GAUSSIAN PROCESS LSTM RECURRENT NEURAL NETWORK LANGUAGE MODELS FOR SPEECH RECOGNITION

Max W. Y. Lam[⋆] *Xie Chen*[†] *Shoukang Hu*[⋆] *Jianwei Yu*[⋆] *Xunying Liu*[⋆] *Helen Meng*[⋆]

[⋆]The Chinese University of Hong Kong, Hong Kong SAR, China
[†]Microsoft AI and Research, One Microsoft Way, Redmond, WA, USA
{wylam, skhu, jwyu, xyliu, hmmeng}@se.cuhk.edu.hk, xieche@microsoft.com

ABSTRACT

Recurrent neural network language models (RNNLMs) have shown superior performance across a range of speech recognition tasks. At the heart of all RNNLMs, the activation functions play a vital role to control the information flows and tracking longer history contexts that are useful for predicting the following words. Long short-term memory (LSTM) units are well known for such ability and thus widely used in current RNNLMs. However, the deterministic parameter estimates in LSTM RNNLMs are prone to over-fitting and poor generalization when given limited training data. Furthermore, the precise forms of activations in LSTM have been largely empirically set for all cells at a global level. In order to address these issues, this paper introduces Gaussian process (GP) LSTM RNNLMs. In addition to modeling parameter uncertainty under a Bayesian framework, it also allows the optimal forms of gates being automatically learned for individual LSTM cells. Experiments were conducted on three tasks: the Penn Treebank (PTB) corpus, Switchboard conversational telephone speech (SWBD) and the AMI meeting room data. The proposed GP-LSTM RNNLMs consistently outperform the baseline LSTM RNNLMs in terms of both perplexity and word error rate.

Index Terms— Neural network language models, LSTM, Gaussian processes, variational inference, speech recognition

1. INTRODUCTION

Language models (LMs) play an important role in speech recognition to obtain the likelihood of a word sequence $\mathbf{w}_1^n = (\mathbf{w}_1, ..., \mathbf{w}_n)$:

$$P(\mathbf{w}_1^n) = P(\mathbf{w}_1, ..., \mathbf{w}_n) = \prod_{t=1}^{n} P(\mathbf{w}_t|\mathbf{w}_1^{t-1}) \tag{1}$$

In general, there are two types of LMs – discrete-space LMs, e.g., n-grams (NGs) [1, 2], and continuous-space LMs, e.g., neural network language models (NNLMs) [3, 4, 5].

In particular, this paper focuses on the recurrent neural network language models (RNNLMs) [5, 6, 7] and the long short-term memory (LSTM) [8, 9] units, which in recent years have been shown to define the state-of-the-art performance and give significant improvements over the conventional back-off n-grams [10, 11, 12]. LSTM RNNLMs recursively accumulates the sequential information in the complete word history and are well known for tracking longer history contexts and thus widely used in current RNNLMs. However, the fixed, deterministic parameter estimate in standard LSTM RNNLMs are prone to over-fitting and poor generalization when given limited and variable training data. Furthermore, the precise forms of activations in LSTM have been largely empirically based.

Gaussian process (GP) is a powerful Bayesian model that handles the uncertainty associated with the choice of functions. In our previous research [13] GPs were successfully applied to deep neural network based acoustic models. In this paper, they are used to construct the hidden nodes in LSTM RNNLMs, resulting in a novel architecture for RNNLMs – GP-LSTM. It allows the optimal forms of gates to be learned for individual LSTM cells. Experiments were conducted on three tasks: the Penn Treebank (PTB) corpus, Switchboard conversational telephone speech (SWBD) and the AMI meeting room data. The proposed GP-LSTM RNNLMs consistently outperform the baseline LSTM RNNLMs in terms of both perplexity and word error rate. To the best of our knowledge, this is the first work that applies GPs as gates in the LSTM RNNLMs.

The rest of the paper is organized as follows. Section 2 gives a brief review of RNNLMs. Section 3 describes our proposed GP-LSTM architecture for RNNLMs. Section 4 presetns the experimental results for the evaluation of GP-LSTM RNNLMs. Finally, we conclude our work in Section 5.

