(h_i)$.

Perform calculations from bottom to top in a recursive manner. Information is propagated from the leaves to the parent node, and finally to the root. At the root c_0 , the state h_0 can be calculated after calculating the intermediate node. Then place the linear layer, and then place the softmax layer for answer classification:

$$y = \text{softmax}(W h_0 + b) \quad (146)$$

where $W \in \mathbb{R}^{z \times n}$, $b \in \mathbb{R}^z$, z is the number of answers.

The advantages of this model include: (1) Improve word-level attention. (2) It can alleviate the shortcomings of E-SA model's poor performance on long problems.

3.6.4. Parse-tree-guided Reasoning Network

Give the tree structure layout of the dependency tree, the Parsed-Tree-Guided Reasoning Network (PTGRN) [138] module proposed by Qingxing Cao et al. is used in order on each word node to excavate visual evidence and integrate the features of its children from bottom to top, and then predict the final answer at the root of the tree, as shown in Fig. 43. Formally, each PTGRN module can be written as

$$\begin{aligned}
att_j &= f_a \left(\{m_{ij}^a\}_{e_{ij} \in E}, v, w \right) \\
v_j &= att_j * v \\
h_j &= f_h \left(\{m_{ij}^h\}_{e_{ij} \in E}, v_j \right) \\
h_j^{att} &= f_h^{att} \left(\{m_{ij}^a\}_{e_{ij} \in E}, att_j \right) \\
m_{jk}^h &= f_{e_{jk}}^h (h_j, q) \\
m_{jk}^a &= f_{e_{jk}}^a (h_j^{att}, q)
\end{aligned} \tag{147}$$

where, v is the image feature vector, w is the word embedding vector, q is the problem code, f_a is the attention module, att_j is the attention map, h_j is the hidden representation, E is a set of edges of the parse tree, and h_j^{att} is the attention map encoding, f_h and f_h^{att} are gated residual composition modules, and m_{ij}^a is the attention feature vector of child nodes.

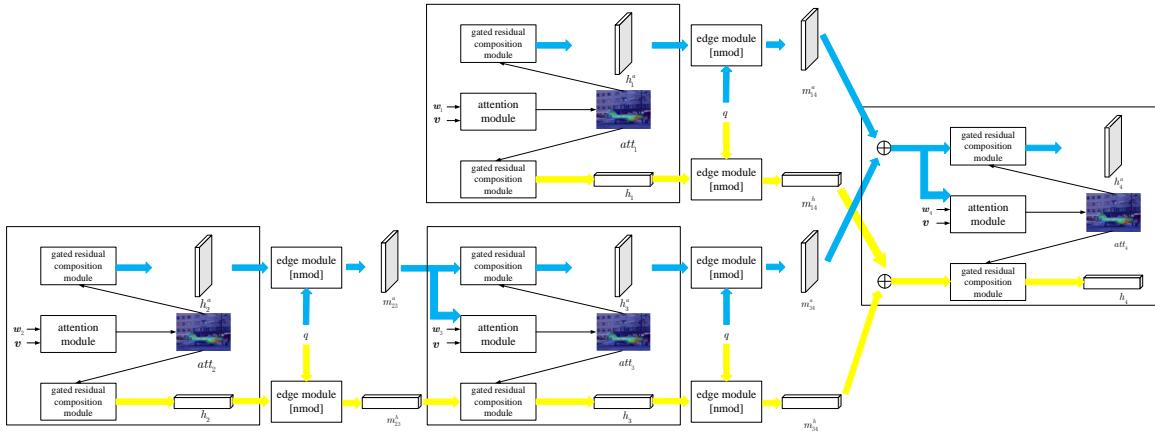


Figure 43: PTGRN method delivery path. Each PTGRN module consists of an attention module and two gated residual composition modules. Each node receives the encoded attention map and hidden feature vector of the child node, as well as image feature and word encoding. The attention module generates a new attention map based on image features, word encoding and previous attention regions given by child nodes. The gated residual composition module trains to evolve higher-level representations by dropping the features of the sub-modules and combining them with local visual characteristics. The edge module transform the outputs attention and hidden feature vectors according to the problem code and relationship type (nmod: noun modifier, dobj: direct object, nsubj: noun subject). The blue arrow indicates the propagation process of attention mapping, and the yellow arrow indicates the process of visual hidden representation.

Qingxing Cao et al. process each node by performing post-order traversal on the dependency tree. The edge type indicates whether the node acts as a modifier (a modifier can modify its parent node by referencing a more specific object) or as the subject/object of its predicate parent node. Therefore, based on the edge, the attention map and hidden representation of the node are passed to the parent node, so that the parent node can generate a more accurate attention map att_j or integrate the features of the child node to enhance the representation of a given predicate.

After propagating through all word nodes, the output message $[m_{root}^h, m_{root}^a]$ of the root node is used to predict the answer. Qingxing Cao et al. performed global maximum pooling on the encoded attention map m_{root}^a and connected it with m_{root}^h . This concatenated feature predicts the final answer y through a multilayer perceptron with three layers.

The model is formed by stacking a list of node modules following a tree structure layout. The weight is shared among all node modules. The entire model can be trained in an end-to-end manner using only the supervision signal y .

The advantages of this model include: (1) Starting from the problem, an explainable reasoning process is automatically executed on a general dependent parse tree, which greatly broadens the application field. (2) The workload becomes smaller and the model performance becomes better.

3.6.5. General TreeNet

The general TreeNet [139] consists of two parts: a token encoder and a semantic compositor. In the constituency tree, tokens (such as words and punctuation) are basic elements and leaf nodes. Since the dimension of the word vector is generally different from the dimension of the sentence vector, the token encoder is the part of semantic conversion between words and sentences. Then, the semantic compositor learns the representation of internal nodes on the constituency tree in a partial order.

- 1) Token encoder: Convert leaf nodes (word embedding) into semantic space of sentences.
- 2) Compositor: Learn the representation of the current node by merging the left sibling and right child of the node.

Fig. 44 shows the dependency graph calculated on the constituency tree. The dependency graph becomes a partial order. Zhou Cheng et al. believe that this order conforms to the cognitive rules.

For each sentence, the d -dimensional distribution vector is used as its semantic representation, and the token can be expressed as a one-hot vector, a random distribution vector of any dimension, or a pre-training vector from other works such as GloVe [140]. Therefore, general TreeNet must use a token encoder to learn a transformation function that encodes the distributed representation of tokens into sentence representations.

$$s_w = \text{Encoder}(w_{emb}) \quad (148)$$

where w_{emb} represents the word vector of the w -th word, and s_w is its corresponding vector in the sentence semantic space. *Encoder* stands for any neural network.

Unlike leaf nodes, each internal node has one or more sibling nodes and child nodes. Since the left internal node has no left sibling node, a zero initialization vector is used during calculation. For an internal node with multiple child nodes, the output of the right child node will be used as a representation of its child nodes. All child nodes of the same parent node are processed from left to right, so that one person can understand a sentence. The formula is as follows:

$$s_t = \text{Compositor} (s_{t-1}, s_c) \quad (149)$$

where s_c represents the only child node of the current node t (a label represents s_w) or the right child node of all child nodes, s_{t-1} represents the output of its left sibling node, and the *Compositor* function can be any network that learns how to merge them.

