Layer Flexible Adaptive Computational Time for Recurrent Neural Networks

Lida Zhang[1] and Diego Klabjan[1]

[1]Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL, 60208

January 22, 2019

Abstract

Deep recurrent neural networks perform well on sequence data and are the model of choice. It is a daunting task to decide the number of layers, especially considering different computational needs for tasks within a sequence of different difficulties. We propose a layer flexible recurrent neural network with adaptive computational time, and expand it to a sequence to sequence model. Contrary to the adaptive computational time model, our model has a dynamic number of transmission states which vary by step and sequence. We evaluate the model on a financial data set and Wikipedia language modeling. Experimental results show the performance improvement of 2% to 3% and indicate the model's ability to dynamically change the number of layers.

1 Introduction

Recurrent neural networks (RNN) are widely used in supervised machine learning tasks for its superior performance in sequence data, such as machine translation [1, 21], speech recognition [12, 13], image description generation [17, 23], and music generation [6]. The design of the underlying network is always a daunting task requiring substantial computational resources and experimentation. Many recent breakthroughs hinge on multilayer neural networks ability to increase model accuracy, [15, 25, 31], leading to the important decision in RNNs of the number of layers used. First, the right choice requires running several very expensive training processes to try many different numbers of layers. Even if a reinforcement learning algorithm is used to determine a good number of layers, [2, 36], it still requires a substantial training effort. The second issue with the number of layers in RNNs is the fact that the same number of layers is applied to each step in each sequence and the number is the same for each sample. It is conceivable that some samples are harder to classify than others and thus such harder samples should employ more layers. A similar argument holds for steps, e.g., certain steps in a sample can bear less predictive power and thus should use fewer layers in order to decrease the computational burden. The goal of our work is to introduce a network that automatically determines the number of layers - and together with this number of hidden vectors to use - in training and inference which is dynamic with respect to samples and step number.

To resolve the inherent problems of fixed structure neural networks, Graves [11] addresses this by providing an Adaptive Computational Time (ACT) model for RNN. In Graves' model, a sigmoidal halting unit is utilized to calculate a halting probability for each intermediate round within a step, and a computation stops when the accumulated halting probability reaches or exceeds a threshold. ACT can calculate the computational time in each RNN step and dynamically adapt to different samples and steps. The model is appealing due to its modeling flexibility and its advantages in increasing models accuracy, [9]. However, unlike multilayer networks, ACT

utilizes a single hidden vector and thus lacks the outstanding information transmission abilities of deep networks. It has multiple rounds (and thus cells) within each individual step, but only a single hidden vector. With the ACT mechanism, when a step of computation is halted, all intermediate states and outputs are used to calculate one mean-field state and output. As a result, ACT cannot efficiently represent functions of former hidden states and inputs as a multilayer network can due to its limited capacity. Surprisingly, our experimental results on a financial related data set revealed that a sequence to sequence model (seq2seq) applying ACT in each encoder and decoder step is outperformed by a standard seq2seq model. Therefore, in order to obtain the benefits of both ACT and a multilayer network, we develop a layer flexible RNN model with adaptive computational time. The model uses several rounds in each step similar to ACT but it is also using a flexible number of hidden states between two consecutive steps.

The novelty of our proposed model is its focus on learning the rules of transmitting states of different layers between two consecutive steps. Similar to Graves' work, we also utilize a unit to determine the action of each round within a step by calculating their halting probabilities. Instead of using a single hidden vector, a step in our model produces multiple hidden states (one state per round within the step). These multiple hidden states are then combined into a different number of hidden states for the next step (again, the number of new hidden states equals to the number of rounds in the next step). The combination strategy uses attention ideas, [22]. The network can thus have a flexible number of layers according to adaptive computational time in each step. We also develop several strategies to combine hidden states between two steps and along the way, utilize seq2seq and explore different teaching forcing strategies. Our model increases the accuracy of 2.9% on a financial data set and 1.9 % on Wikipedia language modeling, which attests to its robustness.

Our main contributions are as follows:

1. A layer flexible RNN model is proposed with adaptive computational time.
2. A seq2seq model applying our layer flexible RNN in each step. We note that ACT has been developed in the RNN setting and not seq2seq.
3. A new teacher forcing strategy in the decoder part of seq2seq.

The rest of the manuscript is structured as follows. In Section 2 we review the literature. In Section 3, the flexible layer adaptive computational time RNN model is presented, including all of the alternative options. In Section 4 we introduce the data sets and discuss all the experimental results.