2. NEURAL NETWORK LANGUAGE MODELS

This paper focuses on the recurrent neural network language models (RNNLMs), which computes the word probability as

$$P(\mathbf{w}_t|\mathbf{w}_1^{t-1}) \approx P\left(\mathbf{w}_t|\mathbf{w}_{t-1}, \mathbf{h}_{t-1}\right), \tag{2}$$

where $\mathbf{h}_{t-1}$ is the *hidden state* that tries to encode $(\mathbf{w}_{t-2}, ..., \mathbf{w}_1)$ into a D-dimensional vector, and D is the number of hidden nodes that we refer to throughout this paper. In NNLMs, a word $\mathbf{w}_t$ is often represented by a N-dimensional one-hot row vector $\tilde{\mathbf{w}}_t$, where N is the number of words in the vocabulary. To deal with sparse text data, the one-hot vector is first projected into a M-dimensional continuous space [3] where M is the embedding size (usually $M \ll N$):

$$\mathbf{x}_t = \mathbf{U}\tilde{\mathbf{w}}_t^{\top}, \tag{3}$$

where $\mathbf{U}$ is a projection matrix that can be learned during training. Followed by the word embedding, we obtain the hidden state $\mathbf{h}_t$ by recursively applying a gating function: $\mathbf{h}_t = \mathbf{g}(\mathbf{h}_{t-1}, \mathbf{x}_{t-1})$, which is a vector function that controls the amount of information carried by $\mathbf{w}_{t-1}$ preserved in the "memory" $\mathbf{h}_t$. The commonly used architecture of RNNLMs uses a single sigmoid activation based gating function parameterized by a weight matrix $\boldsymbol{\Theta} \in \mathbb{R}^{D\times(M+D+1)}$:

$$\mathbf{g}_{\text{sig}}(\mathbf{x}_{t-1}, \mathbf{h}_{t-1}) = \boldsymbol{\sigma}\left(\boldsymbol{\Theta}\left[\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1\right]^{\top}\right), \tag{4}$$

This research was partially funded by Research Grants Council of Hong Kong General Research grant Fund No.14200218, and the Chinese University of Hong Kong (CUHK) grant No. 4055065.

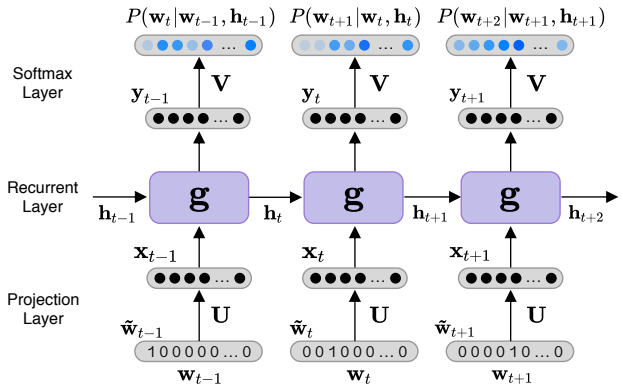

Fig. 1. The logic flow of RNNLMs

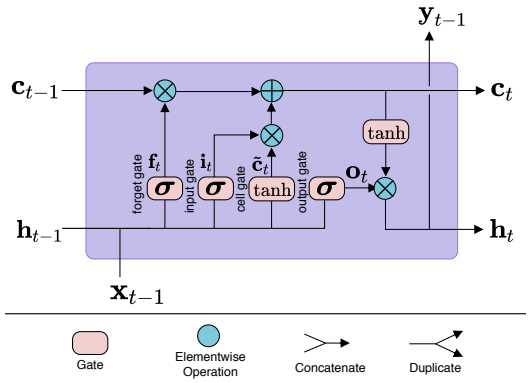

Fig. 2. An example architecture of a LSTM cell

where $\boldsymbol{\sigma}(\mathbf{u}) = [\sigma(u_1), ..., \sigma(u_D)]$ for any $\mathbf{u} \in \mathbb{R}^D$ while each $\sigma(u_i) = (1 + e^{u_i})^{-1}$. To align with the common input-output definition in neural networks, we let $\mathbf{y}_{t-1} \triangleq \mathbf{h}_t$ to be the output corresponding to input $\mathbf{x}_{t-1}$. We can then compute the approximated probability taking a *Softmax approximation*:

$$P(\mathbf{w}_t|\mathbf{w}_{t-1}, \mathbf{h}_{t-1}) = \frac{\exp(\mathbf{V}\mathbf{y}_{t-1}^\top)}{\sum \exp(\mathbf{V}\mathbf{y}_{t-1}^\top)}, \tag{5}$$

where $\mathbf{V}$ is another learnable projection matrix that projects the D-dimensional output back to the N-dimensional vocabulary space, and $\exp(\mathbf{v}) = [e^{v_1}, ..., e^{v_N}]$ for any $\mathbf{v} \in \mathbb{R}^N$. All in all, the logic flow of RNNLMs is illustrated on Fig. 1.

2.1. Long Short-Term Memory RNNLMs

In practice, it turns out that a simple architecture for RNNLMs (e.g. using a single gate) does not perform well in learning long-term dependencies due to the issue of vanishing gradients. Recently, significant improvements over simple gated RNNLMs have been achieved by a more complex architecture called long short-term memory (LSTM) [8] for RNNLMs [9].

In the LSTM architecture, the problem of vanishing gradients is tackled by introducing another recursively computed variable $\mathbf{c}_t$, namely *memory cell*, which aims to preserve the information over longer periods of time. Analogous to the logic gates in an electronic circuit, at time t four gates are computed – the forget gate $\mathbf{f}_t$, the input gate $\mathbf{i}_t$, the cell gate $\tilde{\mathbf{c}}_t$ and the output gate $\mathbf{o}_t$:

$$\mathbf{f}_t = \boldsymbol{\sigma}\left(\boldsymbol{\Theta}_\mathrm{f}\left[\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1\right]^\top\right) \tag{6}$$

$$\mathbf{i}_t = \boldsymbol{\sigma}\left(\boldsymbol{\Theta}_\mathrm{i}\left[\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1\right]^\top\right) \tag{7}$$

$$\tilde{\mathbf{c}}_t = \tanh\left(\boldsymbol{\Theta}_\mathrm{c}\left[\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1\right]^\top\right) \tag{8}$$

$$\mathbf{o}_t = \boldsymbol{\sigma}\left(\boldsymbol{\Theta}_\mathrm{o}\left[\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1\right]^\top\right) \tag{9}$$

where $\tanh(\mathbf{u}) = [\tanh(u_1), ..., \tanh(u_D)]$ for any $\mathbf{u} \in \mathbb{R}^D$. It is also possible to put $\mathbf{c}_t$ inside the multiplication with $\boldsymbol{\Theta}_\mathrm{f}$, $\boldsymbol{\Theta}_\mathrm{i}$, $\boldsymbol{\Theta}_\mathrm{c}$ and $\boldsymbol{\Theta}_\mathrm{o}$ (a.k.a. the "peephole" connections), however in practice the improvements are not significant compared to the substantial increase of memory requirement. Given the four gating outputs, we update

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tilde{\mathbf{c}}_t \tag{10}$$

$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t), \tag{11}$$

where $\otimes$ is the Hadamard product. An example LSTM architecture is shown on Fig. 2.

3. GAUSSIAN PROCESS LSTM RNNLMS

We first start with a formulation of gates $\mathbf{g}(\cdot)$ in a more generalized fashion such that the activation function $\mathbf{a}(\cdot)$ is not restricted to have the same form at each node, leading to

$$\mathbf{g}\left(\mathbf{z}; \boldsymbol{\Theta}\right) = \mathbf{a}(\boldsymbol{\Theta}\mathbf{z}^\top) = \left[a_1\left(\boldsymbol{\theta}_1 \bullet \mathbf{z}\right), ..., a_D\left(\boldsymbol{\theta}_D \bullet \mathbf{z}\right)\right], \tag{12}$$

where $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_D]^\top \in \mathbb{R}^{D\times(M+D+1)}$ and $\mathbf{z} = [\mathbf{x}_{t-1}, \mathbf{h}_{t-1}, 1]$ so that the gates in Eqn. (4, 6-9), are just the special cases of $\mathbf{g}(\cdot)$.