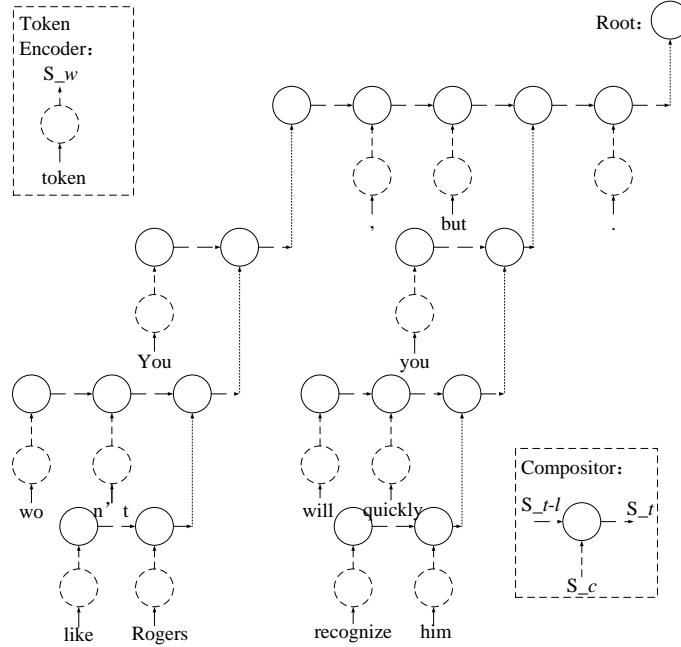


Figure 44: The calculation process of the constituency tree. First, encode the tokens from word space to sentence space. Second, calculate the internal nodes with the same parent node from left to right. At the same time, they incorporate the representation of the right child node that represents all descendants into themselves.

In TreeNet, the compositor function with calculation process plays a vital role in understanding sentences or parts of sentences. From the point of view of sibling nodes, the compositor processes them in sequence like a recurrent network because it assumes that the child node is its input and the left sibling node is its previous state. In terms of the relationship between the parent node and the child node, when the parent node applies the compositor to its left sibling node and right child node, they are updated in a recursive way. This feature of TreeNet is due to the simple calculation process and the intrinsic structure of the language.

The advantages of this model include: (1) Sentences can be captured structurally on the original unconstrained selection tree, where the number of child nodes can be arbitrary. (2) The network can learn on any tree. (3) Reduce the model parameters.

3.6.6. Multiplicative Tree LSTM

Encoding rich linguistic analysis introduces many different edge types or relationships between nodes, such as syntactic dependencies and semantic roles. This provides many possibilities for parameterization, but these methods are not considered in most existing syntax-aware LSTM methods, which only use the input node information.

Nam Khanh Tran et al. fill this gap by proposing a multiplicative tree LSTM (mTreeLSTM) [141] [142] (an extension of the tree LSTM model), and injecting relationship information into each node in the tree. mTreeLSTM introduces more fine-grained parameters based on the edge type. Inspired by multiplicative RNN [18], the hidden-hidden propagation in mTreeLSTM includes a separately learned transition matrix W_{hh} , and each possible edge type is learned by the following formula:

$$\tilde{h}_j = \sum_{k \in C(j)} W_{hh}^{r(j,k)} h_k \quad (150)$$

where $r(j, k)$ represents the connection type between node k and its parent node j . This parameterization is straightforward, but when there are many types of edges, a large number of parameters are required. For example, there are dozens of syntactic edge types, and each type corresponds to a Stanford dependent label.

In order to reduce the number of parameters and take advantage of the potential correlation between fine-grained edge types, Nam Khanh Tran et al. learned the embedding of edge types, and used the product of two dense matrices shared across edge types to factorize transition matrix $W_{hh}^{r(j,k)}$, which is a intermediate diagonal matrix with dependent edge type:

$$W_{hh}^{r(j,k)} = W_{hm} \text{diag}(W_{mr} e_{jk}) W_{mh} \quad (151)$$

where e_{jk} is the embedding of the edge type and is trained together with other parameters. The mapping from h_k to \tilde{h}_j is as follows:

$$\begin{aligned} m_{jk} &= (W_{mr} e_{jk}) \odot (W_{mh} h_k) \\ \tilde{h}_j &= \sum_{k \in C(j)} W_{hm} m_{jk} \end{aligned} \quad (152)$$

The calculation method of the gating unit (input gate i , output gate o and forget gate f) is the same as that of TreeLSTM.

mTreeLSTM can be applied to any tree of connection type between given nodes. For example, in the dependency tree, the semantic relationship $r(j, k)$ between nodes is provided by the dependency parser.

The advantages of this model include: (1) Allow child nodes to use different combination functions, which makes the model more expressive. (2) It can effectively handle long-term dependencies.

4. Other RNNs

Other RNNs are divided into three categories. Array structure RNNs is not to build a single hierarchical structure, but to build a more complex memory structure inside RNNs, creating a bottleneck function by sharing internal state, forcing the learning process to use memory cells to pool the similar or interchangeable content belonging to a hidden unit; Nested and Stacked RNNs are applied to sequence prediction problems, by establishing a temporal hierarchical structure to selectively access long-term information that is only contextually relevant; Memory RNNs use keys and values to convert visual space context into language space, and effectively capture the relationship between visual features and text descriptions. It can be applied to various document reading and question answering applications.

4.1. Array LSTM

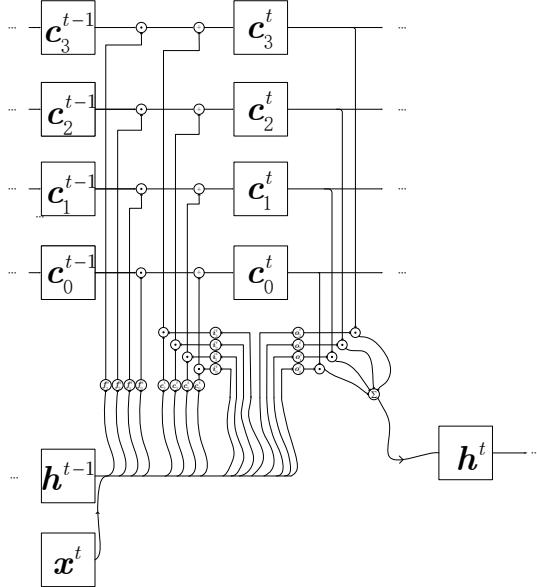


Figure 45: Array LSTM. Each hidden control unit has 4 memory units; the connections that have been modulated and being modulated, non-linearity and bias are omitted for brevity; when each hidden unit only has 1 memory unit, it becomes a standard LSTM (see 2.2.2).

