A New Concept of Convex based Multiple Neural Networks Structure

Yu Wang
Samsung Research America
Mountain View, CA
yu.wang1@samsung.com

Yue Deng
Samsung Research America
Mountain View, CA
y1.deng@samsung.com

Yilin Shen
Samsung Research America
Mountain View, CA
yilin.shen@samsung.com

Hongxia Jin
Samsung Research America
Mountain View, CA
hongxia.jin@samsung.com

ABSTRACT

In this paper, a new concept of convex based multiple neural networks structure is proposed. This new approach uses the collective information from multiple neural networks to train the model. From both theoretical and experimental analysis, it is going to demonstrate that the new approach gives a faster training speed of convergence with a similar or even better test accuracy, compared to a conventional neural network structure. Two experiments are conducted to demonstrate the performance of our new structure: the first one is a semantic frame parsing task for spoken language understanding (SLU) on ATIS dataset, and the other is a hand written digits recognition task on MNIST dataset. We test this new structure using both recurrent neural network and convolutional neural networks through these two tasks. The results of both experiments demonstrate a 4x-8x faster training speed with better or similar performance by using this new concept.

ACM Reference Format:
Yu Wang, Yue Deng, Yilin Shen, and Hongxia Jin. 2019. A New Concept of Convex based Multiple Neural Networks Structure. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

The concept of neural network has a long history in scientific research, which can be traced back to early 1940s, when neural plasticity mechanism was discovered [10]. It is well known that the current neural network structure's prototype was from the concept of perceptron, which is known as the principal component of neural network for a long time. However, during that period, the training mechanism has not been involved in the network structure. In 1970s, Paul Werbos firstly proposed the back-propagation training in his thesis [39], which has been a well known and standard algorithm to adjust the neuron weight in a network since then. Since 1980s, neural network structures with back-propagation training has been widely applied in the field of control, which are mainly functioned for system identification purpose [24]. Over the last decade, deep learning algorithm based on multiple layers neural networks has become extremely popular in the machine learning field. It has been widely applied in many application domains, including computer vision, speech recognition, natural language understanding and etc [6, 15].

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

During the last decade, there are lots of progresses on building a variety of different neural network structures, like recurrent neural networ (RNN), long short-term memory (LSTM) network, convolutional neural network (CNN), time-delayed neural network and etc. [12, 13, 31]. Training these neural networks, however, are still heavily to be relied upon back-propagation [40] by computing the loss functions' gradients, then further updating the weights of networks. Despite the effectiveness of using back-propagation, it still has several well known draw-backs in many real applications. For example, one of the main issues is the well known gradient vanishing problem [3], which limits the number of layers in a deep neural network and also affect the training speed of the neural networks adversely. The other issue is the non-convexity in deep neural networks, which may give sub-optimal result hence may degrade the model's performance.

In order to overcome the gradient vanishing problem and speedup the training performance, a lot of efforts have been made on proposing different neural network structures during the past decades. One of the most well known approach, long short-term memory (LSTM) network, reduces the gradient vanishing effect by controlling the flow of states' memory using multiple gates structure [11, 12]. It also improved the training speed as larger gradient values can be passed from back layers to front layers. Changing the activation function to a rectified linear unit (ReLU) function will also be helpful to improve the vanishing gradient issue [22].

The other common difficulty in training neural networks is the non-convex optimization issue, which is mainly caused by the nonlinear activation function at each layer of neural network. The main consequence of this issue is that the model may converge to a local optimal solution instead of the global one, hence gives an inferior model performance. One solution for this problem is by changing the action function to a linear one, like Rectified Linear Unit(ReLU) , and initializing network's weights using Xavier initialization [5] by keeping the variance to be close for input and output. However, this solution still highly depends on the data distribution in a dataset.

In this paper, we propose a new structure of convex based multiple neural networks to improve the model performance in terms of training speed and accuracy, by bypassing the vanishing gradient issue through a global loss function, and reducing the negative effects due to non-convex optimization. Experiments using convex multiple neural networks on two tasks are discussed: one is to build a semantic frame parser to jointly detect intent and fill slots tags for an SLU system, the other is a MNIST hand written digits recognition task. The results are compared with other single neural

network baseline models to demonstrate the improvement in terms of training convergence speed and accuracy.

The paper is organized as following: Section 2 gives a background overview related to the convex based multiple models approach. Section 3 gives a detailed explanation on how to implement the convex based multiple neural networks algorithm. A mathematical explanation is also given for explaining the advantages of using this new approach. Experiments on SLU and image recognition are given in section 4.

2 BACKGROUND

In this section, a brief background of the convex based multiple models structure is given, and how the structure can be used in the realm of deep learning.

The structure of convex based multiple models was firstly discussed in [8] for system identification in adaptive control. It uses the convex information of k identification models $\hat{f}(\hat{\theta}^i)(i = \{1 \cdots k\})$ to identify an unknown dynamic plant $f(\theta)$, where θ is an unknown system parameter to be identified. To represent the multiple models structure mathematically, at any time t, it has:

$$\sum_{i=1}^{k} \hat{\alpha}_t^i \hat{f}(\hat{\theta}_t^i) = \hat{f}(\hat{\theta}_t) \tag{1}$$

where $\hat{\alpha}_t^i$ is the convex coefficient corresponding to the i^{th} identifier $\hat{f}(\hat{\theta}_t^i)$, and $f(\hat{\theta}_t)$ converges to $f(\theta)$ when $t \to \infty$. The parameter $\hat{\alpha}_t^i$ also satisfies the convex criterion, *i.e.*, $\sum_{i=1}^{k} \hat{\alpha}_t^i = 1$ at $\forall t \in [0, \infty)$. The learning algorithm for calculating $\hat{\alpha} = [\hat{\alpha}^1, ..., \hat{\alpha}^k]$ is designed as:

$$\dot{\hat{\alpha}} = -\mathbf{E}\mathbf{E}^{\mathbf{T}}\hat{\alpha} + \mathbf{E}e^k \tag{2}$$

where $\mathbf{E} = [e^1 - e^k, \cdots, e^{k-1} - e^k]$, and e^i is the difference between $\hat{f}(\hat{\theta}^i)$ and $f(\theta)$. The details of derivation of (20) and proof of its convergence can be referred to the paper [8, 25]. The multiple models approach using the adaptive algorithm has demonstrated a better performance for system identification in terms of convergence speed and robustness.

