

Overview Paper

Recurrent Neural Networks and Their Memory Behavior: A Survey

Yuanhang Su* and C.-C. Jay Kuo

University of Southern California, Los Angeles, CA, USA

ABSTRACT

After their inception in the late 1980s, recurrent neural networks (RNNs) as a sequence computing model have seen mushrooming interests in communities of natural language processing, speech recognition, computer vision, etc. Recent variations of RNNs have made breakthroughs in fields such as machine translation where machines can achieve human level quality. RNNs assisted speech recognition technology is providing services on subtitles for live streaming videos. In this survey, we will offer a historical perspective by walking through the early years of RNNs all the way to their modern forms, detailing their most popular architectural designs and, perhaps more importantly, demystify the mathematical aspect behind their memory behaviors.

Keywords: Recurrent neural networks, long short-term memory, gated recurrent unit, bidirectional recurrent neural networks, natural language processing.

1 Introduction

Since Elman's "Finding Structure in Time" [19], it has been long believed that the recurrent neural network (RNN) as a sequence learning system is

*Corresponding author: Yuanhang Su, suyuanhang@hotmail.com.

able to handle very long sequences by “encoding” the input into a vector called the hidden state. Applications of learning systems with such capability are omnipresent in our daily lives, including but not limited to spam email detection [53], text content analysis [58, 59], language modeling, machine translation, speech transcription, object tracking for surveillance systems [54, 57] and interactive systems such as automated customer services. More recently, Graves *et al.* [22] has reported the breakthroughs led by RNNs for speech recognition. Work on similar tasks can also be found in [5, 42, 47, 48, 68, 74–79]. Research in RNN-powered machine translation systems such as those described in [50, 52] has led to machine’s human level performance. RNNs’ ability to describe the content of images and videos has also been studied in [4, 67] respectively.

Being different from feed-forward neural network designs, RNNs have at least one cyclic connection that builds a path from a network node back to itself according to [29]. The seemingly simple definition disguises its sophistication in the training process, which requires unrolling of the model across time for gradient generation and model weight sharing. What adds to the puzzle is their memory mechanism after RNNs are trained. To demystify all those puzzles, we will walk through the early designs of RNN – the simple RNN (SRN) for motivation. We will then introduce the concept of time unrolling and elaborate RNN’s training method – the back-propagation through time (BPTT). Then, we will present RNN’s basic building blocks, such as the LSTM and the gated recurrent unit (GRU) as well as models built on top of them with the help of time unrolling. Finally, we examine RNN’s memory behavior.

Although there are many writings on this subject – such as those in [13, 21, 23, 30] – due to its popularity, we try to differentiate ourselves by walking through RNN’s early history and elaborating its macro vs. cell architectures for a full picture. We also offer a case study of using RNNs for neural-based architectural design to illustrate how RNNs can be applicable to real-world problems. Perhaps, the most important differentiating factor is our focus on the understanding of the memory behavior of RNNs, which is a rarely studied topic in the literature. We hope this work can be beneficial to readers for a deeper understanding on this subject.

2 Inception: Simple Recurrent Neural Networks

All RNN’s modern day achievements began with a humble beginning of nothing more than a repeating sequence generator. The very first Jordan’s recurrent neural model as described in [29] is tasked to produce patterns of “AAAB,” where A is a vector of $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and B is a vector $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The model’s graph is shown in Figure 1, where each circle denotes a neuron that takes weighted inputs

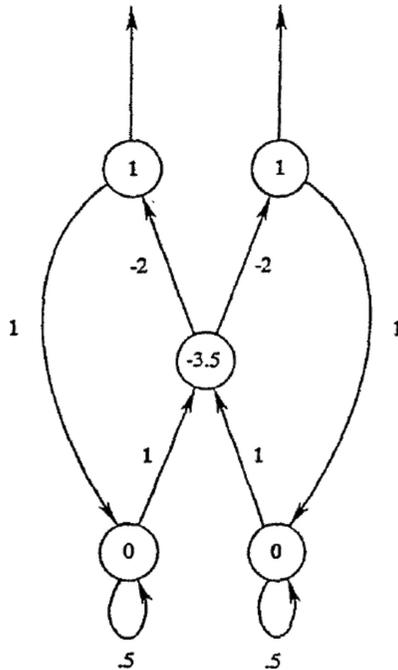


Figure 1: The graph of the sequence generating recurrent neural model by [29].

and computes the corresponding output based on its activation function. The value of an edge that connects one neuron to the other is called the “weight.” The bias terms are shown inside each neuron.

Throughout this survey, the notation for matrix or higher dimensional tensor, vector and scalar are bold-face italic, bold-face italic with straight line below and non-bold italic, respectively (e.g., \mathbf{m} , $\underline{\mathbf{v}}$, and s). We would omit the bias terms by including them in the corresponding weight matrices in the following equations. The multiplication between two equal-sized non-scalar variables in this paper is element-wise multiplication. Like feed-forward models, the output $\underline{\mathbf{h}}_t$ of each neuron with respect of input $\underline{\mathbf{x}}_t$ can be expressed as

$$\underline{\mathbf{h}}_t = f(\mathbf{W}_x \underline{\mathbf{x}}_t). \quad (1)$$

We follow the convention in [11]. That is, $\underline{\mathbf{h}}_t$ is the output ($\underline{\mathbf{y}}_t$ was frequently used in place of $\underline{\mathbf{h}}_t$ in early work), $\underline{\mathbf{x}}_t$ is the input, \mathbf{W}_x is the weight, and f is the activation function that is usually in form of sigmoid or hyperbolic tangent. We would argue that Hornik *et al.* [28]’s “squashing” functions are particular beneficial to the RNN’s numerical stability in the training process.

Jordan’s recurrent neural model has two state neurons (bottom), one hidden

neuron (middle) and two output neurons (top). The state neurons have linear activation function while the other ones have a binary threshold function that outputs 1 if the weighted input is positive and, 0, otherwise. The output of each neuron is shown in Table 1.

Table 1: Jordan’s recurrent model’s state at each time step.

Time	Input activation	Hidden activation	Output activation
0	$[0, 0]^T$	0	$[1, 1]^T \rightarrow A$
1	$[1, 1]^T$	0	$[1, 1]^T \rightarrow A$
2	$[1.5, 1.5]^T$	0	$[1, 1]^T \rightarrow A$
3	$[1.75, 1.75]^T$	1	$[0, 0]^T \rightarrow B$

Being different from feed-forward models, each neuron can find a path back to itself. The simple arrangement opens the door for a flexible modeling of complex temporal dependencies as opposed to the hidden Markov model (HMM) which is studied in [6, 7] or the Naive Bayes (NB) as described in [18, 32], where the temporal dependency concept stays insignificant or irrelevant since they do not model the order of events in a sequence explicitly or assume that events are only associated with each other through their immediate neighbors. Being aware of the potential benefit, Jordan proposed one of the early simple RNNs (SRN). It is shown in Figure 2, where neurons in the bottom left are input neurons while the ones in the bottom right with weights pointing to themselves are called “plan” neurons which take the previous outputs of the model and the previous outputs of themselves as their inputs.

Mathematically, the following equations describe Jordan’s SRN:

$$\underline{\mathbf{s}}_t = f_s(\mathbf{W}_h \underline{\mathbf{h}}_{t-1} + \mathbf{W}_s \underline{\mathbf{s}}_{t-1}), \quad (2)$$

$$\underline{\mathbf{c}}_t = f_c(\mathbf{W}_{in} \underline{\mathbf{X}}_t + \mathbf{W}_s \underline{\mathbf{s}}_t), \quad (3)$$

$$\underline{\mathbf{h}}_t = f_h(\mathbf{W}_c \underline{\mathbf{c}}_t), \quad (4)$$

where $\underline{\mathbf{X}}_t \in \mathbb{R}^M$, $\underline{\mathbf{s}}_t \in \mathbb{R}^N$, $\underline{\mathbf{c}}_t \in \mathbb{R}^N$, $\underline{\mathbf{h}}_t \in \mathbb{R}^N$, $\mathbf{W}_h \in \mathbb{R}^{N \times N}$, $\mathbf{W}_s \in \mathbb{R}^{N \times N}$, $\mathbf{W}_{in} \in \mathbb{R}^{N \times M}$, $\mathbf{W}_c \in \mathbb{R}^{N \times N}$, the activation function for the plan neurons are typically linear. The N is the number of neurons for the plan, hidden and output units. The M represents the dimensionality of the input at each time step.

To understand what this recurrence property implies and delivers in solving real-world problems, Elman conducted a series of experiments to demonstrate SRN’s learning capabilities on sequential patterns. One of such experiments is to predict the next word given the previous words in a sequence. This is called language modeling (LM), where each word is numericalized as a vector. One way of such numericalization is called *one-hot*. In the one-hot scheme, the

position of the “1” bit is only located in input’s entry position in the dictionary while all other positions take value “0.” The vector size is equal to the total number of elements in the dictionary. The behavior of the SRN for character prediction is shown in Figure 3. We see from the figure that the SRN can predict more accurately with more previous characters serving as the context for the target characters. This implies that the SRN is able to extract or “memorize” the information in the past. Furthermore, it is observed that the hidden state of SRN can group words of similar properties into close distanced clusters in a projected compact space as shown in Figure 4. Each word in the clustering analysis is represented by averaging the hidden unit activation vectors which are produced by that word. The distance is the L2 distance for clustering.

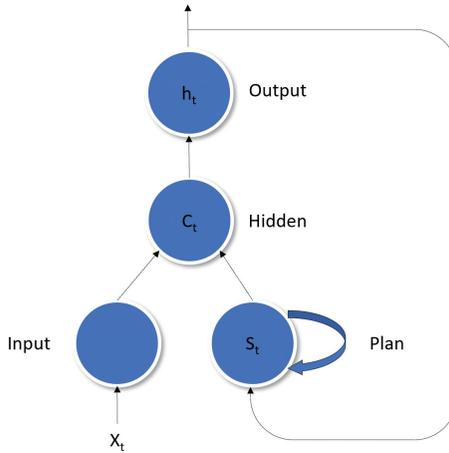


Figure 2: The graph of Jordan’s simple recurrent neural network.

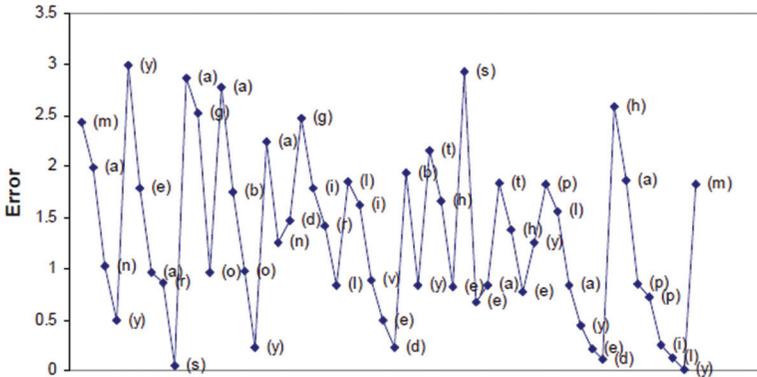


Figure 3: The root-mean-squared error of the SRN for character prediction by [19].