The main idea of array LSTM is not to build a single layer hierarchical structure, (such as stacked LSTM, gated feedback RNNs), but to build a more complex memory structure in the RecurrentNNs unit [143] (similar ideas by constructing more complex transition functions within the layer). Rocki et al. want to create a *bottleneck* function by sharing internal state, forcing the learning process to use a memory unit belonging to a hidden unit to pool similar or interchangeable content [144]. A similar concept already exists in convolutional networks, i.e. spatial pooling. The hidden state should be treated as a complex unit pool to handle multiple possible sub-states. Fig. 45 and Eq.(12)-(17) describe the array LSTM architecture.

$$f_k^t = \sigma(W_{fk}x^t + U_{fk}h^{t-1} + b_{fk}) \quad (153)$$

$$i_k^t = \sigma(W_{ik}x^t + U_{ik}h^{t-1} + b_{ik}) \quad (154)$$

$$o_k^t = \sigma(W_{ok}x^t + U_{ok}h^{t-1} + b_{ok}) \quad (155)$$

$$c_k^t = f_k^t \odot c_k^{t-1} + i_k^t \odot \tilde{c}_k^t \quad (156)$$

$$h^t = \sum_k o_k^t \odot \tanh(c_k^t) \quad (157)$$

The advantages of this model include: (1) More memory units. Given the same amount of memory allocation, because the matrix between hidden units and gates is smaller, the array LSTM effectively uses more memory units. (2) More parallelism. The greater the number of units, the more independent elementwise calculations are produced. (3) More data locality. Units belonging to the same hidden unit will be related to each other in some way, which can improve the cache hit ratio. In addition, for more memory units in each unit, there may be connections from sparse hidden layers to hidden layers. (4) Approximate. The stochastic array memory is resilient to noise input.

4.2. Nested and Stacked RNNs

Next, the various network architectures of Nested and Stacked RNNs will be described in detail.

4.2.1. Stacked LSTM

Stacked LSTM is now a stable technology that challenges sequence prediction problems. The stacked LSTM [145] [146] [147] structure can be defined as an LSTM model composed of multiple LSTM layers. The upper LSTM layer provides sequence output rather than a single value output to the lower LSTM layer. Specifically, each input time step corresponds to one output, rather than all input time steps correspond to one output time step. The stacked LSTM formula is as follows:

$$i_t^l = \sigma(W_i^l h_t^{l-1} + U_i^l h_{t-1}^l + b_i^l)$$

$$f_t^l = \sigma(W_f^l h_t^{l-1} + U_f^l h_{t-1}^l + b_f^l)$$

$$\tilde{c}_t^l = \tanh(W_c^l h_t^{l-1} + U_c^l h_{t-1}^l + b_c^l)$$

$$o_t^l = \sigma(W_o^l h_t^{l-1} + U_o^l h_{t-1}^l + b_o^l)$$

$$c_t^l = i_t^l \odot \tilde{c}_t^l + f_t^l \odot c_{t-1}^l$$

$$h_t^l = o_t^l \odot \tanh(c_t^l) \quad (158)$$

where $t \in \{1, \dots, T\}$, $l \in \{1, \dots, T\}$, $W^l \in \mathbb{R}^{d_l \times d_{l-1}}$, $U^l \in \mathbb{R}^{d_l \times d_l}$ and $b^l \in \mathbb{R}^{d_l}$ are trainable parameters. $\sigma(\cdot)$ and $\tanh(\cdot)$ are sigmoid activation and hyperbolic tangent activation functions, respectively. Choi et al. assume that $h_t^0 = x_t \in \mathbb{R}^{d_0}$, where x_t is the t -th input of the network.

The input gate i_t^l and the forget gate f_t^l control the amount of information sent from \tilde{c}_t^l and c_{t-1}^l (pre-cell state and previous cell state) to the new cell state c_t^l . Similarly, the output gate o_t^l soft selects which part of the cell state c_t^l is used in the final hidden state h_t^l .

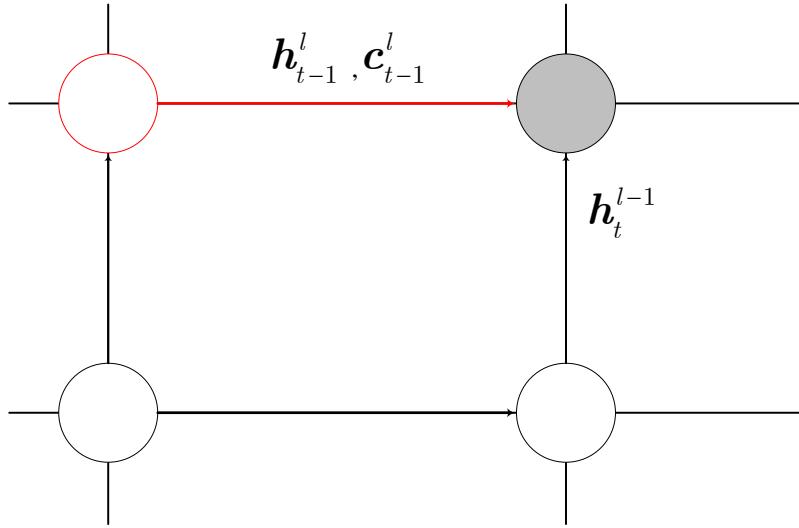


Figure 46: Stacked LSTM.

The advantages of this model include: (1) The amount of information to be transmitted can be adjusted not only in the horizontal recurrence, but also in the vertical connection. (2) The useful features extracted from the lower layer are effectively transferred to the upper layer.

4.2.2. Nested LSTM

The output gate in LSTM can encode an intuition that irrelevant memories at the current time step are still worth remembering. Nested LSTM (NLSTM) [148] [149] uses this intuition to create a temporal hierarchy of memories. Access to internal memory is gated in the same way, so long-term information that is only situationally relevant can be selectively accessed.

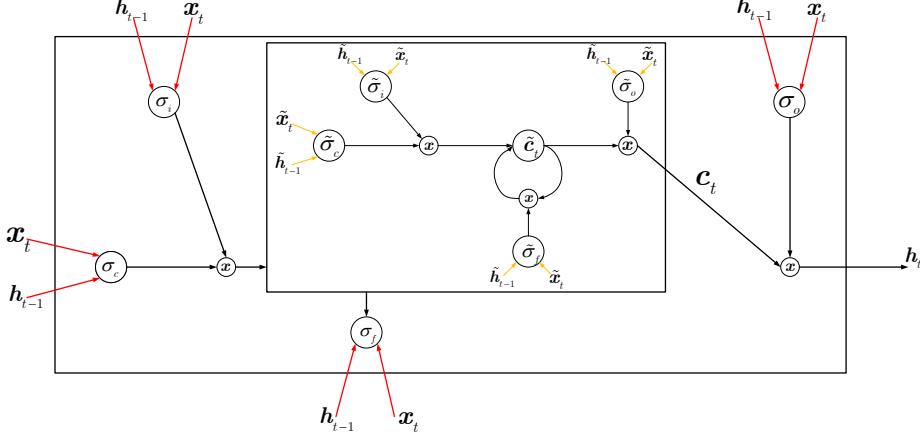


Figure 47: The NLSTM architecture.

NLSTM replaces the addition operation used to calculate c_t in the general LSTM with the learned state function, $c_t = m_t(f_t \odot c_{t-1}, i_t \odot g_t)$. Krueger et al. regard the state m of the function at time t as internal memory, and call the function that counts c_t and count m_{t+1} at the same time. Use the memory function as another LSTM memory unit, thus generating an NLSTM (see Fig. 47). The memory function can be another nested LSTM unit, which can be nested to any depth.