Despite decent performance has been obtained by using convex based multiple models in system identification, it has never shown that this multiple models approach is feasible to deep learning. The obstacles are mainly from three aspects:

1. The adaptive learning algorithm in system identification are smooth continuous differential equations, whereas training the data-driven machine learning models should be in a discrete data-driven manner.
2. The inputs between a system identification model and a deep learning model are very different. The input of the former task is normally functional based (either continuous or discrete), but that of the later model are normally very sparse data which may not form a function.
3. The outputs between a system identification model and a deep learning model can also be very different. The predicted outputs from a deep learning model are mostly probabilistic based, *i.e.*, the outputs are probabilities for different labels. However, the outputs of a system identification model are mostly function-generated deterministic values.

Due to these three major differences between system identification and deep learning tasks, it is very hard to directly apply the convex based multiple models algorithm into the field of machine learning, and even harder to transfer the algorithm to deep neural networks.

To overcome these difficulties, in this paper, we propose a new convex based multiple models algorithm, which can be used for general deep neural network based tasks. In next section, we will give a detailed mathematical explanation of the proposed algorithm, and how to train it using two types of loss functions.

3 A CONVEX BASED MULTIPLE NEURAL NETWORKS STRUCTURE

In this section, a new convex based multiple neural networks structure is proposed. Its structure is as shown in Figure 1. In Figure 1, $x \in X$ represent the input training data and $y \in Y$ are their corresponding output labels. N^i stands for the i^{th} neural network among a total of k networks, which have the same structure but different initial weights values. For any input-output data pair $(x, y), x \in X$ and $y \in Y$, $N^i(\mathbf{w^i}, x)$ is a neural network based estimator to match the input x to its output label y, and $\mathbf{w^i}$ is the target weights of the i^{th} network. During the training procedure, $\hat{\mathbf{w}}^\mathbf{i}$ is an estimation of $\mathbf{w^i}$, and $\hat{y}^i$ is the estimated output generated using $N^i(\hat{\mathbf{w}}^\mathbf{i}, x)$. To represent it mathematically:

$$\hat{y}^i = N^i(\hat{\mathbf{w}}^\mathbf{i}, x) \tag{3}$$

In order to train a deep neural network model shown in (3), the conventional deep learning algorithm is to update $\hat{w}^i$ through back-propagating the output error $e_t^i = \hat{y}_t^i - y_t$, such that the predicted output $\hat{y}_t^i$ will gradually approach its true label y_t when $t \to \infty$.

In our new structure, instead of using each $\hat{y}^i$ as the predicted value of y^i, a convex combination of the outputs from k prediction models will be used as an estimation of y_t at each time step t:

$$\hat{y}_t = \sum_{i=1}^{k} \hat{\alpha}_t^i N^i(\hat{\mathbf{w}}_t^\mathbf{i}, x_t) \tag{4}$$

where $\hat{y}_t^i \to y_t$ if $\hat{\alpha}_t^i \to \alpha^i$. $\hat{\alpha}_t^i$ is an estimator of the target convex coefficient α^i at time step t satisfying:

$$y_t = \sum_{i=1}^{k} \alpha_t^i N^i(\mathbf{w}^\mathbf{i}{}_t, x_t) \tag{5}$$

The target convex coefficient α^i satisfies the convex criteria as:

$$\sum_{i=1}^{k} \alpha^i = 1 \tag{6}$$

In order to derive the new algorithm based on the convex property, we also constrain the estimated convex coefficients $\hat{\alpha}_t^i$ at time step t within a convex domain, *i.e.*

$$\sum_{i=1}^{k} \hat{\alpha}_t^i = 1 \qquad \forall t \in T \tag{7}$$

where k is the total number of models.

3.1 Introducing Two Types of Loss Functions

In order to train the multiple model structure, two loss functions are used concurrently. One is the local cross entropy loss functions $\mathcal{L}_t^i$ for training each neural network individually using back propagation at each time step t, and the other is a global loss function g_t to train the convex coefficients α^i.

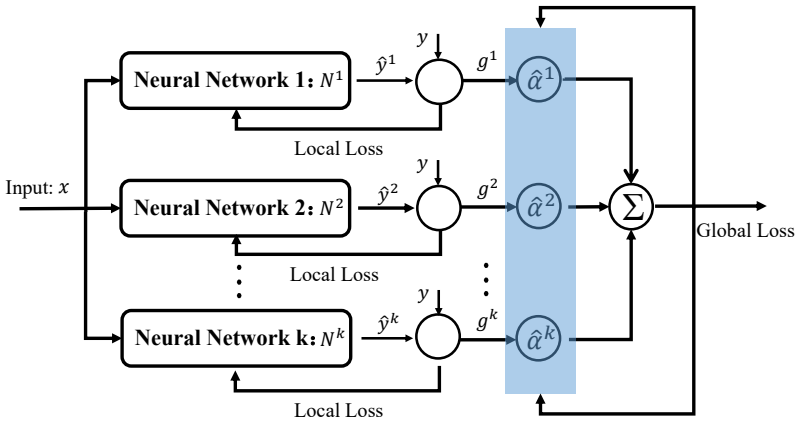

Figure 1: A new convex based multiple neural networks structure

The local loss function $\mathcal{L}_t^i$ is defined as the cross-entropy of outputs

$$\mathcal{L}_t^i = \sum_{j=1}^{n} y_{j,t}^i \log \hat{y}_{j,t}^i \tag{8}$$

where $y_{j,t}^i$ stands for the predicted probability of the j^{th} label for the i^{th} model at time step t, and n is the total number of label classes in output y^i.