3.1. Activation Function Combination

To offer flexibility in determining the form of functions, we consider a finite set of K basis functions $\mathcal{B} = \{\phi_1, ..., \phi_K\}$ where each $\phi_k : \mathbb{R} \mapsto \mathbb{R}$ can be any activation function such as *Sigmoid* or *Tanh*. However, the degree of freedom K is usually small given the limited varieties of activations that can be used in combination. An effective way to expand $\mathcal{B}$ is to consider the activation functions parameterized by different weights $\boldsymbol{\theta}_d$ as different basis functions. In which case the d^{th} node of a gate can be represented by

$$g_d\left(\mathbf{z}; \boldsymbol{\lambda}_d, \boldsymbol{\theta}_d\right) = a_d\left(\boldsymbol{\theta}_d \bullet \mathbf{z}; \boldsymbol{\lambda}_d\right) = \sum_{k=1}^{K} \lambda_{k,d}\phi_k\left(\boldsymbol{\theta}_d \bullet \mathbf{z}\right), \tag{13}$$

where $\boldsymbol{\lambda}_d = [\lambda_{d,1}, ..., \lambda_{d,K}]$ are K basis activation coefficients. We can view the gates in LSTM as special cases where $\boldsymbol{\lambda}_d$ is an one-hot vector. For the consideration of comparable number of parameters, we use the same weight matrix for all basis functions in this paper.

3.2. Gaussian Process Activation Functions

To handle the activation weight parameter uncertainty for better generalization ability, the deterministic form of activation combination is modified into the following marginalization:

$$g_d\left(\mathbf{z}\right) = \int \sum_{k=1}^{K} \lambda_{kd}\phi_k\left(\boldsymbol{\theta}_d \bullet \mathbf{z}\right) p(\boldsymbol{\theta}_d|\mathcal{D})\, d\boldsymbol{\theta}_d \tag{14}$$

$$\approx \int \sum_{k=1}^{K} \phi_k\left(\boldsymbol{\theta}_d \bullet \mathbf{z}\right) p(\phi_k|\boldsymbol{\theta}_d) p(\boldsymbol{\theta}_d|\mathcal{D})\, d\boldsymbol{\theta}_d, \tag{15}$$

where under suitable normalization, each parameter λ_{kd} can be seen as an approximation of $p(\phi_k|\boldsymbol{\theta}_d)$, and can be estimated from the standard cross-entropy training with error back-propagation.

The remaining problem is to find the posterior of activation weight parameters $p(\boldsymbol{\theta}_d|\mathcal{D})$ given the training dataset $\mathcal{D} = \{\mathbf{w}_{t-1}, \mathbf{w}_t\}_{t=2}^n$ containing pairs of word contexts and target word. In our previous work [13], the basis combination of activations was applied to deep neural network based acoustic models, where a sinusoidal basis was used, and the posterior of $\boldsymbol{\Theta}$ was approximated by elementwise i.i.d. Gaussians. We showed that replacing activation functions by $g_d(\cdot)$ in deep neural networks results in a Gaussian process with generalized spectral kernels [14].

As a means of unifying the notations, we define the Gaussian process activation functions (GPacts) as

$$g_d(\mathbf{z}) = \int \sum_{k=1}^{K} \lambda_{kd}\phi_k(\boldsymbol{\theta}_d \bullet \mathbf{z}) \frac{P(\mathcal{D}|\boldsymbol{\theta}_d)\mathcal{N}(\boldsymbol{\theta}_d; \mathbf{0}, \mathbf{I})}{P(\mathcal{D})} d\boldsymbol{\theta}_d, \quad (16)$$

where $\mathcal{N}(\boldsymbol{\theta}_d; \mathbf{0}, \mathbf{I})$ is the non-informative standard Gaussian prior. Replacing the gates of fixed activation and deterministic weight parameters in the conventional LSTM cells with the above GPact nodes leads to the GP-LSTM RNNLM proposed in this paper. However, here $P(\mathcal{D})$ does not have a closed-form solution so it is necessary to employ advanced approximation techniques.