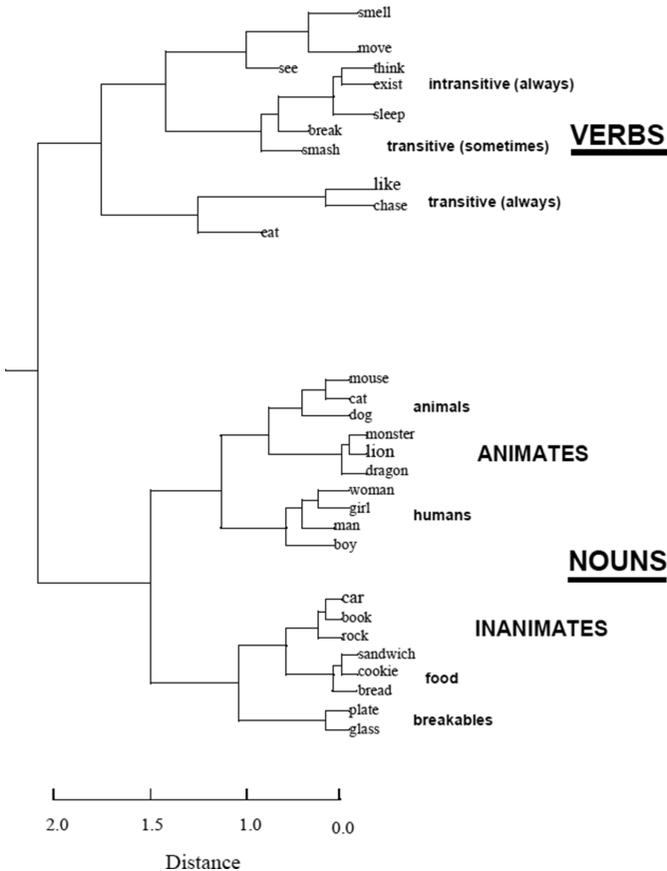


Figure 4: The hierarchical clustering result on SRN’s LM task by [19], where the distance is in the L2 space.

Although being conducted mostly experimentally, Elman’s results were convincing enough to deliver the following message. An artificial neural network (ANN) with carefully designed recurrent connection is able to “find structure in time.” A structure that encapsulates the information as a function of time/sequence, through which the information flow is always directional – the further away the information is from the current moment, the less observable it is. The significance of finding such a structure is that, for a large number of sequence learning problems including natural language processing (NLP) and video processing, we are concerned with finding semantic patterns (which is also studied in [16]) to solve the underlining problem by providing the sequential/temporal information. The capability of a learning system in grasping such patterns is thus crucial to its final performance.

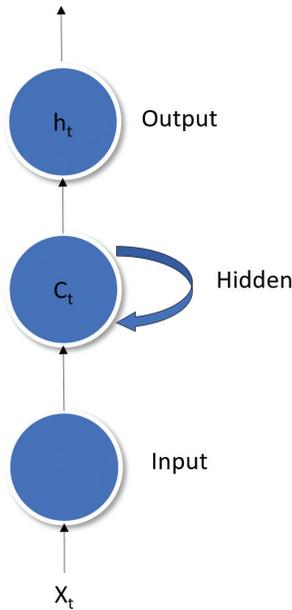


Figure 5: The graph of Elman's SRN.

Elman also proposed a simpler version of SRN without plan units as shown in Figure 5. Mathematically, it can be written as

$$\mathbf{c}_t = f_c(\mathbf{W}_{in}\mathbf{X}_t + \mathbf{W}_c\mathbf{c}_{t-1}), \quad (5)$$

$$\mathbf{h}_t = f_h(\mathbf{W}\mathbf{c}_t). \quad (6)$$

By comparing Elman's SRN with Jordan's, we see that the hidden state is a function of the current input as well as all previous inputs. Thus, such an architecture is ideal for building a probabilistic model to calculate the posterior of $p(\mathbf{h}_t | \{\mathbf{X}_i\}_{i=1}^t)$. At the end of this survey, we will show such a property holds the key to the analysis of RNN's memory. Before getting there, we would like to examine the RNN from a different perspective by unrolling its graph across time in the next section. This will help us understand how an RNN is trained.

3 Time Unrolling and Back Propagation Through Time

Time unrolling or unfolding is to create an equivalent RNN's graph without recurrent edges (see [10] for further details). This is made possible by presenting the input X as a sequence from the input at the first time step

(time step 1) all the way to the one at the last time step (time step T , T can go to positive infinity). Then, at each time step, the recurrent edges are redirected to the corresponding node in the next time step. Figure 6 shows the unrolling of Elman's SRN's hidden state. For a recurrent edge that does not point to the same node, we illustrate its time unrolling version in Figure 7.

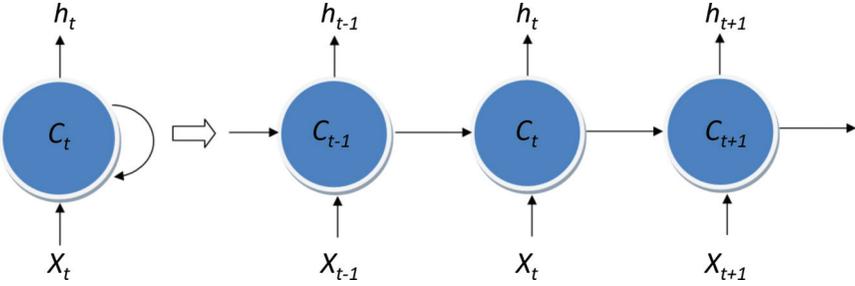


Figure 6: Time unrolling of a graph with edges pointing to the same node.

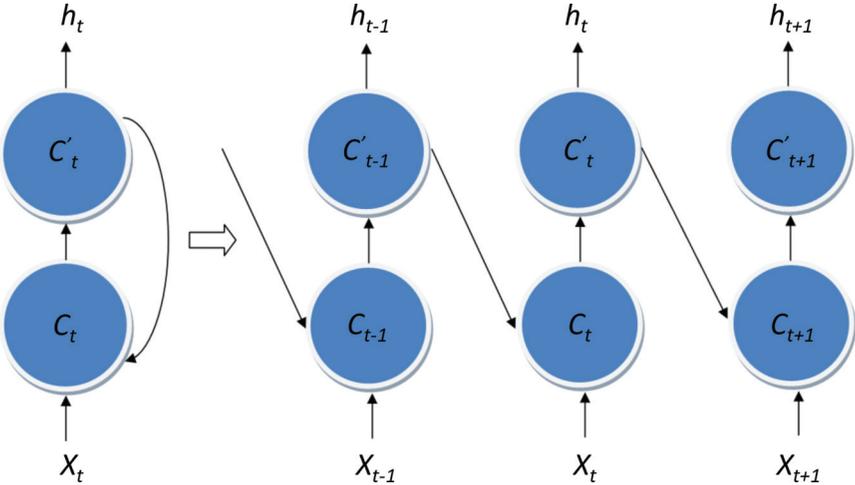


Figure 7: Time unrolling of a graph with edges pointing to a different node.

Time unrolling not only helps in better understanding of RNN's temporal dynamics but also serves as the implementation guidance since it simplifies the process into the creation of a feed-forward graph within a temporal loop. The graph at each time step is called RNN's **cell unit**. This terminology was first introduced in the LSTM work to describe the model's basic computing

unit and then extended to general RNNs. The seemingly feed-forward like structure of a cell often misleads people to an analogy of a deep layered CNN (further readings of interpreting deep CNN can be found in [31, 73]). The difference is that the cell at each time step has identical parameters (also called **weight sharing**). After all, they are the “mirror” of the same graph before unrolling.

Furthermore, the time unrolling/unfolding technique enables a particular way of RNN training called back-propagation through time (BPTT). Unlike feed-forward NN models, the error gradient is back-propagated not only to the model at the current time step but also to previous time steps all the way to the beginning of the sequence if the BPTT length is not truncated. If it is truncated, BPTT will stop at a pre-defined distance to the current time step t . Although time unrolling can allow the sequence to be infinitely long conceptually, BPTT only applies to a sequence with a pre-defined maximum length in practice because of the computational power constraint. This is implemented by either fixing the sequence length up to a maximum value in training or using truncated BPTT. The BPTT process is illustrated in Figure 8.

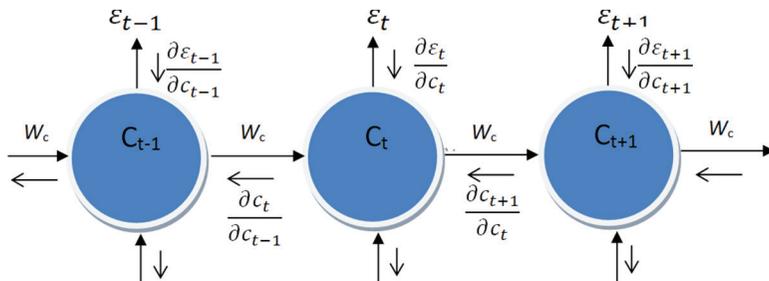


Figure 8: Illustration of the back-propagation through time (BPTT) process.

With BPTT, an RNN can be trained more efficiently by considering the temporal/sequential nature of the input signal explicitly. Since the error gradient now becomes a function of time, the shared model parameters updated by these gradients become a function of time as well. RNN models trained with BPTT can handle temporal inputs more easily. The problem arises in BPTT training is the so-called gradient vanishing/exploding problem. Although being different from feed-forward model, the training of RNNs shares a similar problem along the temporal dimension since the gradient back-propagates across time steps in a fashion similar to the one flowing through layers of feed-forward models. According to [27, 43], gradient vanishing/exploding happens when the input sequence becomes longer as shown in the paired comparison between ordinary back-propagation and BPTT in Equations (7) and (8), respectively. Let ϵ_t and \underline{c}_t be the training loss and the hidden state

at time step t respectively, \mathbf{W}_c the recurrent weight, and η the learning rate. Then, we have

$$\mathbf{W}_c = -\eta \frac{\partial \epsilon_t}{\partial \mathbf{W}_c} + \mathbf{W}_c, \quad (7)$$

$$\mathbf{W}_c = -\eta \sum_{\substack{1 \leq t \leq T, \\ 1 \leq k \leq t}} \left(\frac{\partial \epsilon_t}{\partial \mathbf{c}_t} \underbrace{\left(\prod_{j=k}^t \frac{\partial \mathbf{c}_j}{\partial \mathbf{c}_{j-1}} \right)}_{\substack{\text{gradient vanishing} \\ \text{explosion}}} \frac{\partial \mathbf{c}_k}{\partial \mathbf{W}_c} \right) + \mathbf{W}_c. \quad (8)$$

By adopting a similar approach developed by [27], Razvan *et al.* [43] examined the bounds of $|\frac{\partial \mathbf{c}_j}{\partial \mathbf{c}_{j-1}}|$ for SRN and drew two conclusions. Let λ be the largest eigenvalue of the Jacobian of $\frac{\partial \mathbf{c}_j}{\partial \mathbf{c}_{j-1}}$ and γ be the upper bound of its absolute value. It is sufficient for gradient vanishing to occur if $\lambda\gamma < 1$. It is necessary for gradient explosion to happen if $\lambda\gamma > 1$. The gradient vanishing/exploding problem makes RNNs difficult to train. This motivates the proposal of LSTM, which will be elaborated in the next section.

4 LSTM and Further Extensions

4.1 Cell Models: LSTM and GRU

The conversion of an RNN from a sequence generator to its powerful modern form is attributed to German computer scientists, Sepp Hochreiter and Jürgen Schmidhuber, on their work on the **long short-term memory (LSTM)** in the late 1990s. To solve the gradient vanishing/exploding problems, they introduced an internal recurrent structure called the *constant error carousel (CEC)* as shown in Figure 9.

Figure 9 shows the original proposal of LSTM with ϕ , σ and \otimes denote the hyperbolic tangent function, the sigmoid function and the multiplication operation, respectively. All of them operate in an element-wise fashion. The LSTM cell has an input gate, an output gate, and a CEC module. The central idea of CEC is to force the error gradient for the recurrent connection, $\frac{\partial \mathbf{c}_i}{\partial \mathbf{c}_{i-1}}, \forall i \in 1, \dots, t$, to unity so that the multiplicative term in Equation (8) does not converge to zero or diverge to infinity. It was argued that the presence of the input gate and the output gate is to ensure the error gradient does not flow from the current cell to other cells. Such argument was inconsistent with the ablation studies in [23], where it was observed that an LSTM without an output gate performs equally well in many cases. A similar observation was also given in [33]. The functionalities of LSTM's gates are still debatable nowadays.