The global loss function g_t is further defined as the difference between the true label y and the predicted label $\hat{y}$:

$$g_t = y_t - \hat{y}_t \tag{9}$$

By substituting (4) into (9), and combining the convex property as shown in (7), the global loss function can be rewritten as:

$$\begin{aligned} g_t &= y_t - \hat{y}_t \\ &= y_t - (\hat{\alpha}_t^1 N_t^1 + \hat{\alpha}_t^2 N_t^2 + \cdots + \hat{\alpha}_t^k N_t^k) \\ &= \sum_{i=1}^{k} \hat{\alpha}_t^i y_t - (\hat{\alpha}_t^1 N_t^1 + \hat{\alpha}_t^2 N_t^2 + \cdots + \hat{\alpha}_t^k N_t^k) \\ &= \sum_{i=1}^{k-1} \hat{\alpha}_t^i g_t^i + \hat{\alpha}_t^k g_t^k \\ &= \sum_{i=1}^{k-1} \hat{\alpha}_t^i g_t^i + (1 - \sum_{i=1}^{k-1} \hat{\alpha}_t^i) g_t^k \\ &= \sum_{i=1}^{k-1} \hat{\alpha}_t^i \tilde{g}_t^i + g_t^k \end{aligned} \tag{10}$$

where N^i is an abbreviation of $N^i(\hat{\mathbf{w}}_\mathbf{t}^\mathbf{i}, x_t)$. g_t^i is defined as the difference between the i^{th} model's output and the target output, *i.e.* $g_t^i = y_t - N_t^i$. $\tilde{g}_t^i$ is defined as:

$$\tilde{g}_t^i = g_t^i - g_t^k \tag{11}$$

which is the difference between the i^{th} model loss function and the last k^{th} model loss function.

The loss function derived from (10) can be further simplified as:

$$\tilde{g}_t = \tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T} \tilde{\alpha}_\mathbf{t} \tag{12}$$

where $\tilde{g}_t = g_t - g_t^k$, $\tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T} \in \mathbb{R}^{m \times (k-1)}$ (m output labels and k models) is defined as $\tilde{\mathcal{G}}_\mathbf{t}$=$[\tilde{g}_t^1, \tilde{g}_t^2, \cdots, \tilde{g}_t^{k-1}]$, and the convex coefficient vector is defined as $\tilde{\alpha}_\mathbf{t}$=$[\hat{\alpha}_t^1, \cdots, \hat{\alpha}_t^{k-1}] \in \mathbb{R}^{(k-1)\times 1}$.

3.2 Training using Global Loss Function

As discussed earlier, two types of loss functions are used to train the system in a concurrent manner. The local loss function $\mathcal{L}_t^i$ is used for training each individual network $N^i(\hat{\mathbf{w}}_t^\mathbf{i}, x_t)$ by adjusting the adaptive weights $\hat{\mathbf{w}}_\mathbf{t}^\mathbf{i}$ through back-propagation, which is a standard training mechanism. On the other hand, the global loss function g_t together with its derived loss function vector $\tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T}$ will be used to train the convex coefficient vector $\tilde{\alpha}_\mathbf{t}$ at the same time.

As the training using local loss function $\mathcal{L}_t^i$ is a standard back-propagation neural network training, the paper's emphasis is not on this and will omit the discussion. Our discussion is mainly on how to use the global loss function vector $\tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T}$ to train the convex coefficients, such that it can overcome the non-convex optimization problem and also further improve the training performance of the neural network.

Multiplying $\tilde{\mathcal{G}}_\mathbf{t}$ on both sides of (12), it becomes:

$$\tilde{\mathcal{G}}_\mathbf{t} \tilde{g}_t = \tilde{\mathcal{G}}_\mathbf{t} \tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T} \tilde{\alpha}_\mathbf{t} \tag{13}$$

Then by moving left hand side element to the right of equation, the learning rule of the convex coefficients can be derived as:

$$\delta \tilde{\alpha}_\mathbf{t} = -\tilde{\mathcal{G}}_\mathbf{t} \tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T} \tilde{\alpha}_\mathbf{t} + \tilde{\mathcal{G}}_\mathbf{t} \tilde{g}_t \tag{14}$$

where $\delta \tilde{\alpha}_\mathbf{t} \triangleq \tilde{\alpha}_{\mathbf{t+1}} - \tilde{\alpha}_\mathbf{t}$, hence

$$\tilde{\alpha}_{\mathbf{t+1}} = \tilde{\alpha}_\mathbf{t} - \tilde{\mathcal{G}}_\mathbf{t} \tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T} \tilde{\alpha}_\mathbf{t} + \tilde{\mathcal{G}}_\mathbf{t} \tilde{g}_t \tag{15}$$

Equation (15) is the new learning law for the convex coefficients $\tilde{\alpha}_\mathbf{t}$.

Remarks: It is worth to notice that:

1. At each time step t, the estimated convex coefficients satisfy $\sum_{i=1}^{k} \hat{\alpha}_t^i = 1$, hence $\hat{\alpha}_t^k = 1 - \sum_{i=1}^{k-1} \hat{\alpha}_t^i$, i.e. *only $k-1$ convex coefficients are needed to be calculated at each time step t.*

2. In order to form a convex hull based on the outputs of multiple

Algorithm 1 Convex based multiple neural networks

Input Data: X, Output Labels: Y
Initialize $N^i(i = \{1, \cdots, k\})$ with the same NN structures.
number of iteration = $iter$,$n = 0$
Initialize the weight vector of N^i: $\mathbf{w_0^i}$
Initialize the convex coefficients of N^i: $\hat{\alpha}_0^i$
while $t < iter$ **do**
  Choose x_t from X, its target label is y_t
  for $i = 1 \rightarrow k$ **do**
    $\hat{y}_t^i \leftarrow N^i(\hat{\mathbf{w}}_\mathbf{t}^\mathbf{i}, x_t)$
    $g_t^i \leftarrow y_t - \hat{y}_t^i$
    Local loss: $\mathcal{L}_t^i \leftarrow \sum_{j=1}^m y_{j,t}^i \log \hat{y}_{j,t}^i$
    Back-propagate training $\mathbf{w_t^i}$ using $\mathcal{L}_t^i$
  end for
  Global loss: $g_t \leftarrow \sum_{i=1}^k \hat{\alpha}_t^i g_t^i$
  $\tilde{g}_t^i \leftarrow g_t^i - g_t$
  $\tilde{\mathcal{G}}_\mathbf{t} \leftarrow [\tilde{g}_t^1, \tilde{g}_t^2, \cdots, \tilde{g}_t^{k-1}]$
  Convex coefficient training: $\tilde{\alpha}_{\mathbf{t+1}} \leftarrow \tilde{\alpha}_\mathbf{t} - \tilde{\mathcal{G}}_\mathbf{t}\tilde{\mathcal{G}}_\mathbf{t}^\mathbf{T}\tilde{\alpha}_\mathbf{t} + \tilde{\mathcal{G}}_\mathbf{t}\tilde{g}_t$
  $n \rightarrow n + 1$
end while