3.3. Variational Training for GP-LSTM RNNLMs

In order to handle this issue, variational approximation [15, 16] with sampling based efficient training approach is used to make the inference part efficient and compatible with the standard backpropagation training. The idea is to find a variational distribution $q(\boldsymbol{\Theta})$ to approximate $p(\boldsymbol{\Theta}|\mathcal{D})$. In this paper, we define $q(\boldsymbol{\Theta})$ as

$$q(\Theta_{d,r}) = \mathcal{N}(\mu_{d,r}, \exp(\gamma_{d,r})^2) \quad (17)$$

for $d \in \{1, ..., D\}$, $r \in \{1, ..., D+M+1\}$, so that we can update the distribution parameters $\mu_{d,r}, \gamma_{d,r}$ to minimize $\mathrm{KL}\{q(\boldsymbol{\Theta})||p(\boldsymbol{\Theta}|\mathcal{D})\}$ – the Kullback-Leibler (KL) divergence [17] that measures the closeness of distributions. Since KL divergence is always non-negative, we can effectively derive an upper bound of cross-entropy (CE) loss:

$$\begin{aligned}&\mathrm{CE} + \mathrm{KL}\{q(\boldsymbol{\Theta})||p(\boldsymbol{\Theta}|\mathcal{D})\} \\ &= \underbrace{-\mathbb{E}_{q(\boldsymbol{\Theta})}\left[\log P(\mathcal{D}|\boldsymbol{\Theta})\right]}_{\mathcal{L}_1} + \underbrace{\mathrm{KL}\{q(\boldsymbol{\Theta})||p(\boldsymbol{\Theta})\}}_{\mathcal{L}_2} = \mathcal{L}, \end{aligned} \quad (18)$$

where the first term can be approximated by Monte-Carlo sampling on the CE loss, i.e., to draw S samples from $q(\boldsymbol{\Theta})$ using a differentiable expression for the s^{th} sample as $\Theta_{d,r}^{(s)} = \mu_{d,r} + \exp(\gamma_{d,r})\epsilon_{d,r}^{(s)}$ with $\epsilon_{d,r}^{(s)} \sim \mathcal{N}(0, 1)$ and then we average over the CE losses produced by all samples for every training batch of n words:

$$\mathcal{L}_1 \approx -\frac{1}{S}\sum_{s=1}^{S}\sum_{t=2}^{n} \hat{P}(\mathbf{w}_t|\mathbf{w}_1^{t-1}) \bullet \log P\left(\mathbf{w}_t|\mathbf{w}_{t-1}, \mathbf{h}_{t-1}; \boldsymbol{\Theta}^{(s)}\right), \quad (19)$$

where $\hat{P}(\mathbf{w}_t|\mathbf{w}_1^{t-1}) \in \mathbb{R}^N$ is the target word probability vector. It is worth pointing out that when minibatch training is employed, it suffices in practice to take only one sample ($S = 1$) per batch so that no additional time cost is added to the standard CE training. The second term can also be expressed in closed-form:

$$\mathcal{L}_2 = \frac{1}{2}\sum_{d=1}^{D}\sum_{r=1}^{M+D+1} \mu_{d,r}^2 + \exp(2\gamma_{d,r}) - 2\log(\gamma_{d,r}) - 1 \quad (20)$$

since the prior is set to $p(\Theta_{d,r}) = \mathcal{N}(0, 1)$ elementwise. Note that the gradient computation using backpropagation is applicable to both $\mathcal{L}_1$ and $\mathcal{L}_2$. In particular, we have three types of parameters to be updated in each batch of optimization: (1) all parameters originally defined in the LSTM RNNLMs, (2) the mean parameter $\boldsymbol{\mu}$, and (3) the log-scale standard deviation parameter $\boldsymbol{\gamma}$. For (1) the calculation of gradients remain the same that only needs to concern about $\mathcal{L}_1$, whereas for (2) and (3), the gradients are written as

$$\frac{\partial \mathcal{L}}{\partial \mu_{d,r}} = \mu_{d,r} - \frac{\partial \mathcal{L}_1}{\partial \mathbf{y}_{t-1}}\frac{\partial \mathbf{y}_{t-1}}{\partial \Theta_{d,r}^{(s)}}, \quad (21)$$

$$\frac{\partial \mathcal{L}}{\partial \gamma_{d,r}} = \exp(2\gamma_{d,r}) - \frac{1}{\gamma_{d,r}} - \frac{\partial \mathcal{L}_1}{\partial \mathbf{y}_{t-1}}\frac{\partial \mathbf{y}_{t-1}}{\partial \Theta_{d,r}^{(s)}}\epsilon_{d,r}^{(s)}, \quad (22)$$

where the definition of $\mathbf{y}_{t-1}$ is referred back to Eqn. 5.