Given these structures, the input and hidden state of the memory function in NLSTM become:

$$\tilde{h}_{t-1} = f_t \odot c_{t-1}$$

$$\tilde{x}_t = i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \quad (159)$$

It should be particularly noted that if the memory function is addition, the system will become a general LSTM, because the unit update will become:

$$c_t = \tilde{h}_{t-1} + \tilde{x}_t \quad (160)$$

In the NLSTM structural variant, LSTM is used as a memory function, and the internal LSTM operates as follows:

$$\tilde{i}_t = \tilde{\sigma}_i(\tilde{x}_t \tilde{W}_{xi} + \tilde{h}_{t-1} \tilde{W}_{hi} + \tilde{b}_i)$$

$$\tilde{f}_t = \tilde{\sigma}_f(\tilde{x}_t \tilde{W}_{xf} + \tilde{h}_{t-1} \tilde{W}_{hf} + \tilde{b}_f)$$

$$\tilde{c}_t = \tilde{f}_t \odot \tilde{c}_{t-1} + \tilde{i}_t \odot \tilde{\sigma}_c(\tilde{x}_t \tilde{W}_{xc} + \tilde{h}_{t-1} \tilde{W}_{hc} + \tilde{b}_c)$$

$$\tilde{o}_t = \tilde{\sigma}_o(\tilde{x}_t \tilde{W}_{xo} + \tilde{h}_{t-1} \tilde{W}_{ho} + \tilde{b}_o)$$

$$\tilde{h}_t = \tilde{o}_t \odot \tilde{\sigma}_h(\tilde{c}_t) \quad (161)$$

The unit states of the external LSTM are updated to:

$$\tilde{c}_t = \tilde{h}_t \quad (162)$$

The advantages of this model include: (1) A clearer temporal hierarchies in the activation of its memory units. (2) The internal memory unit of the NLSTM forms an internal memory, which can only be accessed by other computing units through the external memory unit, realizing the form of a temporal hierarchical structure.

4.3. Memory RNNs

Next, the various network architectures of Memory RNNs will be described in detail.

4.3.1. Key-Value Memory Networks

Key-Value Memory Networks (KVMN) [150] [151] [152], the memory slots in the network are key-value pairs $(k_1, v_1), \dots, (k_T, v_T)$. Keys and values are used to convert visual space context into language space and effectively capture the relationship between visual features and text descriptions. The definition of memory, addressing and reading schema are summarized as follows [153]:

Keys (K): Use a CNN encoder to generate a visual context key k_i for each frame I_i of the video. These appearance feature vectors are combined with the sequential structure (video is an image sequence) through the RNN encoder. It can be seen from $k_i = h_i^e$ that the hidden state h_i^e at each time step is extracted as the key k_i .

Value (V): For each image frame I_i , a semantic embedding v_t^{sem} representing the meaning of a specific frame/key text is generated. It is very difficult to jointly learn visual semantic embedding in the Encoder-Decoder model, and the supervision signal only comes from the annotation description [154]. In order to alleviate this situation, it is necessary to pre-calculate the semantic embedding corresponding to each frame in the video. For example, suppose that v_i is obtained from the pre-trained model ψ , which simulates the visual and semantic embedding of the image given by $v_i = \psi(I_i)$ [155][156]. Now, for each frame I_i in the video, there is a key-value memory slot (k_i, v_i) .

Key Addressing: This corresponds to the soft attention mechanism used to assign a relevance probability α_t to each memory slot. These relevance probabilities are used for value reading.

Value Reading: The value reading of the memory slot is the weighted sum of the key value feature vectors $\phi_t(K)$ and $\phi_t(V)$ at each time step. $\phi_t(K)$ is used for key addressing at the next time step, and $\phi_t(V)$ is passed as input to the decoder RNN to generate the next word.

$$\phi_t(K) = \sum_{i=1}^T \alpha_i^{(t)} k_{i,\phi_t(V)} = \sum_{i=1}^T \alpha_i^{(t)} v_i \quad (163)$$

The advantages of this model include: (1) The key-value memory network is extended to a multi-modal environment to generate a natural description of the video. (2) Solve the problem of maintaining long-term temporal dependencies in the videos.

4.3.2. Recurrent Memory Network

It has been proven that RNNs can retain input information for a long period. However, the existing RNNs architecture is difficult to analyze at each time step which information is accurately retained in its hidden state, especially when the data has a complex underlying structure (this is common in natural languages). Based on this problem, Ke Tran et al. proposed a novel RNN architecture called Recurrent Memory Network (RMN) [157]. For linguistic data, RMN can not only determine which linguistic information will be retained over time and why this is the case, but also find dependencies in the data. RMN consists of two components: LSTM and Memory Block (MB). MB takes the hidden state of LSTM and uses the attention mechanism to compare it with the latest input [158][159]. Memory block is actually a variant of memory network [160]. Therefore, analyzing the attention weight of a trained model can provide a deep understanding of the information retained over time in the LSTM.

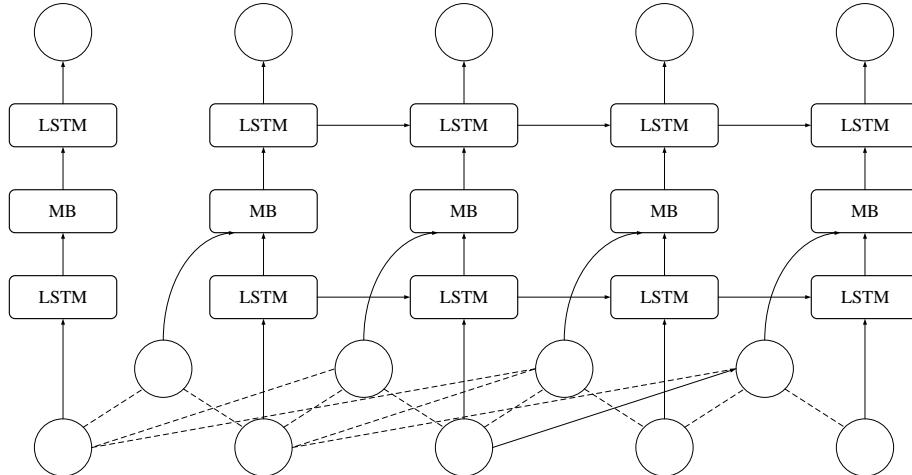


Figure 48: RMN structure.

The advantages of this model include: (1) Implicitly capture certain types of dependencies that are important for word prediction. (2) The function of RecurrentNNs is magnified, and it is helpful to understand its internal function, and can discover potential patterns in the data.

5. The Problems and Future Directions

RNNs try to combine with other networks to achieve more complex functions and achieve better results. They have made significant progress in recent years and are the current mainstream artificial intelligence learning direction. This article sorts out the main types of RNNs, and gives the model's construction process, advantages and disadvantages, and problems in the model. Although RNNs have great potential, there are also many challenges:

The disappearance and explosion of gradient: This is not only the problem of RNNs. Due to the chain rule and nonlinear activation function, almost all networks will have the problem of gradient disappearance and explosion. Many model variants that have emerged to solve this problem can only slightly alleviate the problem of gradient disappearance and explosion. Therefore, there is still a lot of space to completely solve the problem of gradient disappearance and explosion.

Dedicated to multi-task learning and general natural language understanding: all models applied throughout the paper are task-specific. But people are becoming more and more interested in multi-task learning in NLP. Since common concepts in the language will be applied to a single task, sharing information across tasks is intuitive. This can be seen as the first step in a universal, task-independent natural language understanding model. Although people have always been interested in using neural networks for multi-task learning, the improvement is still relatively small, and the best mechanism for sharing knowledge across tasks is unclear, so multi-task learning is an area worthy of in-depth research.

Generalization ability: Machine learning theory believes that a good model must have better generalization ability. From the viewpoints of model complexity and bias-variance trade-offs, theoretically discuss the learning mechanisms of various RNNs, enrich the theoretical foundation of the model, and improve the generalization ability of recursive and recurrent neural networks.