neural networks, the total number of models, k, needs to satisfy an inequality involved the total number of output labels n:

$$k \geqslant n + 1 \tag{16}$$

This means that the total number of neural network models needs to be larger than the number of labels in order to form a convex hull using the output of multiple neural networks. Though this property seems to increase the total number of parameters, in the following experiment section, it will demonstrate that a smaller hidden layer size (or cell units in LSTM) in multiple neural networks can be used for our new structure, to achieve a similar or even better performance. It needs to clarify that the model can still be trained if property 2 is not satisfied, but may result in an inferior performance. It is because that there will be no real solutions for the convex coefficients α^i if $k < n + 1$, but only numerical approximations.

In the next part, a detailed explanation on synchronized training using both local and global loss functions will be given.

3.3 Synchronized Training using Local and Global Loss Functions

The definition of local loss $\mathcal{L}_t^i$ and global loss vector $\tilde{\mathcal{G}}_t$ are given in last section. It is also explained that how to train the convex coefficient α^i based on the global loss g_t and its vector $\tilde{\mathcal{G}}_t$. The combined synchronized training using local and global loss functions is going to be discussed in this section in detail. Algorithm 1 shows the training flow for the convex based multiple neural networks.

At each iteration, each of the neural networks will be trained through back-propagation using their local losses $\mathcal{L}_t^i$ first, then a global loss vector $\tilde{\mathcal{G}}_t$ generated by the collected information from the outputs of multiple neural networks will be used to train the convex coefficients $\hat{\alpha}^i$. A detailed step by step explanation of the training procedure is as shown in Algorithm 1.

3.4 Performance Analysis

In our new proposed convex based multiple neural networks structure, two important performance features need to be noticed:
1. The back-propagation used in gradient descent optimization for each individual network involves calculating the gradient of nonlinear activation functions, which is much slower than that of adjusting the convex coefficients $\hat{\alpha}$. As shown in (15),comparatively, the update of $\hat{\alpha}$ is a linear difference equation, which gives an exponential convergence/learning speed of the convex coefficient $\hat{\alpha}$.
2. The convex output $\hat{y}_t$ generated by multiple neural networks will be close to the true label y_t once $\hat{\alpha}_t$ converges, *i.e.* $\hat{y}_t = \sum_{i=1}^k \hat{\alpha}_t^i N^i(\hat{\mathbf{w}}_\mathbf{t}^\mathbf{i}, x_t) \cong y_t$, once $\hat{\alpha}_t \cong \alpha$. A simple theoretical explanation is given by the facts that: the global loss g_t is a convex combination of each loss g_t^i, i.e. $g_t = \sum_{i=1}^k \alpha_t^i g_t^i$. Once the $\hat{\alpha}_t$ converges, which indicates that the learning gradient approaches to zero as shown in (15), i.e. $-\tilde{\mathcal{G}}_t\tilde{\mathcal{G}}_t^T \rightarrow 0$. This means that $g_t \rightarrow 0$ as $\tilde{\mathcal{G}}_t$ is a linear matrix of g_t.

The above two features give a clear explanation that why the convex based approach can give a faster learning speed, and the model can also avoid the gradient vanishing problem in some sense as the model will be not affected by the local loss since the global loss will converge much faster than the local losses.

Remarks: Besides the applications on system identification tasks as described in: [25, 33], multiple models based algorithms has been also widely applied on a variety of reinforcement learning structures as in [9, 23, 26, 29, 34, 35] and NLP applications, like spoken language understanding [36, 38], entity recognition [36], and visual question understanding [37]. In our model, we proposed a general structure that can be used for different tasks as to be shown in the experiment section.

4 EXPERIMENT

In this section, we perform experiments on two well-knonw public datasets ATIS and MNIST [16, 28], for spoken language understanding (SLU) and hand written digits recognition. For each task, we compare the test results using single neural network and convex based multiple neural networks structure. The experiment on ATIS dataset is to build an NN model to generate the intent and slot tags for a given utterance. The previous state-of-the-art result is obtained by using attention based Bi-LSTM model in [20]. Without loss of generality, we are also using the attention bi-direction LSTM as each individual network in our multiple neural networks structure for a fair comparison. The experiment on MNIST dataset is a classification task to find the number zero to nine from images. The baseline single model is chosen as the DropConnnect network [32], which demonstrates best accuracy on the task so far. Since the SLU task using ATIS dataset is not as well known as the MNIST image recognition task, a detailed SLU task description is given as follows.

4.1 Description of two SLU tasks: Intent Detection and Slot Filling

Two important tasks in spoke language understanding are intent detection and slot filling. The first task is to detect the intent of a query and the second task is to label the input utterance with correct slot tags. For example, Figure 2 is an example of part of the utterance from ATIS dataset [28] with its intent and the corresponding slot labels.

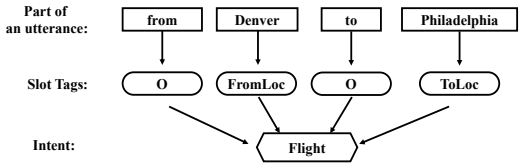

Figure 2: Utterance example for intent classification and slot tagging

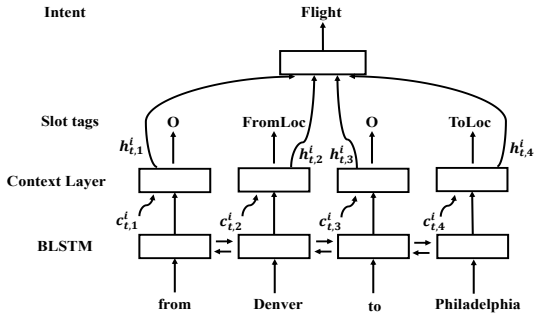

Figure 3: A single attention based BLSTM (A-BLSTM) for joint intent detection and slot filling

In [20], it is suggested using one joint recurrent neural network model to learn two tasks together. By taking advantage of an attention based Bi-LSTM (A-BLSTM) structure, it gives the state-of-art results on ATIS dataset. A brief overview of the structure is as shown in Figure 3, where $c^i_{t,j}$ is the attention based contextual information [2]. In our experiment, we will choose each single network in our multiple model structure as the same attention based Bidirectional LSTM (A-BLSTM) structure shown in Figure 3, and further compared our performance with the single A-BLSTM structure in [20].