2 Literature Review

A deep learning model and algorithm have many upper parameters. In a RNN, one of the problems is deciding the computation amount of a certain input sequence. A simple solution is comparing different depths of networks and manually selecting the best option, but a series of expensive training processes is required to make the right decision. Hyperparameter optimization [5, 4] and Bayesian optimization [30, 24, 28] have been proposed to select an efficient architecture of a network. Based on these concepts, Zoph [36] and Baker [2] propose mechanisms for network configuration using reinforcement learning. However, massive training efforts are still present. Another problem if such approaches is the assumption of a fixed structure of the network, irrespective of the underlying sample. The difficulty of classification varies in each data set and sample, and it is comprehensible that harder samples would require more computation. Therefore, applying networks with the same number of layers is inflexible and it cannot achieve the goal of flexible computing time among different samples. Conditional computation provides general ideas for alleviating the weaknesses of a fixed-structure deep network by a establishing learning policy [8, 3]. A halt neuron is designed and used as an activation threshold in self-delimiting neural networks [29, 32] to stop an ongoing computation whenever

it reaches or exceeds the halting threshold. [34] shows that conditional computation helps the networks obtain adaptive depth and thus acquire higher accuracy than fixed depth structures. Graves [11] introduces an Adaptive Computation Time (ACT) mechanism for RNN to dynamically calculate each input step's computing time and determine their halting condition. This series of work focuses on formulating the policies of halting condition and uses a single hidden vector in each cell, but none of them contributes to designing flexible multilayer networks or studies on learning the rules of states transmission.

The ACT mechanism [11] is proved to improve performances and is applied in a few different problems. Universal Transformers [9] apply ACT to improve a model's accuracy by dynamically adapting the times of revising representation in each position in a sequence. A dynamic time model for visual attention [20] is proposed to accelerate the processing time by adding a binary action at each step to determine whether to continue or stop. Figurnov et al. [10] prove that applying ACT on Residual Networks could dynamically choose the number of evaluated layers and propose a spatially adaptive computation time for Residual Networks for video processing to adapt the computation amount between spatial positions. Similarly, Neumann et al. [26] extend ACT to a recognizing textual entailment task. In addition, ACT is also applied to reduce computation cost and calculate computing time in speech recognition [19], image classification [18], natural language processing [35], and highway network [27]. These models simply apply ACT mechanism on other models to achieve the abilities of adaptively halting computations. They focus on solving their specific problems but do not make any change of the structure of ACT cells. However, our work concentrates in the inner design of layer flexible ACT cell for its ability of automatically and dynamically adapting the number of layers.

3 Model

We start with an explanation of RNN and ACT. A standard RNN contains three layers: the input layer, the hidden layer, and the output layer. The input layer receives input sequences x and transmits it to the hidden layer to compute the hidden states u. The output layer calculates the output y based on the updated state of each step. The equations are as follows:

$$u_t = f(x_t, u_{t-1})$$

$$y_t = \sigma(W_o u_t + b_o).$$

In step t, input x_t from the input sequence x is delivered to the network. A cell in the hidden layer uses the input x_t and the state u_{t-1} from the previous step to update the hidden state u_t in the current step. Long Short-Term Memory (LSTM) [16] and Gated Recurrent Unit (GRU) [7] are frequently applied in the hidden layer cell f, which contain the dynamic computing information and the activations of the hidden cells. The output y_t is computed utilizing an output weight W_o, an output bias b_o, and an activation function σ.

ACT extends the standard RNN. The hidden layer contains several rounds of computation and each round produces an intermediate state and output. The representation of intermediate states u_t^n and intermediate outputs o_t^n are as follows:

$$u_t^n = \begin{cases} f(x_t^0, u_{t-1}), & n = 0 \\ f(x_t^n, u_t^{n-1}) & n > 0 \end{cases}$$

$$x_t^n = (\delta_n, x_t)$$

$$o_t^n = \sigma(W_o u_t^n + b_o).$$

The first hidden cell, in step t, receives the state u_{t-1} from the previous step $t-1$ and computes the first intermediate state. All the following rounds of computation use the previous intermediate output u_t^{n-1} and produce an updated state u_t^n. To distinguish different rounds of computation, a flag δ_0 is augmented to the input x_t for the first round and another flag δ_n is

added for all the later ones. Each intermediate output o_t^n is computed based on the intermediate state u_t^n in the same round.

To determine the halting condition of a series rounds of computation, units h_t^n are introduced in each computing round n:

$$h_t^n = \sigma(W_h s_t^n + b_h). \tag{1}$$

The total computational time N_t in a step is decided by the halting units and the maximum threshold L. Whenever the accumulated halting units' value in a step t is over 1 or the computational time reaches L, the computation halts. The definition of total computational time N_t is as follows:

$$N_t = min\{min\{n|\sum_{i=1}^{n} h_t^i \geq 1 - \epsilon\}, L\}, \tag{2}$$

where ϵ is a hyper-parameter.