4. EXPERIMENTS

In this section, we evaluate the performance of GP-LSTM RNNLMs using the perplexity (PPL) measure and the word error rate (WER) obtained in automatic speech recognition (ASR) tasks. Both the LSTM and GP-LSTM language models were implemented using the Python GPU computing library PyTorch [18], while the code can be released upon request. For RNNLMs, we used 200 hidden nodes in the standard LSTM, and the basis activations $\mathcal{B} = \{\sin, \cos, \boldsymbol{\sigma}, \tanh, \mathrm{ReLU}\}$ and 150 hidden nodes in GPact to retain equal number of parameters for a fair comparison. Regarding the setup of training method for RNNLMs, parameters were updated in mini-batch optimization (10 sentences per batch) using the typical stochastic gradient descent (SGD) algorithm with an initial learning rate of 4. Under this setup, the average processed words per second in LSTM and GP-LSTM are 7438.23 and 4626.94, respectively, on the NVIDIA Tesla K80 GPU device. It is notable that our method was implemented to conduct experiments, and can be further optimized for practical use. In our experiments, all RNNLMs were interpolated with n-gram LMs to complement with each other as in many state-of-the-art systems [4, 5, 9, 19, 20, 21]:

$$P(\mathbf{w}_t|\mathbf{w}_1^{t-1}) = \lambda P_{\mathrm{NG}}(\mathbf{w}_t|\mathbf{w}_1^{t-1}) + (1-\lambda)P_{\mathrm{RNN}}(\mathbf{w}_t|\mathbf{w}_1^{t-1}), \quad (23)$$

where λ is the global weight of the n-gram LM $P_{\mathrm{NG}}(\cdot)$ that can be optimized using the EM algorithm on a validation set.

4.1. Experiments on Penn Treebank Corpus

Language Model	PPL	PPL(+4G)
4-gram	141.7	-
Standard LSTM	114.4	99.7
(P1) GPact as the forget gate	115.2	92.4
(P2) GPact as the input gate	115.1	91.7
(P3) GPact as the cell gate	111.9	88.3
(P4) GPact as the output gate	109.4	88.3
(P5) GPact as the $\mathbf{c}_t$ gate	111.2	88.2
(P6) GPact as a new gate for $\mathbf{h}_{t-1}$	**108.2**	**88.1**
(P7) GPact as a new gate for $\mathbf{x}_t$	112.0	90.0

Table 1. Perplexities (PPL) attained on PTB test set by applying GPact to different positions inside a LSTM cell

Because of practical limitation, in this paper we only consider replacing or adding a gate using GPact. To start with, we examine

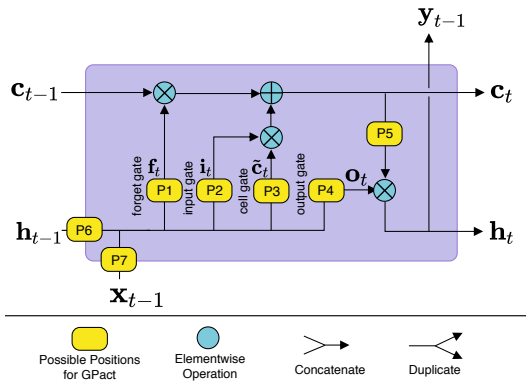

Fig. 3. Seven possible positions to use a Gaussian process activation (GPact) (highlighted as yellow) inside a LSTM cell

the efficacy of GPact at different positions in LSTM cell as shown in Fig. 3 using the Penn TreeBank (PTB) corpus [22], which consists of 10K vocabulary, 930K words for training, 74K words for development, and 82K words for testing. The resulting PPLs achieved by the 4-gram LM, baseline LSTM RNNLM and GP-LSTMs of placing GPact at positions (P1-7) are shown on Table 1. It is interesting to see that, no matter where we place GPact at, the resulting GP-LSTM consistently improves over the standard LSTM after interpolating with 4-gram LM (the 3rd column in Table 1). Meanwhile, placing GPact at P6 gave the best performance out of 7 possible positions. Hence, in all GP-LSTM RNNLMs of the following experiments GPact was placed at P6.