4.1.1 Data. The ATIS dataset used in this experiment follows the same format as in [19–21, 41]. The training set contains 4978 utterance and the test set contains 893 utterance, with a total of 18 intent classes and 127 slot labels. It is worth noticing that the slot tags follows the BIO encoding, which requires both B and I tags to be correctly tagged for a slot to be considered correct.

4.1.2 Convex based Multiple A-BLSTMs (CMA-BLSTM) for Joint Intent Detection and Slot Filling. In this section, we will discuss how to apply the convex based multiple neural networks in joint intent detection and slot filling, by using A-BLSTM as a basic structure for each neural network. Without loss of generality, all the model illustration are using ATIS dataset as an example. The set-up for the models on our in-house dataset will be similar. A graphical illustration is given in Figure 4. Since the total number of intent classes for ATIS dataset is 18, 19 models will be used to guarantee that a convex solution can be obtained for α^i, that is, $\sum_{i=1}^{m} \alpha^i = 1 (m = 19)$.

For each individual network of the structure, an attention based BLSTM (A-BLSM) shown as in Figure 3 is used. At each iteration t, an input utterance $x_t = [x_t^1, \cdots, x_t^l]$ with l words will be fed into multiple A-BLSTMs word by word. Each word is represented by a word vector generated by the embedding layer. A sequence of predicted slot labels are generated as $\hat{s}_t^i = [\hat{s}_{t,1}^i, \cdots, \hat{s}_{t,l}^i]$ for the i^{th} model, where $i \in \{1, \cdots, 19\}$. The output of each A-BLSTM after reading in all the words in x_t is the predicted intent label $\hat{y}_t^i$. The final estimated output $\hat{y}_t$ at iteration t is equal to the convex sum of the outputs of 19 A-BLSTMs:

$$\hat{y}_t = \sum_{i=1}^{19} \hat{\alpha}_t^i \hat{y}_t^i \tag{17}$$

The loss function of the convex based multiple attention BLSTMs (CMA-BLSTM) structure is also divided into two parts as described earlier: the local losses $\mathcal{L}_t^i$ and a global loss g_t with its correspond loss vector $\mathcal{G}_t$. Specifically in this task, the local loss function for the i^{th} network can be defined as the sum of the loss of intent detection task($\mathcal{L}_{1,t}^i$) and slot filling task($\mathcal{L}_{2,t}^i$):

$$\begin{aligned} \mathcal{L}_t^i &= \mathcal{L}_{1,t}^i + \mathcal{L}_{2,t}^i \\ &= y_t^i \log(\hat{y}_t^i) + \sum_{j=1}^{l} s_{t,j}^i \log(\hat{s}_{t,j}^i) \end{aligned} \tag{18}$$

The local loss functions are used to train each individual attention based BLSTM(A-BLSTM) through back-propagation.

The global loss function is defined as

$$\begin{aligned} g_t &= y_t - \hat{y}_t \\ &= y_t - (\hat{\alpha}_t^1 \hat{y}_t^1 + \hat{\alpha}_t^2 \hat{y}_t^2 + \cdots + \hat{\alpha}_t^{19} \hat{y}_t^{19}) \end{aligned} \tag{19}$$

The global loss vector $\tilde{\mathcal{G}}_t$ cam be derived from g_t, hence giving the convex based learning algorithms as

$$\tilde{\alpha}_{\mathbf{t+1}} = \tilde{\alpha}_{\mathbf{t}} - \tilde{\mathcal{G}}_{\mathbf{t}} \tilde{\mathcal{G}}_{\mathbf{t}}^{\mathbf{T}} \tilde{\alpha}_{\mathbf{t}} + \tilde{\mathcal{G}}_{\mathbf{t}} \tilde{g}_t \tag{20}$$

The subsequent training procedure follows the steps shown in Algorithm 1. The outputs of the convex based multiple A-BLSTMs (CMA-BLSTM)include two parts: one is the predicted slot filling vector $\hat{s}_t$, and the other is the intent label predicted $\hat{y}_t$, as shown in Figure 4. The performance is evaluated based on the classification accuracy for intent detection task and F1-score for slot filling task.

4.1.3 Experiment Setup and Results. To compare the performance of our new convex based multiple attention based BLSTMs (CMA-BLSTM) structure with the results obtained using A-BLSTM in [20], the same or smaller numbers of units in an LSTM cell are chosen for CMA-BLSM, as 128, 64, 32 and 16, while A-BLSM uses a total numbers of 128 units. Based on the size of our dataset, the number of hidden layers is chosen as 1. The size of word embedding is 128, which are initialized randomly at the beginning of experiment. The weight vectors $\mathbf{w^i}$ are also initialize randomly with different values. The experiments are conducted on an NVIDIA M40 GPU.

Firstly, in order to demonstrate the speed of the new proposed algorithm, we will compare the model based on two metrics:
1. The number of epochs needed for convergence using different algorithms. An early stopping mechanism is applied here if there are five consecutive epochs without any improvement.
2. Since the training time for each epoch using different models may be different, we also recorded the total training time needed for different models using the same NVIDIA M40 GPU.

A comparison of results is summarized in Table 1.