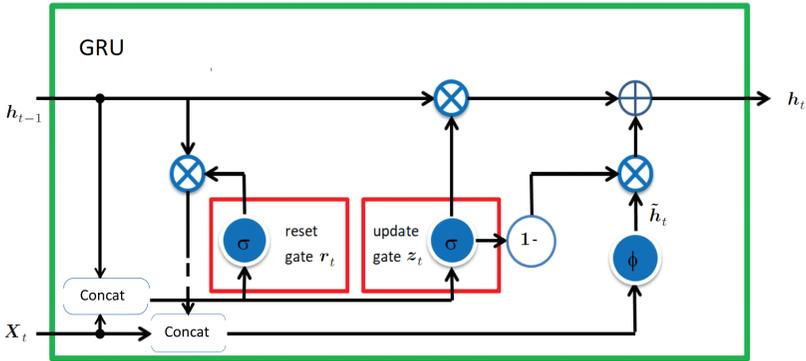


Figure 9: The diagram of the original LSTM cell.

One important gating design of an LSTM in its modern form is however missing in this earliest proposal. It is the forget gate. Soon after the original LSTM, the forget gate was introduced to deal with the so-called *hidden state overflow* problem. That is, when the absolute value of \underline{c}_t becomes very large, its hyperbolic-tangent-activated output would stay at $+1$ or -1 . This is incurred by the gradient enforcement of CEC, which makes the hidden state of an LSTM to increase or decrease to a prolonged period of time as shown in Figure 10 if the input sequence keeps a similar pattern.

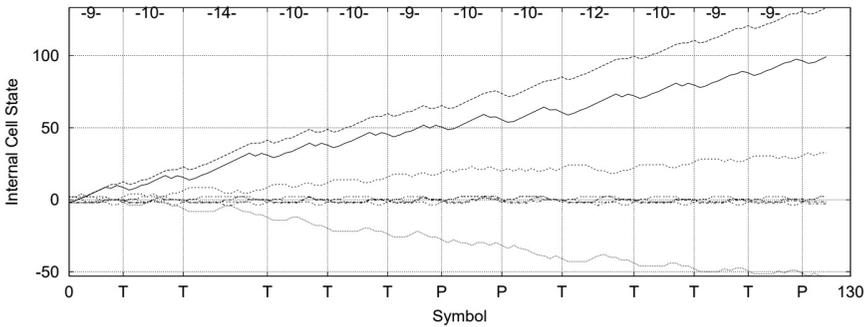


Figure 10: Illustration of the hidden state overflow problem by [20]. Starts of new sequences with repeating symbols are indicated by vertical lines labeled by the symbols (P or T) to be repeated in the sequences until the next ones start.

When the input consists of a series of repeating symbols “T” and “P,” we show the evolution of \underline{c}_t in Figure 10 as the input sequence becomes longer. The interval length between symbols in the horizontal axis denotes the length of the repeating symbols start from its left, the vertical line denotes the start

of a new series. As it can be seen, the hidden state of an LSTM increases or decreases regardless of the input of new series.

The forget gate allows the recurrent error gradient to be equal or less than one. This is equivalent to allowing gradient vanishing as long as the memory length is long enough for the specific task. It was observed in [20] that the forget gate is acting like a resetting mechanism. When a terminal signal is received, its activation will be set close to zero so that hidden state \underline{c}_t will be “refreshed” as if the previous elements in the sequence are forgotten as given in Figure 11. We will later argue that such a “resetting” phenomenon actually stems from the memory decay property of the forget gate, which would constrain LSTM’s memory capability.

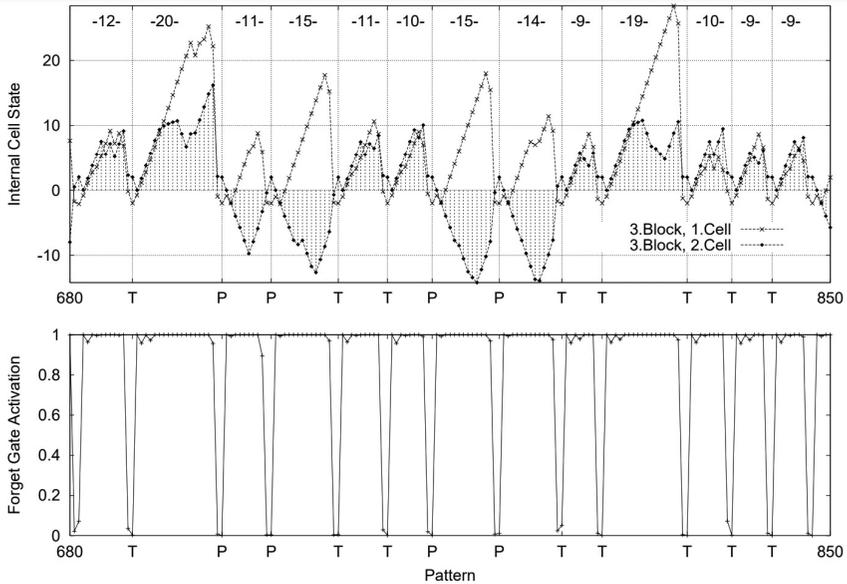


Figure 11: Resetting hidden states by introducing the forget gate in an LSTM by [20]. Top: Internal states s_t of the two cells in a LSTM network during a test stream presentation. Starts of new sequences with repeating symbols are indicated by vertical lines labeled by the symbols (P or T) to be repeated in the sequences until the next ones start. Bottom: simultaneous forget gate activations of the LSTM.

An LSTM with the forget gate is the most common form. Mathematically, it can be written as

$$\underline{c}_t = \sigma(\mathbf{W}_f \underline{\mathbf{I}}_t) \underline{c}_{t-1} + \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (9)$$

$$\underline{h}_t = \sigma(\mathbf{W}_o \underline{\mathbf{I}}_t) \phi(\underline{c}_t), \quad (10)$$

where $\underline{c}_t \in \mathbb{R}^N$, column vector $\underline{\mathbf{I}}_t \in \mathbb{R}^{(M+N)}$ is a concatenation of the current input, $\underline{\mathbf{X}}_t \in \mathbb{R}^M$, and the previous output, $h_{t-1} \in \mathbb{R}^N$ (i.e., $\underline{\mathbf{I}}_t^T = [\underline{\mathbf{X}}_t^T, h_{t-1}^T]$).

Furthermore, \mathbf{W}_f , \mathbf{W}_i , \mathbf{W}_o , and \mathbf{W}_{in} are weight matrices for the forget gate, the input gate, the output gate and the input, respectively. The detailed block diagram is shown in Figure 12.

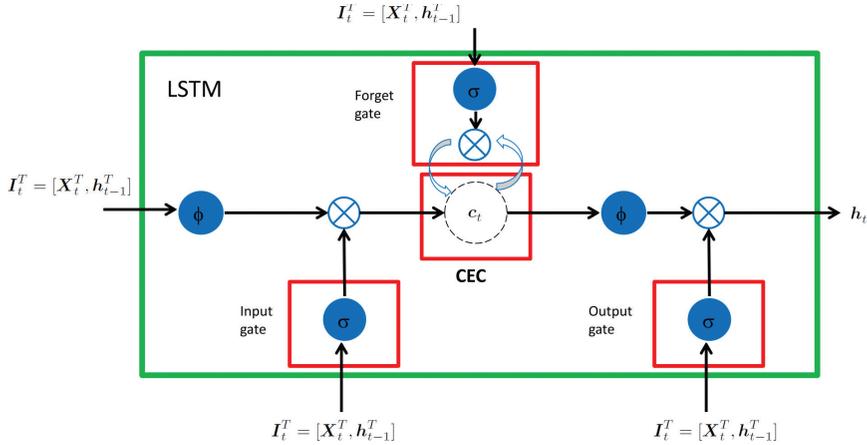


Figure 12: The diagram of a LSTM cell with the forget gate.

Another variation of LSTM, called the vanilla LSTM, is popularized by [23]. It has peephole connections from the CEC to the input, output and forget gates. Thus, Equations (9) and (10) can be written as

$$\underline{\mathbf{c}}_t = \sigma(\mathbf{W}_f \underline{\mathbf{I}}_t + \underline{\mathbf{p}}_f \underline{\mathbf{c}}_{t-1}) \underline{\mathbf{c}}_{t-1} + \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t + \underline{\mathbf{p}}_i \underline{\mathbf{c}}_{t-1}) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (11)$$

$$\underline{\mathbf{h}}_t = \sigma(\mathbf{W}_o \underline{\mathbf{I}}_t + \underline{\mathbf{p}}_o \underline{\mathbf{c}}_t) \phi(\underline{\mathbf{c}}_t), \quad (12)$$

where $\underline{\mathbf{p}}_i, \underline{\mathbf{p}}_f, \underline{\mathbf{p}}_o \in \mathbb{R}^N$ are called the peephole weights.

Based on the vanilla design, a convolutional LSTM (ConvLSTM) was proposed by [45] to deal with spatial-temporal inputs for problems like rainfall intensity prediction, where inputs are multiple values defined on a two-dimensional grid collected for a certain period of time. Unlike language problems where input tokens are mostly one dimensional vectors (see Section 4.3), the ConvLSTM takes in three dimensional (e.g., $C \times H \times W$) sequence of inputs, filtering them to get extracted feature maps in a fashion similar to CNNs by substituting matrix multiplications in Equations (11) and (12) with convolutions. As a result, we have the following equations for the ConvLSTM:

$$\mathbf{c}_t = \sigma(\mathbf{W}_f * \mathbf{I}_t + \mathbf{p}_f \mathbf{c}_{t-1}) \mathbf{c}_{t-1} + \sigma(\mathbf{W}_i * \mathbf{I}_t + \mathbf{p}_i \mathbf{c}_{t-1}) \phi(\mathbf{W}_{in} * \mathbf{I}_t), \quad (13)$$

$$\mathbf{h}_t = \sigma(\mathbf{W}_o * \mathbf{I}_t + \mathbf{p}_o \mathbf{c}_t) \phi(\mathbf{c}_t), \quad (14)$$

where $*$ denotes the convolution operation, \mathbf{I}_t , \mathbf{c}_t , and \mathbf{h}_t are all three dimensional tensors. The ConvLSTM design is largely functional. It allows

higher dimensional inputs and preserves LSTM’s memory capability. Before proceeding to the next topic, we would like to mention that the LSTM actually has many different forms, and only the most popular ones with proven efficacy were reviewed above.

Another popular RNN cell model is called the **GRU**. It was originally proposed for neural machine translation by Cho *et al.* in [12]. Its operations can be expressed as:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{X}_t + \mathbf{U}_z \mathbf{h}_{t-1}), \quad (15)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{X}_t + \mathbf{U}_r \mathbf{h}_{t-1}), \quad (16)$$

$$\tilde{\mathbf{h}}_t = \phi(\mathbf{W} \mathbf{X}_t + \mathbf{U}(\mathbf{r}_t \otimes \mathbf{h}_{t-1})), \quad (17)$$

$$\mathbf{h}_t = \mathbf{z}_t \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \tilde{\mathbf{h}}_t, \quad (18)$$

where \mathbf{X}_t , \mathbf{h}_t , \mathbf{z}_t , and \mathbf{r}_t denote the input, the hidden-state, the update gate and the reset gate vectors, respectively, and \mathbf{W}_z , \mathbf{W}_r , \mathbf{W} , are trainable weight matrices. Its hidden-state is also its output as given in Equation (18). Its diagram is shown in Figure 13, where “Concat” denotes the vector concatenation operation.