4.2. Experiments on Conversational Telephone Speech

Language Model	PPL	WER (%)	
		swbd	callhm
4-gram	80.6	12.1	23.9
LSTM	89.3	11.4	23.9
GP-LSTM	**87.2**	**11.3**	23.9
4-gram + LSTM	71.7	11.3	23.2
4-gram + GP-LSTM	**70.1**	**11.0**	**23.1**
4-gram + LSTM + GP-LSTM	**67.2**	**10.8**	**23.0**

Table 2. Perplexities (PPL) and word error rates (WER) on SWBD test speech attained by 4-gram, LSTM and GP-LSTM

To evaluate the performance of GP-LSTM RNNLMs in speech recognition, we used a 300-hour conversational telephone speech dataset on the Switchboard (SWBD) English system consisting of 3.6M words for training and 50K words for development. The perplexities performed by 4-gram, standatd LSTM and GP-LSTM on the development set are shown on Table 2. MPE-trained stacked hybrid DNN-HMM acoustic models constructed using the HTK toolkit v3.5 [23] were used. Both the top and bottom level DNNs contain 6 hidden layers (base on sigmoid activations) of 2000 nodes each, except the 2nd last bottleneck (BN) layer in the bottom level DNN contains 39 nodes used to produce BN features. The Bottleneck DNN was CE trained using 40-dimensional log Mel-filter bank (FBK) features spliced over a 9-frame window covering 4 frames to left and right as the input, and decision tree clustered 12k triphone state labels as the output layer targets. Finally, the WERs on recognizing Switchboard **swbd** and CallHome **callhm** test speeches are shown on Table 2, where GP-LSTM outperforms the standard LSTM with WER reductions of 0.5% and 0.2% (bottom line in Table 2) in **swbd** and **callhm**, respectively, when linearly combined with both the 4-gram LM and the LSTM RNNLM.

4.3. Experiments on Meeting Transcription Task

Language Model	PPL	WER (%)	
		dev	eval
4-gram	111.1	30.4	31.0
LSTM	83.4	29.4	30.0
GP-LSTM	**81.2**	**29.3**	**29.8**
4-gram + LSTM	76.8	29.3	29.8
4-gram + GP-LSTM	**74.2**	**29.0**	**29.4**
4-gram + LSTM + GP-LSTM	**71.2**	**28.7**	**29.3**

Table 3. Perplexities (PPL) and word error rates (WER) on AMI test speech attained by 4-gram, LSTM and GP-LSTM

The efficacy of GP-LSTM was also investigated in the highly challenging meeting transcription task [24, 25, 26] on the 59-hour Augmented Multi-party Interaction (AMI) corpus [27]. We used a mixture of text corpora with 26M words (AMI, Fisher 1/2 and SWBD) and 41K words in vocabulary to train the language model. For acoustic modeling, we trained a Tandem system [28] and a Hybrid system [29] separately and then combined them for better performance using joint decoding [30]. All systems were based on state-clustered decision-tree triphone models with 6000 states. For the setup of Tandem system, an initial GMM system with PLP feature was built in a way similar to [25]. 13-dimensional PLP features were extracted with the first and second temporal differentials, yielding a 39-dimensional feature vector. The MPE criterion was used to train the initial system. This initial system, was then extended to a Tandem system by appending 26 BN features learned from a DNN with four hidden layers (2000 nodes per layer) to the PLP features after a semi-tied covariance (STC) transform, giving 65-dimensional Tandem feature. The setup of DNN in Hybrid DNN-HMM system is the same as the setup of DNN in the Tandem system, where the DNN was pretrained discriminatively layer by layer and fine-tuned with several epochs until the frame accuracy converges in cross validation set. 9 consecutive frames are concatenated together as input to the neural networks. The Tandem features were also adopted as input static feature of the DNN and for the alignment for the targets in the Hybrid system. Speaker adaptive training based on CMLLR [31] was also used to adapt to the target test speaker for all systems. We report our performances on the AMI **dev** and **eval** set in Table 3. The results again suggest that the proposed GP-LSTM archecture is superior over the standard LSTM for RNNLMs in terms of both PPL and WER. The best system (bottom line in Table 3) interpolating the baseline LSTM and GP-LSTM RNNLMs outperformed the baseline LSTM RNNLM by 0.5%-0.6% absolute.