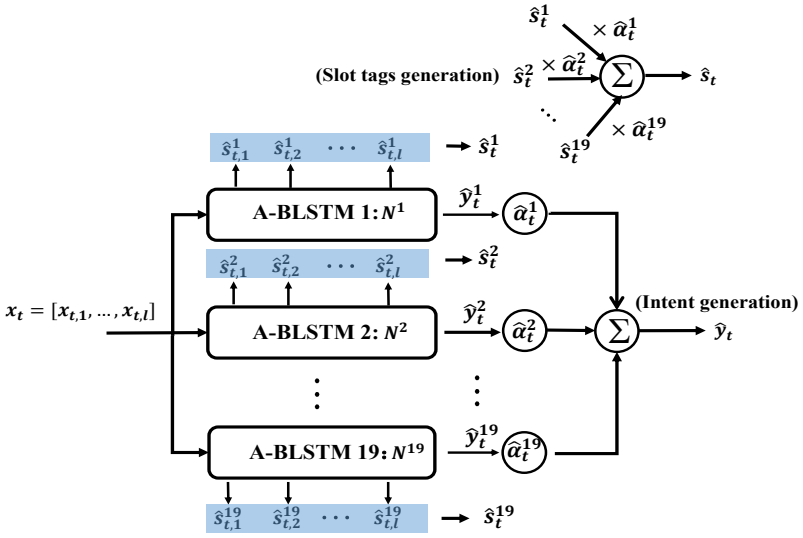

Figure 4: Convex based multiple A-BLSTMs structure for joint intent detection and slot filling

Table 1: Training speed comparison using different models on ATIS Dataset

Model	Epochs	Training time (sec)	Speedup
Recursive NN [7]	121	122.5	6.41 X
Attention Encoder-Decoder NN [20]	145	135.6	7.09 X
Attention BLSTM (A-BLSTM)(n=128) [20]	165	158.6	8.30 X
CMA-BLSTM (n=16)	**13**	**19.1**	1 X
CMA-BLSTM (n=32)	**18**	**25.8**	1.35 X
CMA-BLSTM (n=64)	**24**	**33.2**	1.74 X
CMA-BLSTM (n=128)	**28**	**38.6**	2.02 X

In Table 1, it demonstrates a comparison of convergence speed by using CMA-BLSTMs with that of using A-BLSTM and two other RNN based approaches, on public ATIS dataset. One can observe that, depending on the number of units (n) in an LSTM cell, the convergence epochs needed for the CMA-BLSTM based structure is only about $\frac{1}{8}$ to $\frac{1}{4}$ of the training time needed for the other three approaches to converge. The main reason is that the global convex coefficients $\hat{\alpha}^{\mathbf{i}}$ converge much faster than the weights $\hat{\mathbf{w}}^{\mathbf{i}}$ in each network. Once the convex coefficients converge to their true values, the predicted outputs of models will be close to the target outputs even before each individual network in the multiple networks structure converges, as explained in section 3.4.

It is also observed that, when using CMA-BLSTM, the smaller number of units n used, the faster of the convergence speed can achieve. Although the number of training epoch needed for model convergence is a main criterion to evaluate the efficiency and learning speed of different models, the comparison is only fair and meaningful when the results are based on a similar or better task performance. Hence, in Table 2, another comparison for different models is given by evaluating their intent detection accuracy and slot filling F1 scores on two different datasets.

Table 2: Performance of Different Models on ATIS Dataset

Model	F1 Score	Intent Accuracy
Recursive NN [7]	93.96%	95.4%
RNN with Label Sampling [19]	94.89%	NA
Hybrid RNN [21]	95.06%	NA
RNN-EM [27]	95.25%	NA
CNN CRF [41]	95.35%	NA
Encoder-labeler Deep LSTM [14]	95.66%	NA
Attention Encoder-Decoder NN [20]	95.87%	98.43%
A-BLSTM (n=128) [20]	95.98%	98.21%
A-BLSTM with three layers (n=128)	96.20%	98.32%
CMA-BLSMs (n=16)	95.75%	98.10%
CMA-BLSMs (n=32)	**96.32%**	**98.54%**
CMA-BLSMs (n=64)	**96.53%**	**98.65%**
CMA-BLSMs (n=128)	**96.89%**	**98.88%**
Convex based Multiple Recursive NN	95.78%	96.94%
Convex based Multiple Attention Encoder-Decoder NN (n=128)	**96.32%**	**98.65%**

Results in Table 2 show that, except the case when the smallest number of units used ($n = 16$), the performance of CMA-BLSM with $n = \{32, 64, 128\}$ is better than the current state-of-the-art result as shown in [20]. Also to further demonstrate the our performance improvement is not due to the increasing number of parameters, we also compare increase the number of layers in current state-of-art results to three, the MM approach still outperform it starting from $n = 32$. The results for $n = 16$ are also very close to those using single A-BLSTM. From this comparison, it demonstrates that the CMA-BLSM can achieve a better or faster result compared to a single attention based BLSM, with a need of only fewer than $\frac{1}{4}$ training time.

In order to demonstrate the convex based MM approach can improve model performance in general, we also applied the MM

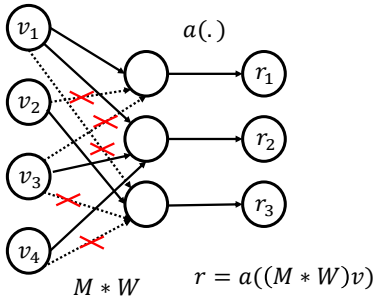

Figure 5: A single DropConnect Neural Network Structure

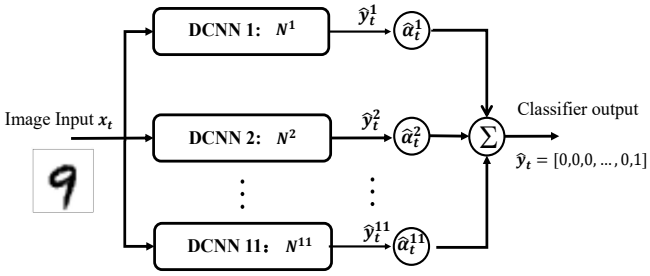

Figure 6: Convex based multiple DropConnect NN (CM-DCNN Structure

algorithm to two other models [7, 20] as shown in Table 1, the result is as given in Table 2. (Due to the space limiation, only $n = 128$ is used for the model "Attention Encoder-Decoder NN"). It can be observed that both models' performance in [7, 20] has been improved after using the convex based multiple models approach.