ACT uses all the intermediate states and outputs to calculate one mean-field state u_t and output y_t (as represented in (4) and (5) below) for each step. A probability p_t^n produced by halting unit h_t^n is introduced into ACT for calculating a mean-field state and output according to the contribution of each intermediate computational round in a step. The updated mean-field state u_t is transmitted to next input step and the output o_t is delivered to the output layer as the current step's output.

$$p_t^n = \begin{cases} h_t^n, & n < N_t \\ 1 - \sum_{i=1}^{N_t - 1} h_t^i & n = N_t \end{cases} \tag{3}$$

$$u_t = \sum_{i=1}^{N_t} p_t^i u_t^i \tag{4}$$

$$y_t = \sum_{i=1}^{N_t} p_t^i o_t^i \tag{5}$$

Given an input sequence x, the ACT model tends to compute as much as possible in each step to avoid making predictions and incurring errors. This can cause an extra computational expense and impede achieving the goal to adapt the computational time. Therefore, training the model to decrease the amount of computation becomes necessary. ACT introduces ponder cost $\mathcal{P}(x)$ as

$$\mathcal{P}(x) = N_t + p_t^{N_t}$$

to represent the total computational time during the input sequence. The loss function $\mathcal{L}(x, gt)$ with gt being the ground truth is modified to encourage the network to also minimize $\mathcal{P}(x)$:

$$\hat{\mathcal{L}}(x, gt) = \mathcal{L}(x, gt) + \tau \mathcal{P}(x)$$

where τ is a hyper-parameter time penalty that balances the ponder cost and prediction errors.

3.1 Layer Flexible Adaptive Computation Time Recurrent Neural Network

In this section, our Layer Flexible Adaptive Computational Time (LFACT) model is introduced. The main idea of LFACT is dynamically adjusting the number of layers according to the imminent characteristic of different inputs and efficiently transmitting each layer's information to the same layer in the next step. Differing from ACT where only the mean-field states u_t in (4) are transmitted to the next step, which can be viewed as a single layer network, LFACT is designed for transmitting each layer's state individually between every consecutive step. In LFACT we compute N_t and N_{t+1} as in ACT. Each cell i at step t takes o_t^{i-1} from the previous cell at t and $\bar{u}_{t-1}^i$ from step $t-1$ as input and creates u_t^i and o_t^i for $i = 1, ..., N_t$. The problem is that at step

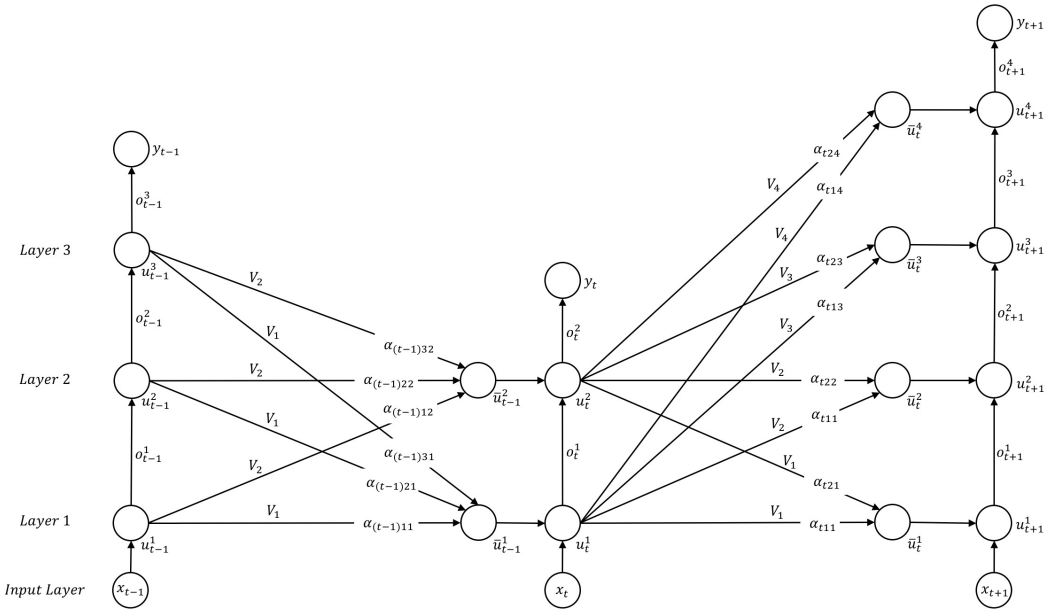


Figure 1: LFACT model - an example of three consecutive steps. Step $t-1$ has three layers, step t has two layers, and step $t+1$ includes four layers.

t we produce u_t^i for $i = 1, ..., N_t$ but for step $t+1$ we need $\bar{u}_t^i$ for $i = 1, ..., N_{t+1}$. The key of our model is to use the attention principle to create $\bar{u}_t^1, \bar{u}_t^2, ..., \bar{u}_t^{N_t+1}$ from $u_t^1, u_t^2, ..., u_t^{N_t}$. *Figure* 1 depicts the model.