5. CONCLUSION

To conclude, consistent improvements were obtained using our proposed GP-LSTM RNNLMs, especially when they are interpolated with the n-grams. The noticeably larger improvements achieved by n-grams interpolated GP-LSTM RNNLMs can be explained by the responsibility of n-grams in interpolation with LSTM RNNLMs, i.e., to complement the overly exploited long-term memory with recent word histories. While GP-LSTM RNNLMs are better at selecting long-term information from the complete word history than the LSTM, it is sensible that n-grams interpolated GP-LSTM RNNLMs leads to a more significant improvement.

6. REFERENCES

[1] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *icassp*, 1995, vol. 1, p. 181e4.

[2] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[4] H. Schwenk, "Continuous space language models," *Computer Speech & Language*, vol. 21, no. 3, pp. 492–518, 2007.

[5] T. Mikolov, M. Karafiát, L. Burget, F. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

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

[7] M. Sundermeyer, I. Oparin, J. L. Gauvain, B. Freiberg, R. Schlüter, and H. Ney, "Comparison of feedforward and recurrent neural network language models," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8430–8434.

[8] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.

[9] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

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

[11] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.

[12] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent lstm neural networks for language modeling," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 3, pp. 517–529, 2015.

[13] M. W. Y. Lam, S. Hu, X. Xie, S. Liu, J. Yu, R. Su, X. Liu, and H. Meng, "Gaussian process neural networks for speech recognition," *Proc. Interspeech 2018*, pp. 1778–1782, 2018.

[14] Y. Kom Samo and S. Roberts, "Generalized spectral kernels," *arXiv preprint arXiv:1506.02236*, 2015.

[15] M I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine learning*, vol. 37, no. 2, pp. 183–233, 1999.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[17] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[18] A. Paszke et al., "Automatic differentiation in pytorch," in *NIPS 2017 Workshop Autodiff*, 2017.

[19] J. Park, X. Liu, M. J. F. Gales, and P. C. Woodland, "Improved neural network based language modelling and adaptation," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[20] A. Emami and L. Mangu, "Empirical study of neural network language models for arabic speech recognition," in *Automatic Speech Recognition & Understanding, 2007. ASRU. IEEE Workshop on*. IEEE, 2007, pp. 147–152.

[21] H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language models for speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 1, pp. 197–206, 2013.

[22] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[23] S. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, A. Ragni, V. Valtchev, P. Woodland, and C. Zhang, *The HTK Book (version 3.5a)*, 2015.

[24] T. Hain, L. Burget, J. Dines, G. Garau, V. Wan, M. Karafi, J. Vepa, and M. Lincoln, "The ami system for the transcription of speech in meetings," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*. IEEE, 2007, vol. 4, pp. IV–357.

[25] C. Breslin, K. K. Chin, M. J. F. Gales, and K. Knill, "Integrated online speaker clustering and adaptation," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[26] T. Hain, L. Burget, J. Dines, P. N. Garner, F. Grézl, A. El Hannani, M. Huijbregts, M. Karafiat, M. Lincoln, and V. Wan, "Transcribing meetings with the amida systems," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 486–498, 2012.

[27] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, et al., "The ami meeting corpus: A pre-announcement," in *International workshop on machine learning for multimodal interaction*. Springer, 2005, pp. 28–39.

[28] F. Grezl and P. Fousek, "Optimizing bottle-neck features for lvcsr.," in *ICASSP*, 2008, vol. 8, pp. 4729–4732.

[29] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 30–42, 2012.

[30] H. Wang, A. Ragni, M. J. F. Gales, K. M. Knill, P. C. Woodland, and C. Zhang, "Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages," 2015.

[31] M. J. F. Gales et al., "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998.