4.2 Descriptions of MNIST Dataset and DropConnect

To demonstrate the generality of our appraoch, we also apply the technique to a hand written digits recognition task, *i.e.* recognizing the handwritten digits images using MNIST dataset[16] . The MNIST dataset consists of 28×28 black and white images, each containing a digit 0 to 9 (10-classes). Each digit in the 60, 000 training images and 10, 000 test images is normalized to fit in a 20×20 pixel box while preserving their aspect ratio. Currently the best performed model is the DropConnect Neural Networks (DCNN) in [32], which gives 0.21% error rate.

4.2.1 DropConnect. DropConnect is the generalization of Dropout in which each connection, rather than each output unit, can be dropped with probability $1 - p$. DropConnect is similar to Dropout as it introduces dynamic sparsity within the model, but differs in that the sparsity is on the weights $\mathbf{W}$, rather than the output vectors of a layer. For a DropConnect layer, the output r is given as:

$$r = a((\mathbf{M} \times \mathbf{W})\mathbf{v}) \tag{21}$$

where $a(\cdot)$ is a nonlinear activation function, v is the input, and $\mathbf{M}$ is a binary matrix encoding the connection information and $\mathbf{M_{ij}}$ follows the Bernoulli distribution of p, *i.e.* $\mathbf{M}$ ~Bernoulli(p). The graphical illustration is given in Figure 5, where the redcross indicates the connections we droped. Due to the space limitation, we will not describe the structure in detail. A full explanation of DropConnect can be found in [32].

4.2.2 Convex based multiple DropConnect (CM-DCNN). In this section, similar to the multiple model structure as given in the SLU case, we will describe how to construct the convex based multiple model structure using DropConnect (CM-DCNN) for digit recognition. As shown in Figure 6, there are m DropConnect networks running in parallel. The number of m networks is chosen as 11 in this case, since there are 10 digits need to be recognized, *i.e.* $\hat{y}_t^i \in \mathbb{R}^{10}$. Similarly, the outputs of the 11 networks $\hat{y}_t^i$ are combined in a convex manner to generate the final classifier output $\hat{y}_t$:

$$\sum_{i=1}^{11} \hat{\alpha}_t^i \hat{y}_t^i = \hat{y}_t \tag{22}$$

where $\hat{\alpha}_t^i$ are adaptively adjusted using the global loss as in (20).

4.3 Experiment Setup and Results

To compare the performance of our new convex based multiple DropConnect neural networks (CM-DCNN) with the results obtained using a single DropConnect neural network, we use the same feature extractor network described in [4]. This feature extractor consists of a 2 layer CNN with 32-64 feature maps in each layer respectively. The last layer's output is treated as the input to the fully connected layer which has 150 *relu* units on which DropConnect are applied. Since the total number of classes for MNIST dataset is 10, 11 models will be used to guarantee that a convex solution can be obtained for α^i, that is, $\sum_{i=1}^{m} \alpha^i = 1 (m = 11)$. The global loss and the learning rule of the convex coefficient are as defined in (10) and (15), where $m = 11$. The learning rate is selected as 0.01 for all models and the experiments are conducted using a single NVIDIA M40 GPU.

Similar to the SLU experiment, we also compare models' performance based on the two metrics, *i.e.* the number of training epochs for convergence and the total training time needed. Due to space limitation and fair comparison purpose, we only use the same CNN setup as in[32] without changing the number of feature maps . A comparison of results is summarized in Table 3. It shows that our convex based model only takes about $\frac{1}{4}$ of the training time compare to the single DropConnect Structure, and up to 30x speed-up compared to other baseline models.

Finally, we compare the error rate on the standard MNIST dataset obtained by using CM-DCNN and state-of-the-art approaches, which is given in Table 4. In our case, we use m=11 models in order to achieve the 0.19% error rate.

From Table 4, it is observed that the convex based multiple DropConnect gives an even better result compared to the current state-of-the-art result. This is mainly because that the collective information from multiple networks help correct the errors made by each individual networks to some extent, as we explained in the *performance analysis* section.

Table 3: Training speed comparison using different models on MNIST Dataset

Model	Epochs	Training time (sec)	Speedup
Recursive CNN [18]	486	886.5	13.7 X
Gated Pooling CNN [17]	724	1242.3	19.66 X
Multi-Column DNN [4]	1008	1853.6	29.33 X
DropConnect [32]	136	268.8	4.25 X
Convex based Multiple DropConnect NN (m=11)	**24**	**63.2**	1.0 X

Table 4: Performance on MNIST Dataset using different models

Model	Error (%)
Recursive CNN [18]	0.31
Gated Pooling CNN [17]	0.29
Multi-Column DNN [4]	0.23
DropConnect [32]	0.21
Convex based Multiple DropConnect NN	**0.19**

Remarks:
It is worth noticing that, to compare it fairly, the training environments are the same for all models by using the same tensorflow [1] version, and python [30] as front end language. Also, the GPU environment is chosen as the same single NVIDIA M40 GPU.

4.3.1 Performance comparison based on convergence errors. In order to better understand the convergence speed of the convex based multiple model approach to that of the single model approach, a graphical comparison of the convergence speeds of their output errors is given in this section. In order to evaluate the changes of errors in terms of training time, we computing the error using a time sliding window approach instead of using the epoch, such that the errors can be evaluated in the real-world time for comparison purpose. The error is defined as the signed mean square error as in (23). DropConnect case.

$$\begin{aligned} e_t &= (sign)\frac{1}{K(t)}\sum_{k=0}^{K(t)-1}||(\hat{y}_{t-k}-y_{t-k})||_2 \quad \text{(DCNN)} \\ &= (sign)\frac{1}{K(t)}\sum_{k=0}^{K(t)-1}||(\sum_{i=1}^{m}\alpha^i_{t-k}\hat{y}^i_{t-k}-y_{t-k})||_2 \quad \text{(CM-DCNN)} \end{aligned} \tag{23}$$

where $K(t) = N(t - w_0, t)$ is the number of samples trained in the time window span $[t - w_0, t]$, and w_0 is the window size selected by the algorithm designer. In our experiments, we choose the $w_0 = 1$ in the unit of second (sec). The sign is defined by using the normalized dot product of two vectors $\hat{y}_t$ and y_t, *i.e.*

$$sign = \begin{cases} 1 & \text{if } \frac{\hat{y}_t \cdot y_t}{|\hat{y}_t||y_t|} \geq 0 \\ -1 & \text{if } \frac{\hat{y}_t \cdot y_t}{|\hat{y}_t||y_t|} < 0 \end{cases}$$

The reason we want to use the signed error instead of MSE itself is because that we would also like to monitor the direction change of our training output $\hat{y}_t$ towards y_t. The error plot is as given in Figure 7. It is shown in the figure that the MSE error of CM-DCNN converges much faster than that of the original DCNN on the MNIST task, which follows the training time results in Table 3. On the other hand, it is noticed that there are rapid changes of the sign of error when we use the convex based multiple DCNNs. This indicates that the direction of error oscillates in a very fast manner as slight changes of the convex coefficiens α will lead to a large overshoot of the convex combined output $\hat{y}_t$. Despite this oscillation if we consider the sign of error, the converge speed of CM-DCNN far surpass that of DCNN by over 4 times.