The representation of the LFACT model is as follows:

$$u_t^n = \begin{cases} f(x_t, \bar{u}_{t-1}^n), & n = 0 \\ f(o_t^{n-1}, \bar{u}_{t-1}^n) & n > 0 \end{cases}$$

$$o_t^n = \sigma(W_o u_t^n + b_o).$$

The LFACT model contains two types of states. One state is the primary production of each hidden cell u_t^n, which is the same as the states in standard RNN. The other state is the transmission state $\bar{u}_t^n$ that is used for transmitting layer information to the next step. In step t, the hidden layer cell f uses the input and the transmission state $\bar{u}_{t-1}^n$ received from the previous step to compute and update the primary state u_t^n. The primary states are used to compute the transmission state $\bar{u}_t^n$ for the next step. The first layer's input x_t is obtained from the input layer, and the later layers' inputs are the output of previous layer o_t^{n-1} in the current step. The equations governing the relationship between the two transmission states read

$$\bar{u}_t^n = \sum_{i=1}^{g_t^n} \alpha_{tin} u_t^n$$

$$\alpha_{tin} = \frac{e^{\beta_{tin}}}{\sum_{j=1}^{g} e^{\beta_{tjn}}} \quad i \leq g_t^n$$

$$\beta_{tin} = V_n^T \cdot \sigma(W_Q o_t^i + V_Q u_t^i + b_Q) \quad i \leq g_t^n. \tag{6}$$

To compute the transmission states $\bar{u}_t^n$, an attention unit α is introduced to represent the relationship between the primary states u_t^n in a certain layer n and the primary states in other

layers. We propose two choices to select g_t^n:

$$g_t^n = \begin{cases} min(N_t, n), & (a) \\ N_t. & (b) \end{cases}$$

Option (a) only considers the relationship between the state u_t^n of the current layer and the states u_t^i from the lower layers (i.e. $i \leq n$), called limited (LTD). Alternative (b) utilizes all computed transmission states (i.e. $i \leq N_t$), called ALL. When strategy LTD is applied, we have $N_{t+1} \leq N_t$; all primary states u_t^i in deeper layers (i.e. $i > N_t$) cannot be used. Strategy ALL aims to include the computed information of all the layers. To distinguish different layers, extra weights V_n are utilized to compute α. Weights V_n, W_Q and V_Q in (6) to compute α can either be matrices or vectors; we compare the different options in the experimental section.

We use the same method as ACT to compute N_t (as represented in (2)), the computing time of each step. But unlike ACT, the halting unit is computed based on the output and transmission state of each layer:

$$h_t^n = \sigma(W_h o_t^n + b_h + V_h \bar{u}_{t-1}^n).$$

In addition, instead of computing a mean-field output, we directly take the output of the deepest layer as one step output:

$$y_t = o_t^{N_t}.$$

3.2 Augmentation Model

To simplify our LFACT model, an alternative layer flexible RNN model is proposed, marked as Augmentation (AUG), as follows:

$$u_t^n = \begin{cases} f(x_t, \widetilde{u}_t^n), & n = 0 \\ f(o_t^{n-1}, \widetilde{u}_t^n) & n > 0 \end{cases}$$

$$\widetilde{u}_t^n = W_u(n, u_{t-1}^{min(n, N_{t-1})}) + b_u$$

$$y_t = \sigma(W_o u_t^{N_t} + b_o).$$

Each layer receives the state from the same layer from the previous step or the deepest one if $N_{t-1} < n$, with augmenting the depth index of layer n as a flag to distinguish different layers. Weights W_u and bias b_u are applied on each augmented state to maintain all of the states in the network to be of the same size. The mechanism of halting and computational time N_t are the same as in ACT expressions (1) and (2).

3.3 Sequence to Sequence Model with LFACT

In order to deal with sequence tasks, we propose a combination model using a seq2seq (encoder-decoder) model and our LFACT model, as *Figure* 2 shows. In the seq2seq model, a cell in each step is replaced with our LFACT model to form a deep and flexible network. The seq2seq encoder part accepts a sequence input, and in the decoder part, we propose 4 strategies for its input based on paradigms of teacher-forcing [33]: (1) No feeding data in the decoder part (None, $z_t = \emptyset$); (2) Feed in the prediction of the previous step (Pred, $z_t = softmax(y_{t-1})$), which is the same as the standard teacher-forcing strategy; (3) Feed the label of the last encoder step into every step of the decoder part (Same, $z_t = gt_{T'}$); (4) Take the average of the previous step prediction and the label of the last encoder step (Ave, $z_t = (gt_{T'} + softmax(y_{t-1}))/2$); (5) Concatenate the output of the previous step and the label of the last encoder step (Concat, $z_t = (gt_{T'}, softmax(y_{t-1}))$). Here $gt_{T'}$ is the ground truth of predicting for $x_{T'}$.