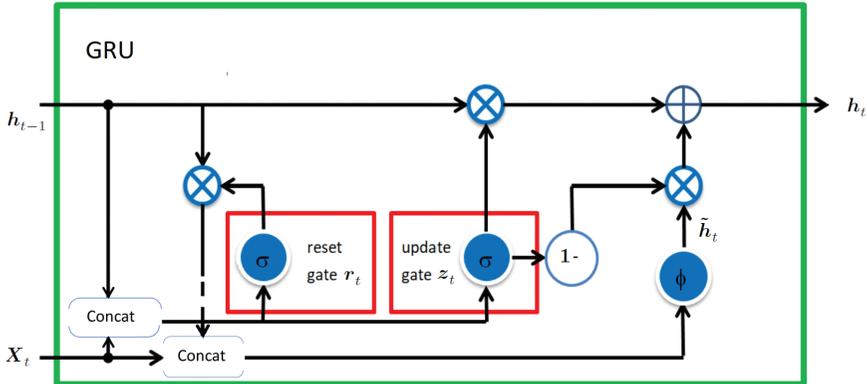


Figure 13: The diagram of a GRU cell.

As shown in the figure, for the same input \mathbf{X}_t and cell number N , a GRU cell uses fewer parameters than an LSTM (see Figure 12). An LSTM has four weight matrices (for the input, the input gate, the output gate, and the forget gate, respectively) while a GRU has only three weight matrices for the update gate, the reset gate and the weight connection for $\tilde{\mathbf{h}}$. In the original GRU paper by [12], it was argued that the reset gate is functioning like the forget gate in the LSTM and the update gate functions like a valve to control the flow of information from the previous inputs. However, we will show that a GRU is

actually the same as an LSTM in terms of their computing architecture with no fundamental difference in the next section.

It was observed in [13] that a GRU outperforms an LSTM in some cases yet no conclusive remarks were given. Similar observations were given in [30]. It was concluded in [23] that there is neither obvious advantage nor disadvantage of an LSTM as compared to a GRU. To sum up, there is no foregone conclusion whether the GRU outperforms the LSTM. A GRU converges faster than an LSTM for some particular RNN models.

4.2 Macro Models: BRNN, Seq2Seq and Deep RNN

A single RNN cell model is rarely used in practice due to its limited expressiveness in real-world problem modeling. Instead, more powerful RNN models are built upon these cells and used with different probabilistic models. One problem of interest is the sequence-in-sequence-out (SISO) problem or the sequence-to-sequence problem. That is, the RNN model predicts an output sequence, $\{\underline{\mathbf{Y}}_t\}_{t=1}^{T'}$, with $\underline{\mathbf{Y}}_t \in \mathbb{R}^N$, based on the input sequence, $\{\underline{\mathbf{X}}_t\}_{t=1}^T$, with $\underline{\mathbf{X}}_t \in \mathbb{R}^M$, where T and T' are lengths of the input and the output sequences, respectively.

One of the popular macro RNN models in solving the SISO problem is the **bidirectional RNN (BRNN)**, which was first studied in [44]. As the name indicates, a BRNN takes inputs in both forward and backward directions as shown in Figure 14. It has two RNN cells to take in the input. One takes the input in the forward direction while the other takes the input in the backward direction.

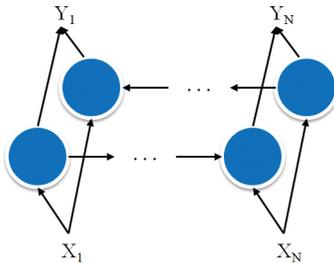


Figure 14: The diagram of a bidirectional RNN (BRNN).

The BRNN design is motivated by utilizing the input sequence fully if the future information ($\{\underline{\mathbf{X}}_i\}_{i=t+1}^T$) is accessible. This is especially helpful if the current output, $\underline{\mathbf{Y}}_t$, is also a function of future inputs. The conditional probability density function of a BRNN is in form of

$$P(\underline{\mathbf{Y}}_t | \{\underline{\mathbf{X}}_t\}_{t=1}^T) = \mathbf{W}^f \underline{\mathbf{p}}_t^f + \mathbf{W}^b \underline{\mathbf{p}}_t^b, \quad (19)$$

$$\hat{\underline{\mathbf{Y}}}_t = \operatorname{argmax}_{\underline{\mathbf{Y}}_t} P(\underline{\mathbf{Y}}_t | \{\underline{\mathbf{X}}_t\}_{t=1}^T), \quad (20)$$

where

$$\underline{\mathbf{p}}_t^f = P(\underline{\mathbf{Y}}_t | \{\underline{\mathbf{X}}_i\}_{i=1}^t), \quad (21)$$

$$\underline{\mathbf{p}}_t^b = P(\underline{\mathbf{Y}}_t | \{\underline{\mathbf{X}}_i\}_{i=t}^T), \quad (22)$$

and \mathbf{W}^f and \mathbf{W}^b are trainable weights, $\hat{\mathbf{Y}}_t$ is the predicted output element at time step t . Thus, the output is a combination of the density estimation of a forward RNN and the output of a backward RNN. Due to the bidirectional design, a BRNN can utilize the information of the entire input sequence to predict each individual output element. To show such treatment is helpful, one example is generating a sentence like “this is an apple” in language modeling. The word “an” is associated with its following word “apple.” It would be difficult in generating “an” before “apple” in a forward directional RNN model.

The **sequence-to-sequence (Seq2Seq) model** was first proposed for machine translation (MT) with GRU in [11]. Although it was referred to as the encoder-decoder model in [11], the encoder-decoder model encompasses more than the Seq2Seq model. It was developed to handle the situation when $T' \neq T$. It consists of two RNN cells. One is called the encoder while the other is called the decoder. We will have more discussion on the encoder-decoder model in Section 4.3.

One of early Seq2Seq RNN models in [61] is illustrated in Figure 15. As shown in Figure 15, the encoder (denoted by Enc) takes the input sequence of length T and generates its output $\underline{\mathbf{h}}_t^{\text{Enc}}$ and hidden state $\underline{\mathbf{c}}_t^{\text{Enc}}$, where $t \in \{1, \dots, T\}$. In the Seq2Seq model, encoder’s hidden state at time step T is used as the representation of the input sequence. The decoder utilizes the hidden state information to generate the output sequence of length T' by initializing its hidden state $\underline{\mathbf{c}}_1^{\text{Dec}}$ with $\underline{\mathbf{c}}_T^{\text{Enc}}$. Thus, the decoding process starts after the encoder has processed the entire input sequence. In practice, the input to the decoder at time step 1 is a pre-defined start decoding symbol. At the remaining time steps, the previous output $\underline{\mathbf{Y}}_{t-1}$ is used as the input. The decoder will stop the decoding process if a special pre-defined stopping symbol is generated.

As compared with the BRNN, the Seq2Seq model is not only advantageous in its capability of handling input/output sequences of different lengths but also more capable in generating aligned output sequences by feeding the previous predicted outputs back to its decoder explicitly so that the prediction of $\underline{\mathbf{Y}}_t$ can have more context, which allows the model to estimate the density function as

$$\underline{\mathbf{p}}_t = P(\underline{\mathbf{Y}}_t | \{\hat{\mathbf{Y}}_i\}_{i=1}^{t-1}, \{\underline{\mathbf{X}}_i\}_{i=1}^T), \quad (23)$$

$$\hat{\mathbf{Y}}_t = \operatorname{argmax}_{\underline{\mathbf{Y}}_t} \underline{\mathbf{p}}_t \forall t \in \{1, \dots, T'\}. \quad (24)$$

One example is the translation from a sentence in Chinese “你来自哪里” to English “where are you from” where “你” corresponds to “you,” “来自”

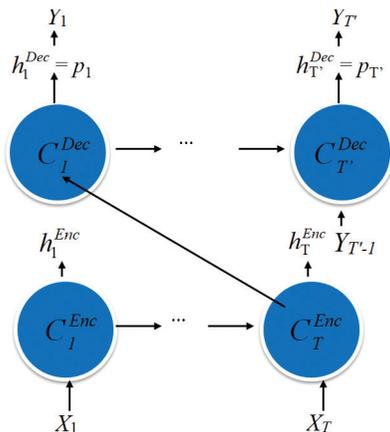


Figure 15: The diagram of the sequence-to-sequence model.

corresponds to “from,” “哪里” corresponds to “where,” and the word “are” has no corresponding Chinese alignment. So the word “are” is more pertinent to the word of “where” and “you” in the translated English sentence than to the source sentence in Chinese. Since the decoder of Seq2Seq has no bidirectional design, “are” cannot context on “you,” but nevertheless, “where” should give strong hint as to what word should be generated next.

To facilitate alignment, various attention mechanisms have been proposed for the Seq2Seq model. In [2, 66], additional weighted connections are introduced to connect the decoder to the hidden state of the encoder.

Although the Seq2Seq model (or an RNN in general) can take variable lengthed input sequences during inference. To train the RNN model, a fixed input/output length is used in practice since the BPTT is applied to an unrolled RNN model. Any sequence that is longer than the fixed length will be truncated and any shorter ones will be padded by pre-defined special symbol or a numericalized vector.

A **deep RNN** is an RNN model with a stack of multiple RNN cells. An example is shown in Figure 16. As discussed in Section 3, an RNN is itself a deep model over time. The deep RNN extends the model depth, making it a deep feed-forward model at each time step. The design of deep RNN models is still an ongoing topic. At present, there is no mature prototype. In [61], a Seq2Seq model with 4 layers of LSTM cells was reported to deliver the best performance for machine translation. In [71], the residue connection as described in [25] is employed in the deep design, which offers a deep Seq2Seq model with 9 layers of encoders and 8 layers of decoders. Four different models were proposed in [41] and shown in Figure 17. Yet, there is no conclusion about which model gives the best performance.

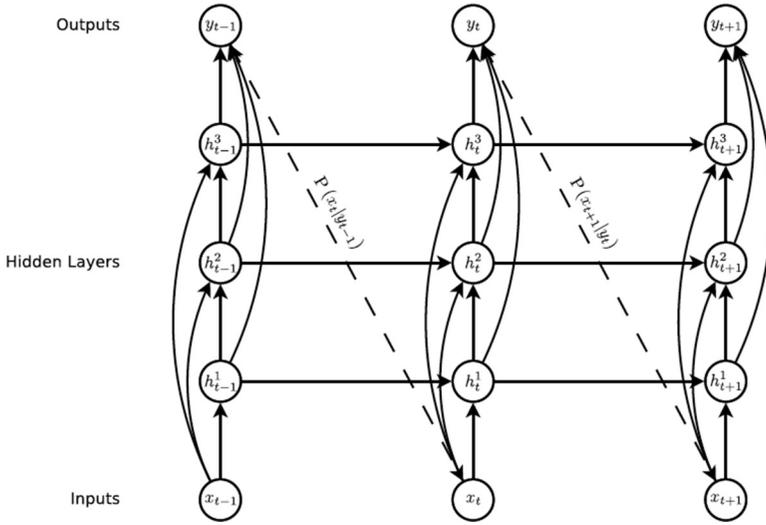


Figure 16: The diagram of a deep RNN by [41].