Application field expansion: The application range of RNNs is relatively small. How to apply the ideas and results of RNNs in common scenarios and how to accelerate the integration with these fields is the key direction for the further development of RNNs in the future. At present, RNNs are mainly used in the field of artificial intelligence, and applications in other fields need to be further developed.

Expansion of evaluation indicators: Although some results have been achieved in measuring the pros and cons of the model, there are still some problems. For some complex tasks such as video tracking, language and image analysis, it is difficult to use existing indicators to evaluate the model. Therefore, expanding the evaluation index can make the model more accurate.

Processing and expansion of available data sets: Although RNNs show good performance on the existing prescribed data sets, in practical applications, the instability and complexity of the data will greatly reduce the accuracy of the model, so the effective preprocessing of the data is a problem

worth studying.

Parameters and calculations: Although RNNs are efficient, the corresponding variants combine different structures to form a complex hybrid structure, resulting in an increase in model parameters and a substantial increase in calculations. Therefore, how to reduce model parameters and improve calculation efficiency under the condition of ensuring model accuracy is a problem worthy of in-depth study.

Model versatility: RNNs and their variants also have many limitations in the fields that can be applied, including the data set that can be processed is limited, so whether the model can be made relatively universal in a certain field is a problem worth studying.

6. Conclusion

In this paper, the branches of Recursive and Recurrent Neural Networks are classified in detail according to the network structure, training objective function and learning algorithm implementation. They are roughly divided into three categories: The first category is General RNNs, including RecursiveNNs and RecurrentNNs, LSTM, Convolutional RNNs, Differential RNNs, One-layer RNNs, High-Order RNNs, Highway Networks, Multi-Dimensional RNNs, Bidirectional RNNs, their basic subjects are RecurrentNNs and very classic variants LSTM and ordinary variants of LSTM. They use various methods to solve the problem that ordinary Recursive and Recurrent Neural Networks cannot learn large-scale sequence dependencies; The second category is Structured RNNs, including Grid RNNs, Graph RNNs, Temporal RNNs, Lattice RNNs, Hierarchical RNNs, Tree RNNs, which realize their functions in different situations through the combination of different structures; the third category is Other RNNs, including Array LSTM, Nested and Stacked RNNs, Memory RNNs. With the development and integration of various networks, many complex sequence, voice and image problems have been solved. From the summary of the above various models, it can be seen that the types of RNNs are quite rich and develop rapidly. Although all types of models have certain problems and limitations, it cannot be denied that with the further deepening of theoretical research and the further expansion of application fields, RNNs will surely become mainstream technologies in the field of artificial intelligence in the future.

References

- [1] Junyoung Chung, Çaglar Gülcöhre, KyungHyun Cho, Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv preprint arXiv: 1412.3555.
- [2] O. Irsay. Deep sequential and structural neural models of compositionality. PhD thesis, Cornell University, 2017.

- [3] Alex Graves:Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence 385, Springer 2012, ISBN 978-3-642-24796-5, pp. 1-131.
- [4] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training recurrent neural networks //Proceedings of the 30th International Conference on Machine Learning, 2013, Atlanta, GA, USA: JMLR Workshop and Conference Proceedings 28, JMLR.org 2013: 1310-1318.
- [5] Sepp Hochreiter, Y oshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
- [6] 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, March 1994, doi: 10.1109/72.279181.
- [7] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780,1997.
- [8] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. Neural Computation, vol. 12, pp. 2451–2471, 1999.
- [9] A. Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850,2013.
- [10] 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.
- [11] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, Chris Dyer. Depth-Gated LSTM. arXiv preprint arXiv: 1508.03790.
- [12] Xiaodan Liang, Liang Lin, Xiaohui Shen, Jiashi Feng, Shuicheng Yan, Eric P. Xing. Interpretable Structure-Evolving LSTM//Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA: IEEE Computer Society 2017: 2175-2184.
- [13] X. Liang, X. Shen, J. Feng, L. Lin, and S. Y an. Semantic object parsing with graph LSTM//Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I:125-143.
- [14] Jinyu Li, Changliang Liu, Yifan Gong. Layer Trajectory LSTM//Proceedings of Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India: ISCA 2018: 1768-1772.
- [15] Jinyu Li, Rui Zhao, Eric Sun, Jeremy H. M. Wong, Amit Das, Zhong Meng, Yifan Gong. High-Accuracy and Low-Latency Speech Recognition with Two-Head Contextual Layer Trajectory LSTM Model. ArXiv preprint ArXiv:2003.07482.

- [16] B. Li and T. N. Sainath. Reducing the computational complexity of two-dimensional LSTMs//Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017:964-968.
- [17] V. V anhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on CPUs//Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011. 2011.
- [18] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks//Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011: 1017-1024.
- [19] Ben Krause, Iain Murray, Steve Renals, Liang Lu. Multiplicative LSTM for sequence modelling. //Proceedings of the 5th International Conference on Learning Representations, 2017, Toulon, France: OpenReview.net 2017.
- [20] Niall McLaughlin, Jesús Martínez del Rincón, Paul C. Miller. Recurrent Convolutional Network for Video-Based Person Re-identification//Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA: IEEE Computer Society 2016: 1325-1334.
- [21] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping//2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, USA: 1735-1742.
- [22] Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, Zhihai He. Spatially supervised recurrent convolutional neural networks for visual object tracking// Proceedings of the IEEE International Symposium on Circuits and Systems, 2017, Baltimore, MD, USA: IEEE 2017: 1-4.
- [23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv preprint arXiv: 1502.03167.
- [24] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. ArXiv preprint arXiv: 1506.02640.
- [25] D. Kingma and J. Ba. Adam: A method for stochastic optimization//3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.
- [26] Jianxi Yang, Likai Zhang, Cen Chen, Yangfan Li, Ren Li, Guiping Wang, Shixin Jiang, Zeng Zeng. A hierarchical deep convolutional neural network and gated recurrent unit framework for structural damage detection. ArXiv preprint arXiv: 2006.01045.

- [27] Yang Feng, Yu Wang, Jiebo Luo. Video-based Person Re-Identification using Gated Convolutional Recurrent Neural Networks. ArXiv preprint arXiv: 2003.09717.
- [28] Avinash Madasu, Vijiini Anvesh Rao. Gated Convolutional Neural Networks for Domain Adaptation. ArXiv preprint arXiv: 1905.06906.
- [29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014.
- [30] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. ArXiv preprint arXiv:1411.4389.
- [31] Chang Wei Tan, Mahsa Salehi, Geoffrey Mackellar. Detecting Driver's Distraction using Long-term Recurrent Convolutional Network. ArXiv preprint arXiv: 2004.11839.
- [32] Xinyu Li, Yanyi Zhang, Jianyu Zhang, Shuhong Chen, Ivan Marsic, Richard A. Farneth, Randall S. Burd. Concurrent Activity Recognition with Multimodal CNN-LSTM Structure. arXiv preprint arXiv: 1702.01638.
- [33] Benmansour, A., Bouchachia, A., and Feham, M. Multioccupant activity recognition in pervasive smart home environments. *ACM Comput. Surv.* 48(3): 34:1-34:36 (2016).
- [34] Gori, I., Aggarwal, J.K., Matthies, L., and Ryoo, M.S. Multitype Activity Recognition in Robot-Centric Scenarios. *IEEE Robotics and Automation Letters*, 1(1), 593-600, 2016.
- [35] Yaguang Li, Rose Yu, Cyrus Shahabi, Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting // Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.
- [36] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic Flow Prediction With Big Data: A Deep Learning Approach. *IEEE Trans. Intell. Transp. Syst.* 16(2): 865-873 (2015).
- [37] Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. Deep learning: A generic approach for extreme condition traffic forecasting // Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017: 777-785.
- [38] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. Draw: A recurrent neural network for image generation. ArXiv preprint arXiv:1502.04623, 2015.