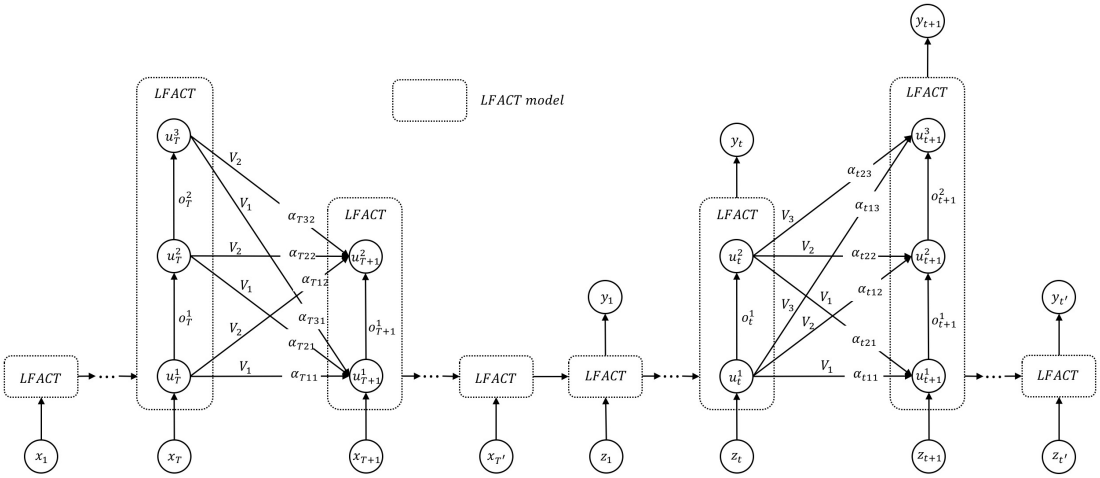


Figure 2: Seq2seq with LFACT model

4 Computational Experiments

4.1 Training

We train LFACT in several phases. We first train the seq2seq model. The weights in our model consist of those pertaining to the seq2seq model, and the remaining weights unique to our model. We use the pretrained seq2seq weights to initialize the former. In the first 25 training epochs, we freeze the weights from seq2seq, and train the remaining variables, and in the following 25 epochs, the seq2seq variables are released to be trained and the rest of the variables are frozen. The process is repeated four times, resulting in a total of 200 training epochs.

4.2 Financial Data Set

We test our LFACT models on a financial data set from [14]. The data set consists of tick prices of twenty-two ETFs at five minute intervals. The data is labeled into five classes to represent the significance of the price changes, e.g., one class corresponds to the price being within one standard deviation. We have a size 22 softmax classification layer in each step. We have three test instances, and in each one we train our model on 50 weeks of returns (45,950 samples), use the next week (905 samples) as validation data to save the best performance weights, and test the model based on the saved weights using the following week (905 samples).

We use the seq2seq version of LFACT to predict the following five steps. The raw sequence data with input length of 20 is compressed into ten through a convolutional neural network (CNN) layer. We train our models for 200 epochs with 64 batch size in a straightforward way without using the strategy from *Section* 4.1. All results are from the test set. We choose 0.001 as our ponder time penalty ($\tau = 0.001$) and utilize the Adam optimizer with 0.00005 learning rate to train our model. The maximum number of layers L is 5. We use GRU in each layer of LFACT.

In *Figure* 3(a) we present the average F1 scores over the three test instances for different models. LTD and ALL are the two strategies for computing transmission states $\bar{u}$, matrix and vector are the two options for computing attention unit α, and s2s indicates the seq2seq model, which works better than RNN. All the models are trained with the teacher-forcing Same strategy ($z_T = gt_{t'}$) which outperforms all strategies (*Figure* 4). We test plain ACT and seq2seq alone which have been tuned with respect to all hyperparameters as our baseline models, and compare them with seq2seq models applying our proposed LFACT models. The red bars represent results of the two baseline models, and the yellow bars are from our proposed

(a) (b)