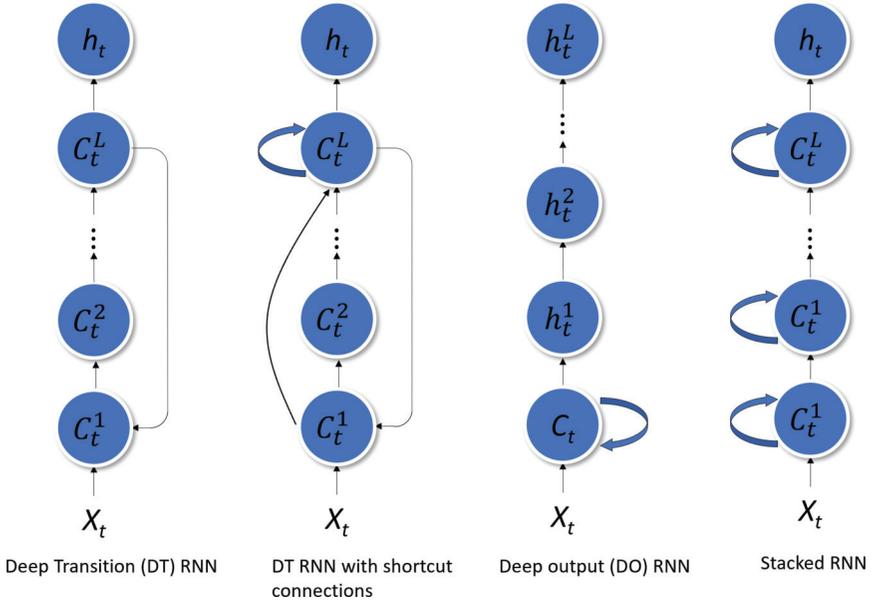


Figure 17: Illustration of four deep RNN proposals.

4.3 Recurrent Auto-encoder for Video Compression: A Case Study of Using RNNs for Neural-based Architectural Design

We have so far introduced the most popular RNN cells and macro models. Yet, the list is far from complete. As a matter of fact, there could be an infinite number of RNN models since one can come up with his/her own design. To pursue this goal, it is helpful to revisit the early SRN design in Section for inspiration – how the recurrence can help achieve the specific learning task (which is the sequence learning in the SRN case)? Then, as mentioned in Section 3, we can design a feed-forward cell model to meet the goal and use time unrolling and BPTT in the training to complete an RNN model. The pipelining of design and deployment of RNN models is shown in Figure 18.

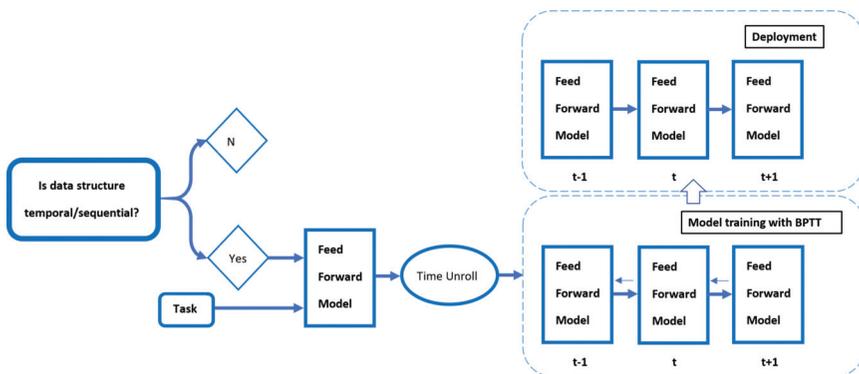


Figure 18: The process of designing and deploying an RNN model.

To further illustrate the abovementioned procedure, we use video compression as an example. Although video compression is still dominated by traditional hand-crafted codecs like H.264 or HEVC whose standards are described in [60, 70] respectively, the learning based approach has been an active research topic for work such as [9, 34–36, 39] in recent years. In this example, we would limit ourselves to the illustration of how an innovative way of using an RNN can contribute to this emerging application and how to come up with a new design if existing RNN models are not applicable without going into exhaustive details or debating the merits of the proposal.

For video data, the input to the system is a sequence of frames. The temporal nature makes RNN an ideal modeling candidate. A feed-forward RNN cell model should reflect the input structure and bear the task in mind. Video compression has two tasks: 1) the system should compress the voluminous raw video data (usually millions of consecutive high resolution frames) as much as possible so that it can be readily stored or streamed for distribution; 2)

it should be able to recover the frames from heavily compressed data with acceptable visual quality. The former task is concerned with information compression that controls the bit-rate while the latter task is concerned with information reconstruction that aims to minimize the distortion between the reconstructed frame for display and the original frame before compression. They are called the rate-distortion problem jointly. The reconstruction task is trivial if the compression is lossless. However, this is rarely the case since lossless compression such as the Huffman or the arithmetic coder alone cannot deliver the desired compression ratio. For lossy compression where a certain amount of information is discarded in the compression process, the distortion minimization task is very challenging.

Without going into further details, we provide a scope of the underlying problem. The spatial-temporal data structure makes the ConvLSTM a good candidate. Yet, the ConvLSTM is not well suited for the rate-distortion problem. A better candidate is called the auto-encoder that is popularized by [26]. The diagram of an auto-encoder is shown in Figure 19.

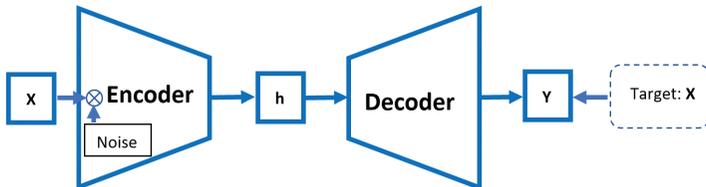


Figure 19: The diagram of an auto-encoder.

Although being seemingly similar to the Seq2Seq RNN, an auto-encoder requires the decoder to recover unperturbed encoder’s input that is error injected (e.g., adding random noise or lossy compressed input in video compression). Since the output of the encoder is a dimensionally reduced representation of the input, the encoder can serve as the compression module while the decoder can serve as the reconstruction module. They offer a candidate to the solution of the rate-distortion problem. Recently, the auto-encoder has been adopted for learning-based image compression in [3, 62, 63].

Since the auto-encoder is a high-level macro model, we have the freedom in choosing popular image processing architectures such as CNNs (e.g., VGG in [46], ResNet in [25], etc.) to deal with the spatial input and generate the reconstructed output. Then, we can feed the output recurrently back to the encoder or the decoder at the next time step by time unrolling. The unrolled graph of the resulting design is shown in Figure 20. This can serve as the guidance for BPTT. The detailed model architectures and hyperparameter settings can be worked out with further experimentation.

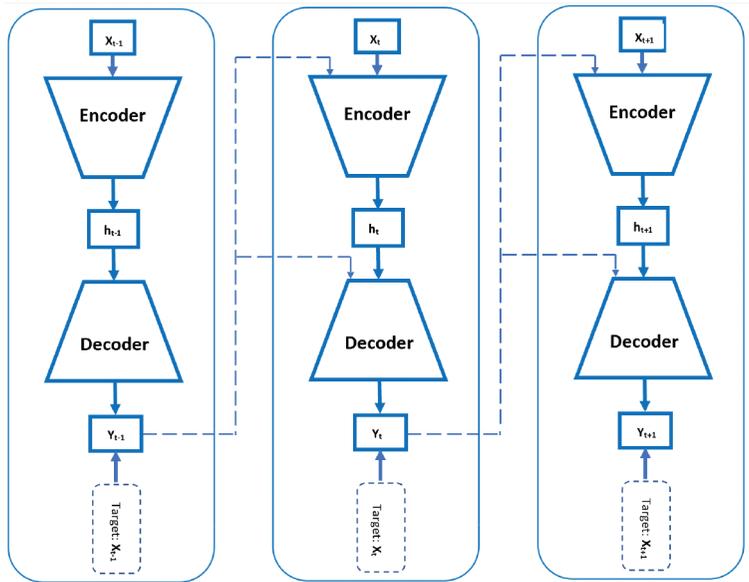


Figure 20: The auto-encoder-based RNN for video compression.

5 Theory of Memory Decay and Enhancement

LSTM and GRU cells were designed to enhance the memory length of RNNs and address the gradient vanishing/exploding issue. However, thorough analysis on their memory decay property has been lacking. In this section, we attempt to fill in the void. We will analyze the memory length of three RNN cells – SRN, LSTM and GRU. Our analysis is different from the investigation of the gradient vanishing/exploding problem in the following sense. The gradient vanishing/exploding problem occurs in the training process while our memory analysis is conducted on a trained RNN model. Based on the analysis, we will discuss how to extend the memory. Along this direction, we will focus on a new cell model called the extended LSTM (ELSTM) that does not suffer from memory decay and delivers better results than existing models. Hopefully, our discussion can serve as food for thought and provoke further research interests in this field.¹

¹The source code for ELSTM can be found at <https://github.com/yuanhangsu/ELSTM-DBRNN>.

5.1 Memory of SRN, LSTM and GRU

For a large number of NLP tasks, we are concerned with finding semantic patterns from input sequences. The memory of a cell characterizes its ability to map input sequences of a certain length into such a representation. Here, we define the memory as a function that maps elements of the input sequence to the current output. Thus, the memory of an RNN is not only about whether an element can be mapped into the current output but also how this mapping takes place. It was reported by [20] that an SRN only memorizes sequences of length between 3 and 5 units while an LSTM could memorize sequences of length longer than 1000 units.

5.1.1 Memory of SRN

For ease of analysis, we begin with Elman’s SRN model with a linear hidden-state activation function and a non-linear output activation function since such a cell model is mathematically tractable while its performance is equivalent to Jordan’s model. The SRN model can be described by the following two equations:

$$\underline{c}_t = \mathbf{W}_c \underline{c}_{t-1} + \mathbf{W}_{in} \underline{\mathbf{X}}_t, \quad (25)$$

$$\underline{h}_t = f(\underline{c}_t), \quad (26)$$

By induction, \underline{c}_t can be written as

$$\underline{c}_t = \mathbf{W}_c^t \underline{c}_0 + \sum_{k=1}^t \mathbf{W}_c^{t-k} \mathbf{W}_{in} \underline{\mathbf{X}}_k, \quad (27)$$

where \underline{c}_0 is the initial internal state of the SRN. Typically, we set $\underline{c}_0 = \underline{0}$. Then, Equation (27) becomes

$$\underline{c}_t = \sum_{k=1}^t \mathbf{W}_c^{t-k} \mathbf{W}_{in} \underline{\mathbf{X}}_k. \quad (28)$$

As shown in Equation (28), SRN’s output is a function of all proceeding elements in the input sequence. The dependency between the output and the input allows the SRN to retain the semantic sequential patterns from the input. For the rest of this survey, we call a system whose function introduces dependency between the output and its proceeding elements in the input as *a system with memory*.

Although the SRN is a system with memory, its memory length is limited. Let $\sigma_{\max}(\cdot)$ denotes the largest singular value (to be differed from the sigmoid

function denoted as σ without subscription). Then, we have

$$\begin{aligned} |\mathbf{W}_c^{t-k} \mathbf{W}_{in} \underline{\mathbf{X}}_k| &\leq \|\mathbf{W}_c\|^{t-k} |\mathbf{W}_{in} \underline{\mathbf{X}}_k| \\ &= \sigma_{\max}(\mathbf{W}_c)^{t-k} |\mathbf{W}_{in} \underline{\mathbf{X}}_k|, k \leq t, \end{aligned} \quad (29)$$

where $\|\cdot\|$ denotes the matrix norm and $|\cdot|$ denotes the vector norm. Both are of the l^2 norm. The inequality is derived by the definition of the matrix norm. The equality is derived by the fact that the spectral norm (i.e., the l^2 norm of a matrix) of a square matrix is equal to its largest singular value.

Here, we are only interested in the memory decay case when $\sigma_{\max}(\mathbf{W}_c) < 1$. Since the contribution of $\underline{\mathbf{X}}_k$, $k < t$, to output $\underline{\mathbf{h}}_t$ decays at least in form of $\sigma_{\max}(\mathbf{W}_c)^{t-k}$, we conclude that **SRN's memory decays at least exponentially with its memory length $t - k$** .