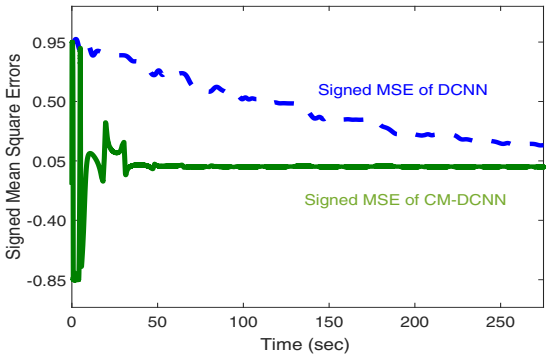

Figure 7: Convergence of Errors using DCNN and CM-DCNN

5 CONCLUSION

In this paper, a new concept of convex based multiple neural networks structure is proposed. We applied this new technique on two tasks: one is a spoken language understanding (SLU) task for intent detection and slot filling, the other is an image recognition task to recognize handwritten digits. The new proposed algorithm demonstrated a much time faster learning speed on both tasks compared to the models with state-of-the art performances. Also, on the SLU task, we show that the new model can give a decent and even better result (compared to a single model), when we further decrease the size of each network to reduce the training time. From both theoretical and experimental perspectives, it can conclude that this new convex based multiple models approach has a great potential in improving the learning speed on different applications using deep neural networks, and at the same time, this technique can further improve the model performance in terms of accuracy and F1 scores.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[4] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745* (2012).

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[7] Daniel Guo, Gokhan Tur, Wen-tau Yih, and Geoffrey Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 554–559.

[8] Zhuo Han and Kumpati S Narendra. 2012. New concepts in adaptive control using multiple models. *IEEE Trans. Automat. Control* 57, 1 (2012), 78–89.

[9] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. 2009. Multi-class adaboost. *Statistics and its Interface* 2, 3 (2009), 349–360.

[10] Donald Olding Hebb. 2005. *The organization of behavior: A neuropsychological theory*. Psychology Press.

[11] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, 02 (1998), 107–116.

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

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[14] Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging Sentence-level Information with Encoder LSTM for Semantic Slot Filling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2077–2083.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[16] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[17] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. 2016. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics*. 464–472.

[18] Ming Liang and Xiaolin Hu. 2015. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3367–3375.

[19] Bing Liu and Ian Lane. 2015. Recurrent neural network structured output prediction for spoken language understanding. In *Proc. NIPS Workshop on Machine Learning for Spoken Language Understanding and Interactions*.

[20] Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454* (2016).

[21] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 23, 3 (2015), 530–539.

[22] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[23] Kumpati S Narendra, Snehasis Mukhopadyhay, and Yu Wang. 2015. Improving the speed of response of learning algorithms using multiple models. *arXiv preprint arXiv:1510.05034* (2015).

[24] Kumpati S Narendra and Kannan Parthasarathy. 1990. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks* 1, 1 (1990), 4–27.

[25] Kumpati S Narendra, Yu Wang, and Wei Chen. 2014. Stability, robustness, and performance issues in second level adaptation. In *American Control Conference (ACC), 2014*. IEEE, 2377–2382.

[26] Kumpati S Narendra, Yu Wang, and Snehasis Mukhopadhay. 2016. Fast Reinforcement Learning using multiple models. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 7183–7188.

[27] Baolin Peng and Kaisheng Yao. 2015. Recurrent neural networks with external memory for language understanding. *arXiv preprint arXiv:1506.00195* (2015).

[28] Patti J Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

[29] Gunnar Rätsch, Takashi Onoda, and K-R Müller. 2001. Soft margins for AdaBoost. *Machine learning* 42, 3 (2001), 287–320.

[30] Guido Van Rossum and Fred L Drake Jr. 1995. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

[31] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. 1990. Phoneme recognition using time-delay neural networks. In *Readings in speech recognition*. Elsevier, 393–404.

[32] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*. 1058–1066.

[33] Yu Wang. 2017. A new concept using LSTM Neural Networks for dynamic system identification. In *2017 American Control Conference (ACC)*. IEEE, 5324–5329.

[34] Yu Wang and Jin Hongxia. 2019. A Deep Reinforcement Learning based Multi-Step Coarse to Fine Question Answering (MSCQA) System. In *Thirty-third AAAI Conference on Artificial Intelligence*. AAAI.

[35] Yu Wang and Hongxia Jin. 2018. A boosting-based deep neural networks algorithm for reinforcement learning. In *American Control Conference (ACC)*.

[36] Yu Wang, Abhishek Patel, Yilin Shen, and Hongxia Jin. 2018. A Deep Reinforcement Learning based Multimodal Coaching Model (DCM) for Slot Filling in Spoken Language Understanding (SLU). *Proc. Interspeech 2018* (2018), 3444–3448.

[37] Yu Wang, Abhishek Patel, Yilin Shen, Hongxia Jin, and Larry Heck. 2018. An Interpretable (Conversational) VQA model using Attention based Weighted Contextual Features. In *The 2nd Conversational AI Worshop, NeurIPS*.

[38] Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A Bi-Model Based RNN Semantic Frame Parsing Model for Intent Detection and Slot Filling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, Vol. 2. 309–314.

[39] Paul Werbos. 1974. Beyond regression: new fools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University* (1974).

[40] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

[41] Puyang Xu and Ruhi Sarikaya. 2013. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 78–83.