- [39] A. Grushin, D. D. Monner, J. A. Reggia, and A. Mishra. Robust human action recognition via long short-term memory// The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013: 1-8.
- [40] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. In Proceedings of the 15th international conference on Multimedia, pages 357–360. ACM, 2007.
- [41] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1–8. IEEE, 2008.
- [42] Vivek Veeriah, Naifan Zhuang, Guo-Jun Qi. Differential Recurrent Neural Networks for Action Recognition//Proceedings of the 2015 IEEE International Conference on Computer Vision, Santiago, Chile: IEEE Computer Society 2015: 4041-4049.
- [43] Naifan Zhuang, The Duc Kieu, Guo-Jun Qi, Kien A. Hua. Deep Differential Recurrent Neural Networks. ArXiv preprint arXiv:1804.04192.
- [44] Alex Graves, Abdel-rahman Mohamed, Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks//IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013:6645-6649.
- [45] Sitian Qin, Xiudong Yang, Xiaoping Xue, Jiahui Song. A One-Layer Recurrent Neural Network for Pseudoconvex Optimization Problems With Equality and Inequality Constraints. IEEE Trans Cybern. 2017;47(10):3063-3074.
- [46] G. Li, Z. Y an, and J. Wang. A one-layer recurrent neural network for constrained nonsmooth invex optimization. Neural Networks 50: 79-89 (2014).
- [47] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. Nonlinear Programming: Theory and Algorithms. New York, NY, USA: Wiley, 1993.
- [48] T. Q. Son and D. S. Kim. A new approach to characterize the solution set of a pseudoconvex programming problem. J. Comput. Appl. Math. 261: 333-340 (2014).
- [49] SM. Forti, P. Nistri, and M. Quincampoix. Generalized neural network for nonsmooth nonlinear programming problems. IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 51, no. 9, pp. 1741–1754, Sep. 2004.
- [50] SQ. Liu and J. Wang. A one-layer recurrent neural network for constrained nonsmooth optimization. IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 41, no. 5, pp. 1323–1333, Oct. 2011.

- [51] SW. Bian and X. Xue. Neural network for solving constrained convex optimization problems with global attractivity. *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 3, pp. 710–723, Mar. 2013.
- [52] Elias B. Kosmatopoulos, Marios M. Polycarpou, Manolis A. Christodoulou, Petros A. Ioannou. High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 1995, 6(2):422-431.
- [53] Chao Zhang, Philip Woodland. High Order Recurrent Neural Networks for Acoustic Modelling. ArXiv preprint arXiv: 1802.08314.
- [54] J. J. Hopfield. Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons. *Proceedings of the National Academy of Sciences*, 1984, 81(10):3088-3092.
- [55] Srivastava, Rupesh K, Greff, Klaus, and Schmidhuber, Juergen. Training very deep networks//*Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, December 7-12, Montreal, Quebec, Canada: 2377-2385.
- [56] Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber. Highway Networks. ArXiv preprint arXiv:1505.00387.
- [57] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M.Jose. Graph Highway Networks. ArXiv preprint arXiv: 2004.04635.
- [58] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, Jürgen Schmidhuber. Recurrent Highway Networks//*Proceedings of the 34th International Conference on Machine Learning*, 2017, Sydney, NSW, Australia: *Proceedings of Machine Learning Research* 70, PMLR 2017: 4189-4198.
- [59] Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. ArXiv preprint arXiv:1406.1078.
- [60] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R, and Schmidhuber, J. LSTM: A Search Space Odyssey. ArXiv preprint arXiv:1503.04069.
- [61] Jozefowicz, Rafal, Zaremba, Wojciech, and Sutskever, Ilya. An empirical exploration of recurrent network architectures//*Proceedings of the 32nd International Conference on Machine Learning*, ICML 2015, Lille, France, 6-11 July 2015: 2342-2350.
- [62] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yao, Sanjeev Khudanpur, James R. Glass. Highway long short-term memory RNNS for distant speech recognition//*2016 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP 2016, Shanghai, China, March 7-11, 2016: 4470-4474.

ence on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016:5755-5759.

- [63] Gakuto Kurata, Bhuvana Ramabhadran, George Saon, Abhinav Sethy. Language Modeling with Highway LSTM. ArXiv preprint arXiv: 1709.06436.
- [64] Hassan Taherian. End-to-end attention-based distant speech recognition with Highway LSTM. ArXiv preprint arXiv: 1610.05361.
- [65] A. Graves, N. Jaitly, and A. Mohamed. Hybrid speech recognition with deep bidirectional LSTM//2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013:: 273-278.
- [66] K. Chen, Z.-J. Yan, and Q. Huo. Training deep bidirectional lstm acoustic model for lvcsr by a context-sensitive-chunk bptt approach//INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015: 3600-3604.
- [67] Wei-Ning Hsu, Yu Zhang, Ann Lee, James R. Glass. Exploiting Depth and Highway Connections in Convolutional Recurrent Deep Neural Networks for Speech Recognition//Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016:395-399.
- [68] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. Convolutional, long short-term memory, fully connected deep neural networks//2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015: 4580-4584.
- [69] Su-Lin Wu, Brian Kingsbury, Nelson Morgan, Steven Greenberg. Incorporating information from syllable-length time scales into automatic speech recognition// Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98, Seattle, Washington, USA, May 12-15, 1998:721-724.
- [70] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks// Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States:190-198.
- [71] Parnia Bahar, Christopher Brix, Hermann Ney. Towards Two-Dimensional Sequence to Sequence Model in Neural Machine Translation//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium: Association for Computational Linguistics 2018: 3009-3015.

- [72] Gundram Leifert, Tobias Strauß, Tobias Grüning, Welf Wustlich, Roger Labahn. Cells in Multi-dimensional Recurrent Neural Networks. ArXiv preprint arXiv: 1412.2620.
- [73] Alex Graves, Jürgen Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks//Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada: Curran Associates, Inc. 2009: 545-552.
- [74] Alex Graves, Santiago Fernandez, Juergen Schmidhuber. Multi-Dimensional Recurrent Neural Networks. ArXiv preprint arXiv: 0705.2011.
- [75] Mike Schuster, Kuldip K. Paliwal. Bidirectional recurrent neural networks// Signal Processing, IEEE Transactions on, vol. 45, no. 11, pp.2673–2681, 1997.
- [76] Joseph Marvin Imperial, Jeyrome Orosco, Shiela Mae Mazo, Lany Maceda. Sentiment Analysis of Typhoon Related Tweets using Standard and Bidirectional Recurrent Neural Networks. ArXiv preprint arXiv: 1908.01765.
- [77] Ozan İrsoy, Claire Cardie. Bidirectional Recursive Neural Networks for Token-Level Labeling with Structure. ArXiv preprint arXiv: 1312.0493.
- [78] Christoph Goller, Andreas Küchler. Learning task-dependent distributed representations by back-propagation through structure//Proceedings of International Conference on Neural Networks (ICNN'96), Washington, DC, USA, June 3-6, 1996:347-352.
- [79] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection//Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain:801-809.
- [80] Wei-Ning Hsu, Yu Zhang, James R. Glass. A prioritized grid long short-term memory RNN for speech recognition//Proceedings of the 2016 IEEE Spoken Language Technology Workshop, San Diego, CA, USA: IEEE 2016: 467-473.
- [81] Sirin Haddad, Siew Kei Lam. Graph2Kernel Grid-LSTM: A Multi-Cued Model for Pedestrian Trajectory Prediction by Learning Adaptive Neighborhoods. ArXiv preprint arXiv: 2007.01915.
- [82] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. ArXiv preprint arXiv:1507.01526.
- [83] Florian L. Kreyssig, C. Zhang, Philip C. Woodland. Improved Tdnns Using Deep Kernels and Frequency Dependent Grid-RNNS//Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing, Calgary, AB, Canada: IEEE 2018: 4864-4868.