Figure 3: Performance of the models

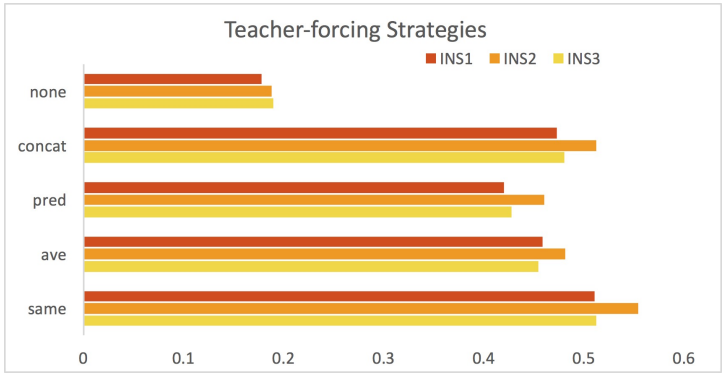


Figure 4: Effect of teacher-forcing

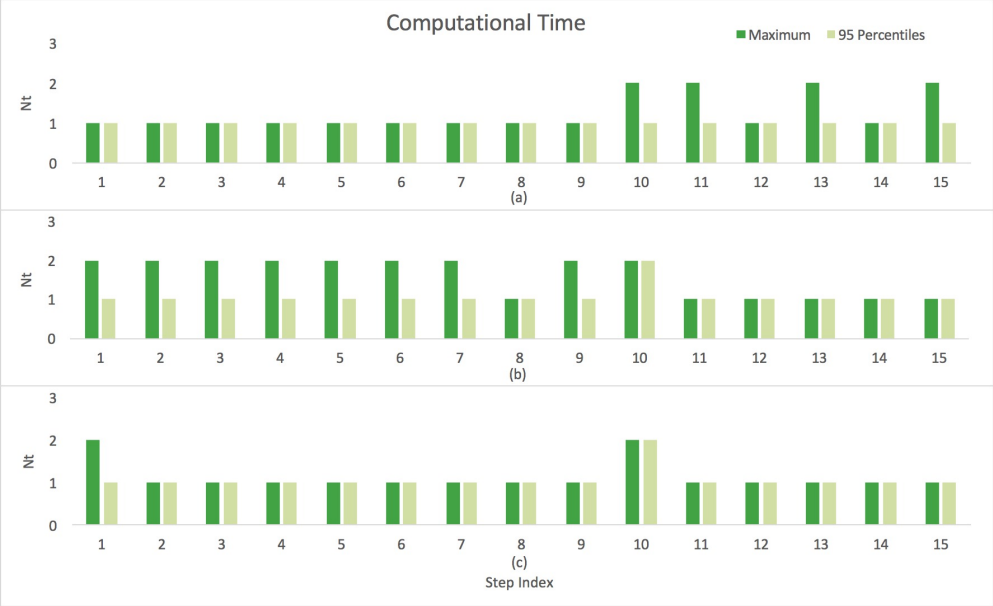


Figure 5: Computation time distribution

models. The best performance model is our proposed model LTD_matrix, which uses strategy LTD to compute transmission states $\bar{u}$ and matrices for attention unit α. It is surprising that seq2seq model applying ACT performs worse than seq2seq alone; it performs the worst over all of the models. We assume that it is due to the information discarding when computing mean-field states and outputs in ACT. In *Figure* 3(b), we compare the results of each test instance individually for our best proposed model LTD_matrix (yellow bars) and the best baseline model seq2seq (red bars). The results show that model LTD_matrix yields significant benefits over the best baseline model for each one of the 3 instances.

In *Figure* 4, we compare the performance of the five teacher-forcing strategies based on model LTD_matrix. The x-axis represents the F1 scores. The red, orange and yellow bars respectively indicate the first, second and third test instance. From the results, strategy Same shows the best performance on all of the three test instances, and the three strategies containing the ground truth of the last encoder step $gt_{T'}$ (Concat, Ave, and Same) all perform better on every test instance than the two that do not (None and Pred). A plausible explanation is that the prediction results (Pred) and the state information (None) in each step are not accurate enough, and feeding these errors into the next step as input can cause further errors in the following predictions. Thus, strategy Same that does not include any prediction results with potential errors works the best, and the two strategies None and Pred that only have prediction information are the worst.

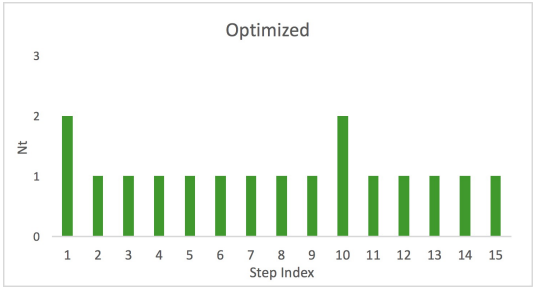
The LFACT model is expected to dynamically adapt the computation time according to the inputs. *Figure* 5 presents the computation time (N_t) result for the first test instance: (a) and (b) are the N_t results of the training and validation process based on the optimized weights, and (c) is for test. We present the maximum N_t of all the samples by dark green bars and the 95 percentile with light green bars to show how is the computational time distributed between different steps. Step indexes 1 to 10 indicate the ten steps in the encoder, and 11 to 15 are the five steps in the decoder. The result show the change of N_t among different steps, indicating that the LFACT model has the ability of adapting computational time dynamically according to its input. The N_t result for test is more similar to the validation than training set, because the test and validation sets are very close on the timescale (have similar distributions), therefore they have similar changing pattern of tick prices.