5.1.2 Memory of LSTM

Under the assumption $\underline{\mathbf{c}}_0 = \mathbf{0}$, the hidden-state vector of the LSTM can be derived by induction from Equations (9) to (10) as

$$\underline{\mathbf{c}}_t = \sum_{k=1}^t \underbrace{\left[\prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right]}_{\text{forget gate}} \sigma(\mathbf{W}_i \underline{\mathbf{I}}_k) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_k). \quad (30)$$

By setting $f(\cdot)$ in Equation (26) to the hyperbolic-tangent function, we can compare outputs of the SRN and the LSTM below:

$$\underline{\mathbf{h}}_t^{SRN} = \phi \left(\sum_{k=1}^t \mathbf{W}_c^{t-k} \mathbf{W}_{in} \underline{\mathbf{X}}_k \right), \quad (31)$$

$$\begin{aligned} \underline{\mathbf{h}}_t^{LSTM} &= \\ &\sigma(\mathbf{W}_o \underline{\mathbf{I}}_t) \phi \left(\sum_{k=1}^t \underbrace{\left[\prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right]}_{\text{forget gate}} \sigma(\mathbf{W}_i \underline{\mathbf{I}}_k) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_k) \right). \end{aligned} \quad (32)$$

We see from above that \mathbf{W}_c^{t-k} and $\prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j)$ play the same memory role for the SRN and the LSTM, respectively.

We can find many special cases where LSTM's memory length exceeds SRN's regardless of the choice of SRN's model parameters (\mathbf{W}_c , \mathbf{W}_{in}). For example,

$$\begin{aligned} \exists \mathbf{W}_f \quad s.t. \quad \min |\sigma(\mathbf{W}_f \underline{\mathbf{I}}_j)| &\geq \sigma_{\max}(\mathbf{W}_c), \\ \forall \sigma_{\max}(\mathbf{W}_c) &\in [0, 1). \end{aligned}$$

Then, we have

$$\left| \prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right| \geq \sigma_{\max}(\mathbf{W}_c)^{t-k}, t \geq k. \quad (33)$$

As given in Equations (29) and (33), the impact of input I_k on the output of the LSTM lasts longer than that of the SRN. This means that **there always exists an LSTM whose memory length is longer than SRN for all possible choices of SRN.**

Conversely, to find an SRN with an advantage similar to LSTM, we need to ensure that

$$\|\mathbf{W}_c^{t-k}\| \geq 1 \geq \left| \prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right|.$$

Although such \mathbf{W}_c exists, this condition would easily leads to memory explosion. For example, one close lower bound for $\|\mathbf{W}_c^{t-k}\|$ is $\sigma_{\min}(\mathbf{W}_c)^{t-k}$, where $\sigma_{\min}(\mathbf{W}_c)$ is the smallest singular value of \mathbf{W}_c . This comes from the fact of

$$\|AB\| \geq \sigma_{\min}(A)\|B\| \text{ and } \|B\| = \sigma_{\max}(B) \geq \sigma_{\min}(B),$$

plus analysis with induction. We need $\sigma_{\min}(\mathbf{W}_c) \geq 1$. Since

$$\|\mathbf{W}_c^{t-k}\| \geq \sigma_{\min}(\mathbf{W}_c)^{t-k},$$

SRN's memory will grow exponentially and end up with memory explosion. This memory explosion condition does not exist in the LSTM.

5.1.3 Memory of GRU

By setting \mathbf{U}_z , \mathbf{U}_r and \mathbf{U} to zero matrices, we can obtain the following simplified GRU system from Equations (15) to (18):

$$\underline{\mathbf{z}}_t = \sigma(\mathbf{W}_z \underline{\mathbf{X}}_t), \quad (34)$$

$$\tilde{\mathbf{h}}_t = \phi(\mathbf{W} \underline{\mathbf{X}}_t), \quad (35)$$

$$\mathbf{h}_t = \underline{\mathbf{z}}_t \mathbf{h}_{t-1} + (\underline{\mathbf{1}} - \underline{\mathbf{z}}_t) \tilde{\mathbf{h}}_t. \quad (36)$$

For the simplified GRU with the initial rest condition, we can derive the following by induction:

$$\mathbf{h}_t = \sum_{k=1}^t \left[\underbrace{\prod_{j=k+1}^t \sigma(\mathbf{W}_z \underline{\mathbf{X}}_j)}_{\text{update gate}} \right] (\underline{\mathbf{1}} - \sigma(\mathbf{W}_z \underline{\mathbf{X}}_k)) \phi(\mathbf{W} \underline{\mathbf{X}}_k). \quad (37)$$

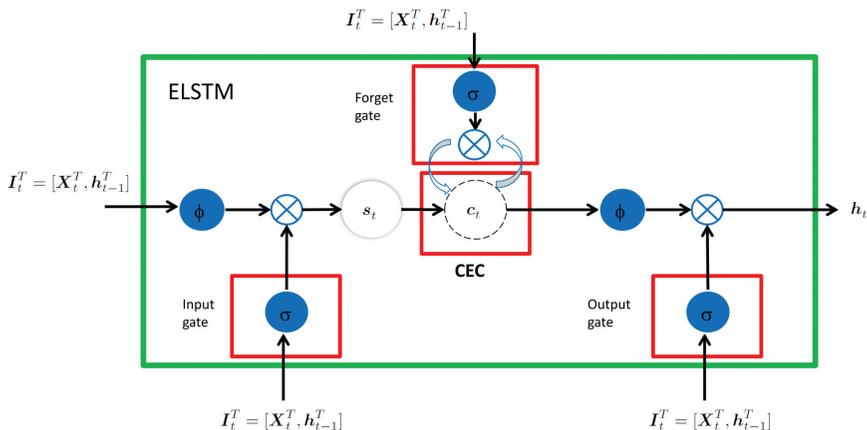


Figure 21: The diagrams of the ELSTM cell.

By comparing Equations (30) and (37), we see that the update gate of the simplified GRU and the forget gate of the LSTM play the same role. In other words, **there is no fundamental difference between GRU and LSTM**. Such finding is substantiated by the non-conclusive performance comparison between GRU and LSTM conducted in [13, 23, 30].

Because of the presence of the multiplication term introduced by the forget gate and the update gate in Equations (30) and (37), respectively, the longer the distance of $t - k$, the smaller these terms. Thus, the memory responses of LSTM and GRU to $\underline{\mathbf{I}}_k$ diminish inevitably as $t - k$ becomes larger. This phenomenon occurs regardless of the choice of model parameters. For complex language tasks that require long memory responses such as sentence parsing, LSTM's and GRU's memory decay may have a significant impact on their performance.

5.2 Extended LSTM

To address this design limitation, we introduce a scaling factor to compensate the fast decay of the input response. This leads to a new solution called the extended LSTM (ELSTM) which is introduced in [55, 56]. The ELSTM cell is depicted in Figure 21, where $\underline{\mathbf{s}}_t \in \mathbb{R}^N$, is the trainable input scaling vectors

The ELSTM cell can be described by

$$\underline{\mathbf{c}}_t = \sigma(\mathbf{W}_f \underline{\mathbf{I}}_t) \underline{\mathbf{c}}_{t-1} + \underline{\mathbf{s}}_t \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (38)$$

$$\underline{\mathbf{h}}_t = \sigma(\mathbf{W}_o \underline{\mathbf{I}}_t) \phi(\underline{\mathbf{c}}_t). \quad (39)$$

a bias term $\underline{\mathbf{b}} \in \mathbb{R}^N$ for $\underline{\mathbf{c}}_t$ is omitted in Equation (39). As shown above, we introduce scaling factor, $\underline{\mathbf{s}}_i$, $i = 1, \dots, t - 1$, to the ELSTM to increase or

decrease the impact of input $\underline{\mathbf{I}}_i$ in the sequence.

To show that the ELSTM has longer memory than the LSTM, we first derive a closed form expression of $\underline{\mathbf{h}}_t$ as

$$\underline{\mathbf{h}}_t = \sigma(\mathbf{W}_o \underline{\mathbf{I}}_t) \phi \left(\sum_{k=1}^t \underline{\mathbf{s}}_k \left[\prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right] \sigma(\mathbf{W}_i \underline{\mathbf{I}}_k) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_k) \right). \quad (40)$$

Then, we can find the following special case:

$$\begin{aligned} & \exists \underline{\mathbf{s}}_k \quad s.t. \\ & \left| \underline{\mathbf{s}}_k \prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right| \geq \left| \prod_{j=k+1}^t \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right| \quad \forall \mathbf{W}_f. \end{aligned} \quad (41)$$

By comparing Equation (41) with Equation (33), we conclude that **there always exists an ELSTM whose memory is longer than LSTM for all choices of LSTM**. Conversely, we cannot find such an LSTM with similar advantage to the ELSTM. This demonstrates ELSTM's system advantage over the LSTM by design.

The numbers of parameters used by various RNN cells are compared in Table 2, where $\underline{\mathbf{X}}_t \in \mathbb{R}^M$, $\underline{\mathbf{h}}_t \in \mathbb{R}^N$ and $t = 1, \dots, T$. As shown in Table 2, the number of parameters of the ELSTM cell depends on the maximum length, T , of the input sequences, which makes the model size uncontrollable. To address this problem, we choose a fixed T_s (with $T_s < T$) as the upper bound on the number of scaling factors, and set $\underline{\mathbf{s}}_t = \underline{\mathbf{s}}_{(t-1) \bmod T_s + 1}$ if $t > T_s$ and t starts from 1, where mod denotes the modulo operator. In other words, the sequence of scaling factors is a periodic one with period T_s , so the elements in a sequence that are distanced by the length of T_s will share the same scaling factor.

Table 2: Comparison of parameter numbers.

Cell	Number of parameters
LSTM	$4N(M + N + 1)$
GRU	$3N(M + N + 1)$
ELSTM	$4N(M + N + 1) + N(T + 1)$

The ELSTM cell with periodic scaling factors can be described by

$$\underline{\mathbf{c}}_t = \sigma(\mathbf{W}_f \underline{\mathbf{I}}_t) \underline{\mathbf{c}}_{t-1} + \underline{\mathbf{s}}_{t_s} \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (42)$$

$$\underline{\mathbf{h}}_t = \sigma(\mathbf{W}_o \underline{\mathbf{I}}_t) \phi(\underline{\mathbf{c}}_t), \quad (43)$$

where $t_s = (t - 1) \bmod T_s + 1$. We observe that the choice of T_s affects the network performance. Generally speaking, a small T_s value is suitable for

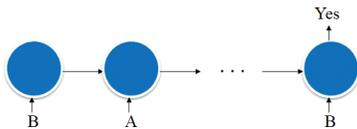


Figure 22: A toy experiment of estimating the presence of “A.”

simple language tasks that demand shorter memory while a larger T_s value is desired for complex ones that demand longer memory. For Seq2Seq RNN model, a larger T_s value is always preferred.

5.2.1 Study of Scaling Factor

To examine the memory capability of the scaling factor, we carry out the following experiment. The RNN cell is asked to tell whether a special element “A” exists in the sequence of a single “A” and multiple “B”s of length T . The training data contain T positive samples where “A” locates from position 1 to T , and one negative sample where there is no “A” exists. The cell takes in the whole sequence and generates the output at time step T as shown in Figure 22.

We would like to see the memory response of the LSTM and the ELSTM to “A.” If “A” lies at the beginning of the sequence, the LSTM’s memory decay may lead to the loss of the information of “A”’s presence. The memory responses of the LSTM and the ELSTM to input $\underline{\mathbf{I}}_t$ are calculated as

$$\underline{\mathbf{mr}}_t^{LSTM} = \left[\prod_{j=t+1}^T \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right] \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (44)$$

$$\underline{\mathbf{mr}}_t^{ELSTM} = \underline{\mathbf{s}}_t \left[\prod_{j=t+1}^T \sigma(\mathbf{W}_f \underline{\mathbf{I}}_j) \right] \sigma(\mathbf{W}_i \underline{\mathbf{I}}_t) \phi(\mathbf{W}_{in} \underline{\mathbf{I}}_t), \quad (45)$$

The detailed model settings can be found in Table 3.