- [84] G. Saon, H. Soltau, A. Emami, and M. Picheny. Unfolded recurrent neural networks for speech recognition//INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014:343-347.
- [85] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, Xavier Bresson. Structured Sequence Modeling with Graph Convolutional Recurrent Networks//Neural Information Processing - 25th International Conference, 2018, Siem Reap, Cambodia, Proceedings, Part I: Lecture Notes in Computer Science 11301, Springer 2018: 362-373.
- [86] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. ArXiv preprint arXiv:1511.05493.
- [87] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 2009.
- [88] Michaël Defferrard, Xavier Bresson, and Pierre V andergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. ArXiv preprint arXiv: 1606.09375.
- [89] Yuan Yuan, Xiaodan Liang, Xiaolong Wang, Dit-Yan Yeung, Abhinav Gupta. Temporal Dynamic Graph LSTM for Action-Driven Video Object Detection// IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017:1819-1828.
- [90] Rohan Chandra, Tianrui Guan, Srujan Panuganti, Trisha Mittal, Uttaran Bhattacharya, Aniket Bera, Dinesh Manocha. Forecasting Trajectory and Behavior of Road-Agents Using Spectral Clustering in Graph-LSTMs. ArXiv preprint arXiv: 1912.01118.
- [91] Linfeng Song, Yue Zhang, Zhiguo Wang, Daniel Gildea. N-ary Relation Extraction using Graph State LSTM. ArXiv preprint arXiv: 1808.09101.
- [92] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, Wen-tau Yih. Cross-Sentence N-ary Relation Extraction with Graph LSTMs. ArXiv preprint arXiv: 1708.03743.
- [93] Vicky Zayats, Mari Ostendorf. Conversation Modeling on Reddit using a Graph-Structured LSTM. ArXiv preprint arXiv: 1704.02080.
- [94] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks//Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29:369-376.
- [95] Chuanlei Zhai, Zhineng Chen, Jie Li, Bo Xu. Chinese Image Text Recognition with BLSTM-CTC: A Segmentation-Free Method//Pattern Recognition - 7th Chinese Conference, CCPR 2016, Chengdu, China, November 5-7, 2016, Proceedings, Part II:525-536.

- [96] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition//IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015:1110-1118.
- [97] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis//2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016: 1010-1019.
- [98] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, Xiaohui Xie. Co-occurrence Feature Learning for Skeleton based Action Recognition using Regularized Deep LSTM Networks. ArXiv preprint arXiv: 1603.07772.
- [99] Jun Liu, Amir Shahroudy, Dong Xu, Alex C. Kot, Gang Wang. Skeleton-Based Action Recognition Using Spatio-Temporal LSTM Network with Trust Gates. ArXiv preprint arXiv: 1706.08276.
- [100] Zhen Zhou, Yan Huang, Wei Wang, Liang Wang, Tieniu Tan. See the Forest for the Trees: Joint Spatial and Temporal Recurrent Neural Networks for Video-Based Person Re-identification//Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, USA, July 21-26, 2017: 6776-6785.
- [101] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator// IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015:3156-3164.
- [102] N. McLaughlin, J. Martinez del Rincon, and P. Miller. Recurrent convolutional network for video-based person reidentification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
- [103] Yue Zhang, Jie Yang. Chinese NER Using Lattice LSTM//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, Volume 1: Long Papers: Association for Computational Linguistics 2018: 1554-1564.
- [104] Hao Wang, Bing Wang, Jianyong Duan, Jiajun Zhang. Chinese Spelling Error Detection Using a Fusion Lattice LSTM. ArXiv preprint arXiv: 1911.10750.
- [105] Jie Yang, Yue Zhang, Shuailong Liang. Subword Encoding in Lattice LSTM for Chinese Word Segmentation. ArXiv preprint arXiv: 1810.12594.
- [106] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer. Neural Architectures for Named Entity Recognition//NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016: 260-270.

- [107] Jacob Buckman, Graham Neubig. Neural Lattice Language Models. *Trans. Assoc. Comput. Linguistics* 6: 529-541 (2018).
- [108] Chao-Wei Huang, Yun-Nung Chen. Learning Spoken Language Representations with Neural Lattice Language Modeling. ArXiv preprint arXiv: 2007.02629.
- [109] Graham Neubig, Masato Mimura, Shinsuke Mori, and Tatsuya Kawahara. Learning a language model from continuous speech//INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010:1053-1056.
- [110] Pierre Dupont and Ronald Rosenfeld. 1997. Lattice based language models. Technical report, DTIC Document.
- [111] Matthias Sperber, Graham Neubig, Jan Niehues, Alex Waibel. Neural Lattice-to-Sequence Models for Uncertain Inputs. ArXiv preprint arXiv: 1704.00559.
- [112] Chaitanya Ahuja, Louis-Philippe Morency. Lattice Recurrent Unit: Improving Convergence and Statistical Efficiency for Sequence Modeling//Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018:4996-5003.
- [113] Zhen-Hua Ling, Yang Ai, Yu Gu, Li-Rong Dai. Waveform Modeling and Generation Using Hierarchical Recurrent Neural Networks for Speech Bandwidth Extension. ArXiv preprint arXiv: 1801.07910.
- [114] Hendrik Schröter, Tobias Rosenkranz, Alberto N. Escalante-B., Pascal Zobel, Andreas Maier. Lightweight Online Noise Reduction on Embedded Devices using Hierarchical Recurrent Neural Networks. ArXiv preprint arXiv: 2006.13067.
- [115] Asier Mujika, Felix Weissberger, Angelika Steger. Decoupling Hierarchical Recurrent Neural Networks With Locally Computable Losses. ArXiv preprint arXiv: 1910.05245.
- [116] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio. SampleRNN: An unconditional end-to end neural audio generation model. ArXiv preprint arXiv:1612.07837.
- [117] D. Yu and M. L. Seltzer. Improved bottleneck features using pretrained deep neural networks//INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011: 237-240.