(a)

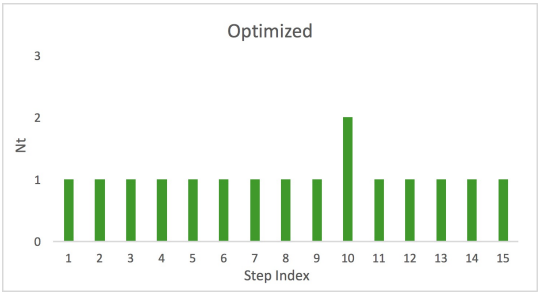
(b)

(c)

(d)



(e)



(f)

Figure 6: Computational time for early and optimized weights

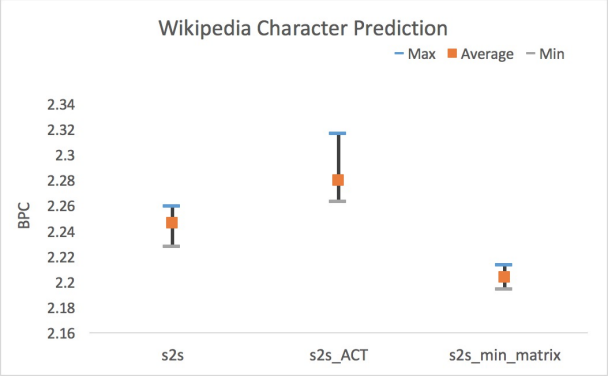


Figure 7: Results of Wikipedia language modeling

We observed in *Figure* 5 that most of the time a single layer is needed based on the optimized weights. We wonder if the model is trying to squeeze out the computational time as training progresses. To this end, *Figures* 6(a) and (b) show N_t of two samples based on early training weights (the 5th epoch), *Figures* 6(c) and (d) are the same samples based on intermediate training epoch weights (the 20th epoch), and *Figures* 6(e) and (f) depict the samples based on the optimized weights (the 57th training epoch). The significant change of N_t during the training process indicates that the loss function adequately captures both the loss and computational time by decreasing the computation time.

4.3 Wikipedia Language Modeling

This task is about predicting characters from the Hutter Prize Wikipedia data set, which is also used in Graves' ACT paper [11]. We clean the data by only keeping lowercase English letters and the common punctuation symbols in the main text. The data set is split into ten equal parts; the first eight are the training set, the ninth is the validation set, and the tenth is the test set. Sequences of 50 consecutive characters are chosen as input, and the following 5 characters are predicted. Each character is represented as one-hot, and presents one timestep.

Training is performed based on *Section* 4.1. The maximum number of layers L is set to 3, and a softmax layer with size 34 is added to each step in the decoder. To deal with overfitting, we apply a dropout layer after each cell with probability 0.8. The learning rate and ponder time penalty (τ) are both 0.0001, and the training epochs and batch size are the same as for the financial data set. We use the bit per character (BPC) to evaluate the performance of our model and the baseline models (the lower the better).

Figure 7 shows the comparison of the models. We applied five different random seeds for each model to initialize the weights. In *Figure* 7, we provide the maximum, minimum, and average BPC. From the figure, we observe that the average BPC of seq2seq LFACT is lower than both seq2seq alone and seq2seq with ACT, and that the worst case of seq2seq LFACT is still better than the best performance of the other two models. These results indicate that our model works best and improves the performance. The relative improvement of LFACT is 1.9% over seq2seq alone, and 3.3% over seq2seq ACT on average. In order to test the stability of the performance, we performed the t-test between LFACT and the other two models. The p-value between LFACT and seq2seq is 0.0006, and the p-value between LFACT and seq2seq ACT is 0.0002. From the t-test results, seq2seq with LFACT has a significant consistent benefit over the two baseline models. Moreover, the BPC range of LFACT is 0.019, which is much smaller than the value of 0.031 for seq2seq alone and 0.054 for seq2seq ACT. At the same time, the standard deviation for the LFACT BPC results is 0.0068, 0.0139 for seq2seq alone, and 0.0217 for seq2seq ACT. These results imply a higher stability for LFACT over seq2seq and seq2seq ACT.