Table 3: Network parameters for a toy experiment.

Number of RNN layers	1
Embedding layer vector size	2
Number of RNN cells	1
Batch size	5

We conduct multiple experiments by increasing the sample length T one at a time and see when the LSTM cannot keep up with the ELSTM. We train

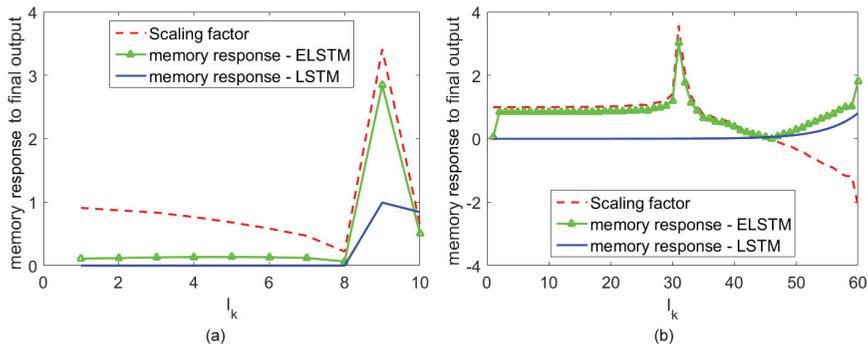


Figure 23: Comparison of LSTM’s and ELSTM’s memory responses.

the LSTM and the ELSTM models with an equal number of epochs until both report no further change in training loss. It is observed that, when $T = 60$, LSTM’s training loss starts to reach a plateau while ELSTM’s loss can further decrease to zero. As a result, the LSTM starts to “forget” when $T \geq 60$. The plot of memory responses for two cases are shown in Figure 23.

Figure 23(a) shows the memory response of the trained LSTM and the trained ELSTM on a sample with $T = 10$ with “A” at position 9. It can be seen that although both the LSTM and the ELSTM have stronger memory responses at “A,” the ELSTM attends better than the LSTM since its response at position 10 is smaller than LSTM’s. We also observe that the scaling factor has a larger value in the beginning and then slowly decreases as the location comes closer to the end. Then, it spikes at position 9. It appears that the scaling factor is doing its compensating job at both ends of the sequence.

Figure 23(b) shows the memory response of the trained LSTM and the trained ELSTM on a sample with $T = 60$ with “A” at position 30. In this case, the LSTM is not able to “remember” the presence of “A” and it does not have a strong response with respect to it. The scaling factor is doing its compensating job at the first half of the sequence and especially in the middle. This causes strong ELSTM’s response to “A.” Although the scaling factor cannot change its value adaptively once its training is completed, it can still learn the model’s memory decay rate and the averaged importance of that position in the training set. It is important to point out that the scaling factor needs to be initialized at 1 for each cell.

5.2.2 Comparison with Existing Models

We compare the performance of four RNN macro-models:

1. cell RNN models;

2. BRNN;
3. Seq2Seq;
4. Seq2Seq with attention by [66].

For each RNN model, we compare three cell designs: LSTM, GRU, and ELSTM. We conduct experiments on language modeling and report the testing perplexity (i.e., the natural exponential of the model’s cross-entropy loss). For a more thorough experiment on sentence parsing and part of speech tagging (POS), please refer to [56] for more details. We set $T_s = 3$ in all models. The training, the validation and the testing datasets used are from the Penn Treebank (PTB) by [38]. The PTB has 42,068, 3370 and 3761 training, validation and testing sentences, respectively. It has 10,000 tokens in total.

The input is first fed into a trainable embedding layer as described in [8, 69] before it is sent to the actual network. Table 4 shows the detailed network and training specifications. We do not finetune network hyper-parameters or apply any engineering trick (e.g., feeding additional inputs other than the raw embedded input sequences) for the best possible performance since our main goal is to compare the performance of LSTM, GRU, ELSTM cells under various macro-models. As shown in Table 5 and Figure 24, the ELSTM cell gives the best performance for all four RNN macro-models.

Table 4: Network parameters and training details.

Embedding layer vector size	5
Number of RNN cells	5
Batch size	50
Number of RNN layers	1
Training steps	11 epochs
Learning rate	0.5
Optimizer	AdaGrad by [17]

Table 5: LM test perplexity.

	LSTM	GRU	ELSTM
CELL RNN	267.47	262.39	248.60
BRNN	78.56	82.83	71.65
Seq2seq	296.92	293.99	266.98
Seq2seq with Att	17.86	232.20	11.43

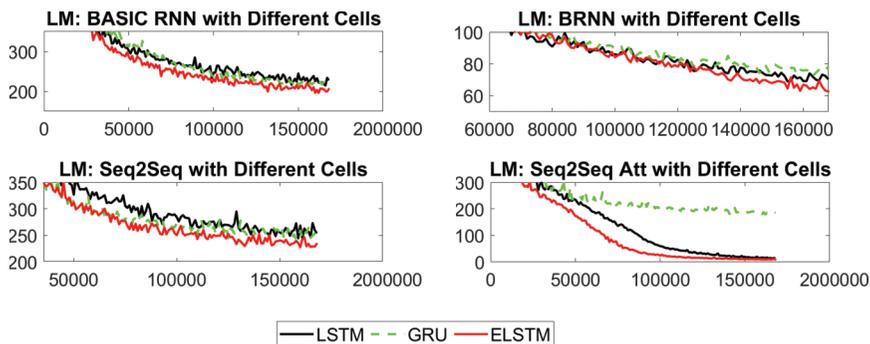


Figure 24: The training perplexity vs. the number of training steps with different cells.

6 Conclusion and Future Work

“A search space Odyssey,” Jürgen Schmidhuber once used it in the title of his paper about LSTM’s learning behavior. The same can be said about the journey people had gone through and probably will continue to take on for the development of RNNs. The unconquered land of understanding these sequence learning systems is full of surprises and serendipities. There are many directionals while each direction is awaiting exploration; thus, each direction is an Odyssey itself.

In this Odyssey, we try to offer perspectives and lay the foundation in addressing the following questions: what are RNNs and where do they come from? How are they trained and what are their usages? Are there any variants of RNNs and how to use them for real world problems? Finally, what is memory and how to build an RNN that can learn efficiently by remembering? We spent great length in answering each of those questions: RNNs are a particular type of neural network that has at least one cyclic connection that builds a path from a network node back to itself. They emerge from SRN and were further refined all the way to LSTM and GRU, etc. They are trained using BPTT by time unrolling and their usages are omnipresent in our daily lives. Cell models like SRN, LSTM and GRU or macro models including BRNN, Seq2Seq, and Deep RNNs are just a few variants of RNNs. One can create more by designing a feedforward model and, then, unrolling it. To answer the last question, we define memory as a system function. From this perspective, we conducted detailed analysis on RNN cell models, demystified their memory properties, and found the downside of their memory decay. Our new proposal of ELSTM can tackle the problem using a trainable scaling factor to extend the memory length. The ELSTM offers outstanding performance for complex language tasks.

Finally, there are many interesting issues to be explored further. Here, we would like to point out two of them as future research directions.

6.1 Acceleration of RNN's Computational Speed

One of the downsides that prevents the wider adoption of RNNs is their lack of capability for parallel computing since the elements in RNNs' output are inherently temporally dependent to each other, as we discussed in Sections 4.3 and 5. It is well known that the recent wave of deep learning-based techniques for artificial intelligence tasks is made possible by accelerating their computation on modern computer chips such as graphic processing unit (GPU) which is designed specifically for parallel computing. In [1], the computation of a deep neural network with 5 CNN layers and 3 fully connected layers are separated onto two GPUs. This configuration has achieved significant speed up of the system as compared with the conventional non-parallelized ones.

Parallelization of a deep learning-based system usually happens on either the input level or on the architectural level. For the former scenario, each independent input sample in a batch can be processed at the same time. For the latter, independent computations are highly optimized onto separate computing threads. The independent nature of the convolution operation that its output at different positions does not interfere with each other makes CNN parallelization friendly. For RNNs, however, architectural level parallelization is still very limited. This has given rise to another sequence processing neural architecture called the transformer which is described in [14, 15, 65], which only has fully connection layers and is thus highly parallelizable.

6.2 RNNs for Multi-modal Systems

One emerging topic in the field of deep learning is multi-modal systems for utilizing the data in diverse forms on the Internet. A multi-modal system is a system that handles data (input and/or output) of multiple modalities. An image caption generation system such as those described in [37, 51, 67, 72] is one such example, where it takes its input in the form of images and generates its output in form of sentences. Architectures for multi-modal systems usually have multiple submodules, where each is tasked to handle different input/output data. With the image caption generation problem as an example, a typical neural architecture is shown in Figure 25. In the system, a

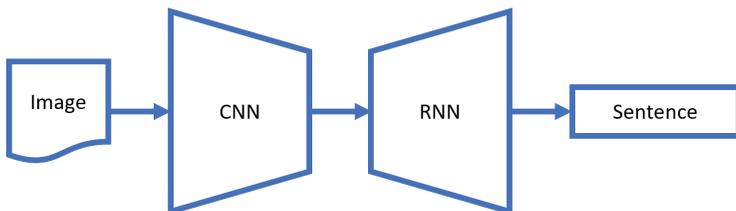


Figure 25: Image caption generation neural architecture.

CNN submodule is tasked to process the input image into a multi-dimensional (usually three-dimensional of height \times width \times channel) tensor. The tensor is then used as the representative feature of the input image and fed into an RNN to generate the descriptive sentence.

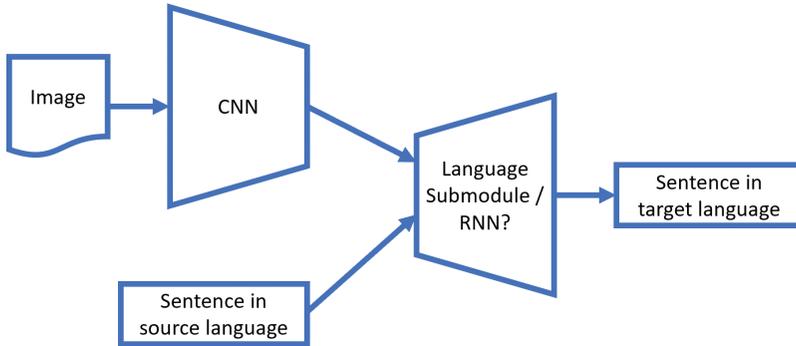


Figure 26: The multi-model machine translation neural architecture.

Another example is the multi-modal machine translation as shown in Figure 26, where the input to the system is already of multiple modalities in form of images and sentences. The proposers of multi-modal machine translation such as Haralampieva *et al.* [24], Nakayama and Nishida [40], Specia [49], Su *et al.* [52], and Toyama *et al.* [64] believe that, by feeding the system with the imagery information, it can produce better translation of words in different languages with similar visual appearances. Similar to image caption generation architectures, images are handled by CNNs. There are many candidate architectures for the sentence submodule, for which the RNN is a promising one to be further investigated.

References

- [1] K. Alex, I. Sutskever, and G. E. Hinton, “Imagenet Classification with Deep Convolutional Neural Networks”, *Advances in Neural Information Processing Systems*, 2012, 1097–105.
- [2] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align And Translate”, in *In Proceedings of The International Conference on Learning Representations*, 2015.
- [3] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational Image Compression with a Scale Hyperprior”, in *International Conference on Learning Representations (ICLR)*, 2018.