- [118] Y. Gu, Z.-H. Ling, and L.-R. Dai. Speech bandwidth extension using bottleneck features and deep recurrent neural networks// Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016:297-301.
- [119] Nan Rosemary Ke, Konrad Zolna, Alessandro Sordoni, Zhouhan Lin, Adam Trischler, Yoshua Bengio, Joelle Pineau, Laurent Charlin, Chris Pal. Focused Hierarchical RNNs for Conditional Sequence Processing. ArXiv preprint arXiv: 1806.04342.
- [120] Mingming Lu, Kunfang Zhang, Haiying Liu, Naixue Xiong. Graph Hierarchical Convolutional Recurrent Neural Network (GHCRNN) for Vehicle Condition Prediction. ArXiv preprint arXiv: 1903.06261.
- [121] Zhanglin Peng, Ruimao Zhang, Xiaodan Liang, Xiaobai Liu, Liang Lin. Geometric Scene Parsing with Hierarchical LSTM//Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016:3439-3445.
- [122] Philipp Harzig, Yan-Ying Chen, Francine Chen, Rainer Lienhart. Addressing Data Bias Problems for Chest X-ray Image Report Generation. ArXiv preprint arXiv: 1908.02123.
- [123] Vahab Mostafapour, Oğuz Dikenelli. Attention-Wrapped Hierarchical BLSTMs for DDI Extraction. ArXiv preprint arXiv: 1907.13561.
- [124] Chen Xing, Yu Wu, Wei Wu, Yalou Huang, Ming Zhou. Hierarchical Recurrent Attention Network for Response Generation//Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018:5610-5617.
- [125] Harshala Gammulle, Simon Denman, Sridha Sridharan, Clinton Fookes. Hierarchical Attention Network for Action Segmentation. ArXiv preprint arXiv: 2005.03209.
- [126] Ruizhe Li, Chenghua Lin, Matthew Collinson, Xiao Li, Guanyi Chen. A Dual-Attention Hierarchical Recurrent Neural Network for Dialogue Act Classification. ArXiv preprint arXiv: 1810.09154.
- [127] Junyoung Chung, Sungjin Ahn, Yoshua Bengio. Hierarchical Multiscale Recurrent Neural Networks//5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.
- [128] Ákos Kádár, Marc-Alexandre Côté, Grzegorz Chrupała, Afra Alishahi. Revisiting the Hierarchical Multiscale LSTM. ArXiv preprint arXiv: 1807.03595.

- [129] Dan Wu, Xin Wang, Junjie Bai, Xiaoyang Xu, Bin Ouyang, Yuwei Li, Heye Zhang, Qi Song, Kunlin Cao, Youbing Yin. Automated anatomical labeling of coronary arteries via bidirectional tree LSTMs. ArXiv preprint arXiv: 1801.07910.
- [130] Ruo Ando, Yoshiyasu Takefuji. A constrained recursion algorithm for batch normalization of tree-structured LSTM. ArXiv preprint arXiv: 2008.09409.
- [131] Jeena Kleenankandy, K. A. Abdul Nazeer. An enhanced Tree-LSTM architecture for sentence semantic modeling using typed dependencies. ArXiv preprint arXiv: 2002.07775.
- [132] Tai KS, Socher R, Manning CD. Improved semantic representations from tree-structured long short-term memory networks. ArXiv preprint arXiv: 1503.00075.
- [133] Zhiyang Teng, Yue Zhang. Bidirectional Tree-Structured LSTM with Head Lexicalization. ArXiv preprint arXiv: 1611.06788.
- [134] Graves A, Jaitly N, Mohamed A. Hybrid speech recognition with Deep Bidirectional LSTM//2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013: 273-278.
- [135] Hongyang Xue, Wenqing Chu, Zhou Zhao, Deng Cai. A Better Way to Attend: Attention with Trees for Video Question Answering. arXiv preprint arXiv:1909.02218.
- [136] D. Klein and C. D. Manning. Accurate unlexicalized parsing//Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan:423-430.
- [137] K.-H. Zeng, T.-H. Chen, C.-Y. Chuang, Y.-H. Liao, J. C. Niebles, and M. Sun. Leveraging video descriptions to learn video question answering//Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA:4334-4340.
- [138] Qingxing Cao, Xiaodan Liang, Bailin Li, Liang Lin. Interpretable Visual Question Answering by Reasoning on Dependency Trees. arXiv preprint arXiv:1809.01810.
- [139] Zhou Cheng, Chun Yuan, Jiancheng Li, Haiqin Yang. TreeNet: Learning Sentence Representations with Unconstrained Tree Structure//Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden:4005-4011.
- [140] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove. Global vectors for word representation//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL:1532-1543.

- [141] Nam Khanh Tran, Weiwei Cheng. Multiplicative Tree-Structured Long Short-Term Memory Networks for Semantic Representations//Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, *SEM@NAACL-HLT 2018, New Orleans, Louisiana, USA, June 5-6, 2018:276-286.
- [142] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks//Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011: 129-136.
- [143] Razvan Pascanu, Çağlar Gülcehre, Kyunghyun Cho, Yoshua Bengio. How to construct deep recurrent neural networks//2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.
- [144] Kamil Rocki. Recurrent Memory Array Structures. ArXiv preprint arXiv: 1607.03085.
- [145] Jihun Choi, Taeuk Kim, Sang-goo Lee. Cell-aware Stacked LSTMs for Modeling Sentences. ArXiv preprint arXiv: 1809.02279.
- [146] Xiaosu Tong, Che-Wei Huang, Sri Harish Mallidi, Shaun Joseph, Sonal Pareek, Chander Chandak, Ariya Rastrow, Roland Maas. Streaming ResLSTM with Causal Mean Aggregation for Device-Directed Utterance Detection. ArXiv preprint arXiv: 2007.09245.
- [147] Boris Mocialov, Graham Turner, Helen Hastie. Transfer Learning for British Sign Language Modelling. ArXiv preprint arXiv: 2006.02144.
- [148] Joel Ruben Antony Moniz, David Krueger. Nested LSTMs//Proceedings of The 9th Asian Conference on Machine Learning, 2017, Seoul, Korea, November 15-17, 2017:530-544.
- [149] Xiaolei Ma, Yi Li, Zhiyong Cui, Yinhai Wang. Forecasting Transportation Network Speed Using Deep Capsule Networks with Nested LSTM Models. ArXiv preprint arXiv: 1811.04745.
- [150] Arnav Kumar Jain, Abhinav Agarwalla, Kumar Krishna Agrawal, Pabitra Mitra. Recurrent Memory Addressing for Describing Videos// Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Honolulu, HI, USA: IEEE Computer Society 2017: 2200-2207.
- [151] Daya Guo, Akari Asai, Duyu Tang, Nan Duan, Ming Gong, Linjun Shou, Dixin Jiang, Jian Yin, Ming Zhou. Inferential Text Generation with Multiple Knowledge Sources and Meta-Learning. ArXiv preprint arXiv: 2004.03070.
- [152] Ghodai Abdelrahman, Qing Wang. Knowledge Tracing with Sequential Key-Value Memory Networks. ArXiv preprint arXiv: 1910.13197.

- [153] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston. Key-value memory networks for directly reading documents. ArXiv preprint arXiv:1606.03126.
- [154] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. ArXiv preprint arXiv:1603.03925.
- [155] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. ArXiv preprint arXiv:1502.03044.
- [156] J. Johnson, A. Karpathy, and L. Fei-Fei. Densecap. Fully convolutional localization networks for dense captioning. ArXiv preprint arXiv:1511.07571.
- [157] Ke M. Tran, Arianna Bisazza, Christof Monz. Recurrent Memory Networks for Language Modeling//NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016:321-331.
- [158] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. A recurrent neural network for image generation// Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015:1462-1471.
- [159] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.
- [160] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks// Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada:2440-2448.