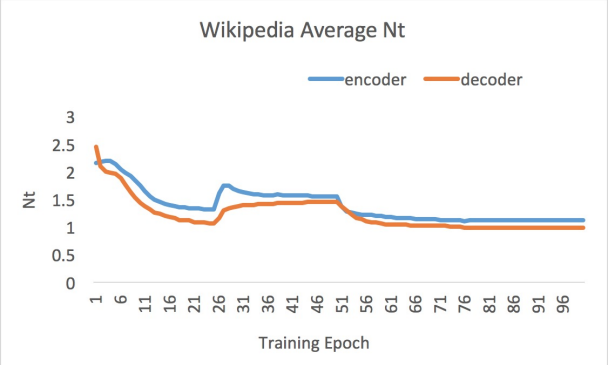


Figure 8: Computational time for Wikipedia language modeling

An interesting observation from both data sets is that ACT is outperformed by a pure seq2seq model. We point out that evaluations in the ACT work [11] have been conducted in the RNN setting where the conclusion is that ACT beats standard RNN.

In *Figure* 8, we provide the average encoder and decoder computation time during training of Wikipedia language modeling. We observe a clear decrease during the early training epochs, which eventually stabilizes. Note that during epochs 25 to 50, the computational time increases and stays at a relatively high level. This happens because we switch the trainable variables from LFACT-model specific to seq2seq specific at the 25th epoch. During this period, variables from the seq2seq model are trained for the first time, and the computational time related variables not belonging to the seq2seq portion cannot be trained. The barchart in Section A1 in the Appendix focuses on computational time based on the optimal weights. It shows the average across all samples in the underlying set. We have 50 time steps in encoder and 5 steps in decoder but to avoid clutter we report N_t every 3 time steps. The average computational time decreases as the sequence index increases, which indicates that the computational complexity is high at the beginning of a sequence and then decreases as more history is available to make predictions which is expected. In addition, we observe that the computational time for the test and validation set are very close but are slightly higher than the training set.

5 Conclusion

Deciding the structure of recurrent neural networks has been a problem in deep learning applications, in particular the number of layers. A halting unit is applied in a previous work to adapt the computational time to inputs, but a single hidden vector structure leads to the information transmission weaknesses. In this paper, we propose LFACT which utilizes an attention strategy in designing an information transmission policy which leads to a flexible multilayer recurrent neural network with adaptive computational time. LFACT can automatically adjust computational time according to the computing complexity of inputs and has outstanding dynamic information transmission abilities between consecutive time steps. We apply LFACT in a seq2seq setting and evaluate the model on a financial data set and Wikipedia language modeling. The experimental results show a significant improvement of LFACT over seq2seq and ACT on both data sets. The different number of layers in practice indicates LFACT's ability of adapting computational time and information transmission.

References

[1] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

[3] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *International Conference on Learning Representations*, 2015.

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, 2013.

[5] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2011.

[6] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[7] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

[8] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[9] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[10] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[11] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[13] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[14] Mark Harmon and Diego Klabjan. Dynamic prediction length for time series with sequence to sequence networks. *arXiv preprint arXiv:1807.00425*, 2018.

[15] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[18] Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. Iamnn: Iterative and adaptive mobile neural network for efficient image classification. *Workshop on International Conference on Learning Representations*, 2018.

[19] Mohan Li and Min Liu. End-to-end speech recognition with adaptive computation steps. *arXiv preprint arXiv:1808.10088*, 2018.

[20] Zhichao Li, Yi Yang, Xiao Liu, Feng Zhou, Shilei Wen, and Wei Xu. Dynamic computational time for visual attention. *IEEE International Conference on Computer Vision Workshop*, 2017.

[21] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[22] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[23] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-RNN). *International Conference on Learning Representations*, 2015.

[24] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, 2016.

[25] Abdel-rahman Mohamed, George E Dahl, Geoffrey Hinton, et al. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[26] Mark Neumann, Pontus Stenetorp, and Sebastian Riedel. Learning to reason with adaptive computation. *NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems*, 2016.

[27] Hyunsin Park and Chang D Yoo. Early improving recurrent elastic highway network. *arXiv preprint arXiv:1708.04116*, 2017.

[28] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, 2016.

[29] Jürgen Schmidhuber. Self-delimiting neural networks. *arXiv preprint arXiv:1210.0118*, 2012.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

[31] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, 2015.

[32] Rupesh Kumar Srivastava, Bas R Steunebrink, and Jürgen Schmidhuber. First experiments with powerplay. *The 2nd Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics*, 2013.

[33] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[34] Chris Ying and Katerina Fragkiadaki. Depth-adaptive computational policies for efficient visual tracking. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 109–122. Springer, 2017.

[35] Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. *The 55th Annual Meeting of the Association for Computational Linguistics*, 2018.

[36] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.

A Appendix

A.1 Computational Time for Wikipedia Language Modeling based on Optimal Weights