- [4] L. Baraldi, C. Grana, and Cucchiara, “Hierarchical Boundary-Aware Neural Encoder for Video Captioning”, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, 3185–94.
- [5] M. K. Baskar, M. Karafiát, L. Burget, K. Veselý, F. Grézl, and J. Černocký, “Residual Memory Networks: Feed-forward Approach to Learn Long-term Temporal Dependencies”, in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2017, 4810–4.
- [6] L. E. Baum and T. Petrie, “Statistical Inference for Probabilistic Functions of Finite State Markov Chains”, *The Annals of Mathematical Statistics*, 37(6), 1966, 1554–63.
- [7] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains”, *The Annals of Mathematical Statistics*, 41(1), 1970, 164–71.
- [8] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A Neural Probabilistic Language Model”, *Journal of Machine Learning Research*, 2003, 1137–55.
- [9] G. Bertasius, H. Wang, and L. Torresani, “Is Space-time Attention All You Need for Video Understanding”, *arXiv preprint arXiv:2102.05095*, 2(3), 2021, 4.
- [10] F. Chen, Y.-C. Wang, B. Wang, and C.-C. J. Kuo, “Graph Representation Learning: A Survey”, *APSIPA Transactions on Signal and Information Processing*, 9, 2020.
- [11] K. Cho, B. v. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation”, in *Proc. EMNLP’2014*, 2014.
- [12] K. Cho, B. v. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation”, in *Proc. EMNLP’2014*, 2014.
- [13] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”, *arXiv preprint*, (1412.3555), 2014.
- [14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *arXiv preprint arXiv:1810.04805*, 2018.
- [15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *arXiv preprint arXiv:2010.11929*, 2020.

- [16] J. Duan and C.-C. J. Kuo, “Bridging Gap between Image Pixels and Semantics via Supervision: A Survey”, *APSIPA Transactions on Signal and Information Processing*, 11, 2022.
- [17] Duchi, “Adaptive Subgradient Methods for Online Learning And Stochastic Optimization”, *The Journal of Machine Learning Research*, 2011, 2121–59.
- [18] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification and Scene Analysis*, Vol. 3, Wiley New York, 1973.
- [19] J. Elman, “Finding Structure in Time”, *Cognitive Science*, 14, 1990, 179–211.
- [20] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to Forget: Continual Prediction with LSTM”, *Neural Computation*, 2000, 2451–71.
- [21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, URL: <http://www.deeplearningbook.org>.
- [22] A. Graves, A.-r. Mohamed, and G. Hinton, “SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS”, in *Acoustics, Speech and Signal Processing (icassp), 2013 IEEE International Conference on. IEEE*, May 2013, 6645–9.
- [23] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey”, *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), 2017, 2222–32.
- [24] V. Haralampieva, O. Caglayan, and L. Specia, “Supervised Visual Attention for Simultaneous Multimodal Machine Translation”, *arXiv preprint arXiv:2201.09324*, 2022.
- [25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016, 770–8.
- [26] G. Hinton and R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, *Science*, 313, 2006, 504–7.
- [27] S. Hochreiter and J. Schmidhuber, “Long Short-term Memory”, *Neural Computation*, 9, 1997, 1735–80.
- [28] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks are Universal Approximators”, *Neural Networks*, 2, 1989, 359–66.
- [29] M. Jordan, “Serial Order: A Parallel Distributed Processing Approach”, *Advances in Psychology*, 121, 1997, 471–95.
- [30] A. Karpathy, J. Johnson, and F.-F. Li, “Visualizing and Understanding Recurrent Networks”, *arXiv preprint*, (1506.02078), 2015.
- [31] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen, “Interpretable Convolutional Neural Networks via Feedforward Design”, *Journal of Visual Communication and Image Representation*, 60, 2019, 346–59.
- [32] P. Langley, W. Iba, K. Thompson, et al., “An Analysis of Bayesian Classifiers”, in *Aaai*, Vol. 90, Citeseer, 1992, 223–8.

- [33] O. Levy, L. Kenton, N. FitzGerald, and L. Zettlemoyer, “Long Short-term Memory as A Dynamically Computed Element-wise Weighted Sum”, in *arXiv preprint*, No. 1805.03716, IEEE, 2018.
- [34] N. Ling, C.-C. J. Kuo, G. J. Sullivan, D. Xu, S. Liu, H.-M. Huang, W.-H. Peng, and J. Liu, “The Future of Video Coding”, *APSIPA Transactions on Signal and Information Processing*, 11, 2022.
- [35] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, “Deep Learning-Based Video Coding: A Review and A Case Study”, *arXiv preprint*, (1904.12462), 2019.
- [36] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, “Dvc: An End-to-end Deep Video Compression Framework”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, 11006–15.
- [37] E. Masmimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, “Generating Images from Captions with Attention”, in *International Conference on Learning Representations (ICLR)*, 2016.
- [38] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank”, *Computational Linguistics*, 19, 1993, 313–30.
- [39] F. Mentzer, G. D. Toderici, M. Tschannen, and E. Agustsson, “High-fidelity Generative Image Compression”, *Advances in Neural Information Processing Systems*, 33, 2020, 11913–24.
- [40] H. Nakayama and N. Nishida, “Zero-resource Machine Translation by Multimodal Encoder–decoder Network with Multimedia Pivot”, *Machine Translation*, 31(1-2), 2017, 49–64.
- [41] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to Construct Deep Recurrent Neural Networks”, *arXiv preprint*, (1312.6026), 2014.
- [42] V. Peddinti, D. Povey, and S. Khudanpur, “A Time Delay Neural Network Architecture for Efficient Modeling of Long Temporal Contexts”, in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
- [43] P. Razvan, T. Mikolov, and Y. Bengio, “On The Difficulty of Training Recurrent Neural Networks”, In *Proceedings of The International Conference on Machine Learning (ICML 2013)*, 2013, 1310–8.
- [44] M. Schuster and K. K. Paliwal, “Bidirectional Recurrent Neural Networks”, *Signal Processing*, 45, 1997, 2673–81.
- [45] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”, in *Advances in Neural Information Processing Systems (NeurIPs)*, 2015, 802–10.
- [46] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-scale Image Recognition”, in *International Conference on Learning Representations (ICLR)*, 2015.

- [47] D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, D. Povey, and S. Khudanpur, “Spoken Language Recognition Using x-vectors.”, in *Odyssey*, 2018, 105–11.
- [48] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust DNN Embeddings for Speaker Recognition”, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, 5329–33.
- [49] L. Specia, “Multimodal Simultaneous Machine Translation”, in *Proceedings of the First Workshop on Multimodal Machine Translation for Low Resource Languages (MMLRL 2021)*, 2021.
- [50] N. Statt, “Google’s AI Translation System is Approaching Human-level Accuracy”, 2016, URL: <https://www.theverge.com/2016/9/27/13078138/google-translate-ai-machine-learning-gnmt>.
- [51] M. Stefanini, M. Cornia, L. Baraldi, S. Cascianelli, G. Fiameni, and R. Cucchiara, “From Show to Tell: A Survey on Deep Learning-based Image Captioning”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [52] Y. Su, K. Fan, N. Bach, C.-C. J. Kuo, and F. Huang, “Unsupervised Multimodal Neural Machine Translation”, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, 10482–91.
- [53] Y. Su, Y. Huang, and C.-C. J. Kuo, “Efficient Text Classification Using Tree-structured Multi-linear Principal Component Analysis”, in *The 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, 585–90.
- [54] Y. Su and C.-C. J. Kuo, “Fast and Robust Camera’s Auto Exposure Control using Convex or Concave Model”, in *Proceedings of IEEE International Conference on Consumer Electronics (ICCE)*, 2015, 13–4.
- [55] Y. Su and C.-C. J. Kuo, “On Extended Long Short-term Memory and Dependent Bidirectional Recurrent Neural Network”, *OpenReview preprint*, 2018.
- [56] Y. Su and C.-C. J. Kuo, “On Extended Long Short-term Memory and Dependent Bidirectional Recurrent Neural Network”, *Neurocomputing*, (356), 2019, 151–61.
- [57] Y. Su, J. Y. Lin, and C.-C. J. Kuo, “A Model-based Approach to Camera’s Auto Exposure Control”, *Journal of Visual Communication and Image Representation*, (36), 2016, 122–9.
- [58] Y. Su, R. Lin, and C.-C. J. Kuo, “Tree-structured Multi-stage Principal Component Analysis (TMPCA): Theory and Applications”, *Expert Systems with Applications*, (118), 2019, 355–64.
- [59] Y. Sua, R. Lina, and C.-C. J. Kuo, “On Tree-structured Multi-stage Principal Component Analysis (TMPCA) for Text Classification”, *arXiv preprint 1807.08228*, 2018.

- [60] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard”, *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 2012, 560–76.
- [61] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks”, in *Advances in Neural Information Processing Systems*, 2014, 3104–12.
- [62] L. Theis, W. Shi, A. Cunningham, and F. Huszár, “Lossy Image Compression with Compressive Autoencoders”, in *International Conference on Learning Representations (ICLR)*, 2017.
- [63] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, “Full Resolution Image Compression with Recurrent Neural Networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, 5306–14.
- [64] J. Toyama, M. Misono, M. Suzuki, K. Nakayama, and Y. Matsuo, “Neural Machine Translation with Latent Semantic of Image and Text”, *arXiv preprint*, 2016, arXiv:1611.08459.
- [65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is All You Need”, in *In Advances in Neural Information Processing Systems (NIPS)*, 2017, 5998–6008.
- [66] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, “Grammar As A Foreign Language”, in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, 2773–81.
- [67] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and Tell: A Neural Image Caption Generator”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2015, 3156–64.
- [68] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme Recognition using Time-delay Neural Networks”, *IEEE transactions on acoustics, speech, and signal processing*, 37(3), 1989, 328–39.
- [69] B. Wang, A. Wang, F. Chen, Y. Wang, and C.-C. J. Kuo, “Evaluating Word Embedding Models: Methods and Experimental Results”, *APSIPA Transactions on Signal and Information Processing*, 8, 2019.
- [70] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264 / AVC Video Coding Standard”, *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 2003, 560–76.
- [71] Y. Wu and et. al., “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”, *1609.08144*, 2016.
- [72] G. Xu, S. Niu, M. Tan, Y. Luo, Q. Du, and Q. Wu, “Towards Accurate Text-based Image Captioning with Content Diversity Exploration”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, 12637–46.

- [73] H. Xu, Y. Chen, R. Lin, and C.-C. J. Kuo, “Understanding Convolutional Neural Networks via Discriminant Feature Analysis”, *APSIPA Transactions on Signal and Information Processing*, 7, 2018.
- [74] S. Zhang, H. Jiang, S. Wei, and L. Dai, “Feedforward Sequential Memory Neural Networks without Recurrent Feedback”, *arXiv preprint arXiv:1510.02693*, 2015.
- [75] S. Zhang, H. Jiang, S. Xiong, S. Wei, and L.-R. Dai, “Compact Feedforward Sequential Memory Networks for Large Vocabulary Continuous Speech Recognition.”, in *Interspeech*, 2016, 3389–93.
- [76] S. Zhang and M. Lei, “Acoustic Modeling with DFSMN-CTC and Joint CTC-CE Learning.”, in *INTERSPEECH*, 2018, 771–5.
- [77] S. Zhang, M. Lei, Z. Yan, and L. Dai, “Deep-FSMN for Large Vocabulary Continuous Speech Recognition”, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, 5869–73.
- [78] S. Zhang, C. Liu, H. Jiang, S. Wei, L. Dai, and Y. Hu, “Feedforward Sequential Memory Networks: A New Structure to Learn Long-term Dependency”, *arXiv preprint arXiv:1512.08301*, 2015.
- [79] S. Zhang, C. Liu, H. Jiang, S. Wei, L. Dai, and Y. Hu, “Nonrecurrent Neural Structure for Long-term Dependence”, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(4), 2017, 871–84.