FEEDFORWARD LEGENDRE MEMORY UNITS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, a new recurrent neural network (RNN) named the Legendre Memory Unit (LMU) was proposed and shown to achieve state-of-the-art performance on psMNIST and other datasets. Here we consider a modified version of the LMU, named ff-LMU, the core of which is a linear time-invariant (LTI) system. We first show that the ff-LMU can be trained in a purely feedforward manner and yet executed during inference in a recurrent fashion. Specifically we demonstrate that it trains about 80x faster than LSTM models of the same size. As a result, it overcomes the well-known limitations of training RNNs on GPUs that make them less scalable than feedforward networks like transformers. Second, to validate its utility, we compare ff-LMU performance against LSTMs on five benchmarks picked from the following categories: sentiment classification, semantic similarity, natural language inference, and image classification. Our models, despite their simplicity, achieve new state-of-the-art results for RNNs on psMNIST and QQP, and exhibit superior performance on the remaining three datasets while using up to 1000x fewer parameters. In general, ff-LMU models are highly parameter efficient. For instance, the first model that beats it on current leaderboards for QQP is a transformer that uses 50,000x more parameters.

1 INTRODUCTION

Recurrent neural networks (RNNs) were the go to architecture for processing time series data (Graves, 2013; Sutskever et al., 2014; Bahdanau et al., 2014) until recently. However, recent attention-based architectures, like the transformer and its derivatives (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020), have taken over as the preferred method for large-scale processing, especially in the domain of natural language processing (NLP). One critical reason for this shift away from RNNs is that transformer-like networks are purely feedforward. Consequently, training such networks is far more efficient on today's commodity GPU hardware because large datasets can be batch processed on large networks during training. The main drawback of such an approach is that during inference, transformer-like networks tend to have very large parameters counts, and so are very demanding of computational resources.

In contrast, commonly used RNNs (e.g. LSTMs, GRUs, etc.) tend to have much lower parameter counts, but they are very slow to train. This cost comes from having to simulate each cycle through the recurrent network over long sequences before being able to backpropagate the error during training. Equivalently, unrolling such networks leads to extremely deep networks for long time window data (which is common in NLP applications). Consequently each epoch through a large dataset quickly becomes prohibitively expensive in time. Additionally, there are concerns that standard RNNs do not handle long distance dependencies well in practice (Li et al., 2018) (e.g., for an LSTM time series memory is usually limited to well less than 1000 steps).

In this paper, we leverage a new RNN architecture called the Legendre Memory Unit (LMU) (Voelker & Eliasmith, 2018; Voelker et al., 2019) that outperforms past RNNs in terms of temporal memory capacity (several orders of magnitude improvement). However, the original LMU architecture still suffers from the challenges of training RNNs. As a result, here we explore an LMU variant, which we call the feedforward-LMU (or ff-LMU for short). In order to evaluate its effectiveness, we consider NLP datasets from each of the following three categories: sentiment classification (IMDB), semantic similarity (MRPC, QQP), and natural language inference (SNLI). Coincidentally, these tasks are similar to the ones considered by the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018). In addition, we also look at the standard psMNIST bench-

mark that is used to test the ability of RNNs to capture complex long term dependencies. We show that ff-LMU 1) can be trained in a purely feedforward manner 2) outperforms LSTM models on the IMDB, QQP, MRPC, and SNLI datasets while using significantly fewer parameters (up to 1000x); 3) achieves a new state-of-the-art result for RNNs on psMSNIT of 98.49%; and 4) realizes up to 80x faster training than the LSTM. As a result, we argue that our new LMU architecture is scalable, accurate, and efficient to train, addressing the core challengenes limiting the adoption of past RNNs.

2 THE LMU AND FF-LMU

2.1 THE LMU

The LMU is an RNN cell whose main component is a linear time-invariant (LTI) system that is coupled to a nonlinear dynamical system. The LTI system is a memory cell that orthogonalizes the input signal, $u(t) \in \mathbb{R}$, across a sliding window of length $\theta \in \mathbb{R}_{>0}$, whereas the non-linear system relies on this memory to compute arbitrary functions across time. It should be noted that in the LMU, relative to the LSTM, memory and computation are independent, i.e., we can increase the memory capacity of the cell by increasing the dimension of LTI system while leaving the dimension of the non-linear system unaltered. The state update equations that define the LMU are given below:

$$u_t = \boldsymbol{e}_x^T \boldsymbol{x}_t + \boldsymbol{e}_h^T \boldsymbol{h}_{t-1} + \boldsymbol{e}_m^T \boldsymbol{m}_{t-1}, \tag{1}$$

$$\boldsymbol{m}_t = \boldsymbol{A}\boldsymbol{m}_{t-1} + \boldsymbol{B}u_t, \tag{2}$$

$$\boldsymbol{h}_t = f(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_m \boldsymbol{m}_t). \tag{3}$$

In equation (2), we see the LTI system defined by the $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{B} \in \mathbb{R}^{d \times 1}$ matrices, where d represents the order of the system. These matrices are usually frozen during training, although they need not be (see Voelker et al. (2019) for more details). Specifically, these matrices are defined by:

$$[\boldsymbol{A}]_{ij} = \frac{(2i+1)}{\theta} \begin{cases} -1 & i < j \\ (-i)^{i-j+1} & i > j \end{cases}, \tag{4}$$

$$[\boldsymbol{B}]_i = \frac{(2i+1)(-1)^i}{\theta}. \tag{5}$$

The input to the LTI system, $u(t)$, is computed by projecting the input to the RNN cell, $\boldsymbol{x}_t \in R^{d_x}$, the hidden state, $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$, and the memory state, $\boldsymbol{m}_t \in \mathbb{R}^d$, onto their respective encoding weights ($\boldsymbol{e}_x, \boldsymbol{e}_h,$ and $\boldsymbol{e}_m$). The memory state, hidden state and the input to the cell are then combined using the weight matrices ($\boldsymbol{W}_x, \boldsymbol{W}_h,$ and $\boldsymbol{W}_m$) and passed through the non-linearity, f. The encoding vectors and the kernel matrices are learned during training.

2.2 THE FF-LMU

In this paper, we propose a modified LMU architecture by making two specific changes to equations (1) and (3) defined above: first, we compute the input to the LTI system, $\boldsymbol{u}_t$, by implementing an affine transformation on the input to the RNN cell, $\boldsymbol{x}_t$, followed by an element wise nonlinearity; thereby, we do not project the input to a scalar before feeding it to the LTI system; and second, we discard the non-linear dynamical system defined in equation (3) and replace it with a feedforward output state $\boldsymbol{o}_t \in \mathbb{R}^{d_o}$. Making these changes, we obtain the following state update equations:

$$\boldsymbol{u}_t = g(\boldsymbol{W}_u \boldsymbol{x}_t + \boldsymbol{b}_u), \tag{6}$$

$$\boldsymbol{m}_t = \boldsymbol{A}\boldsymbol{m}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t, \tag{7}$$

$$\boldsymbol{o}_t = f(\boldsymbol{W}_o \boldsymbol{m}_t + \boldsymbol{b}_o). \tag{8}$$

It should be noted that the input and output states are equivalent to having a "time-distributed" dense layer before and after the layer implementing equation (7), respectively. We thus take ff-LMU to refer to equations (6)-(7), and for convenience, we define ff-LMU$_u$ to refer to equations (6)-(7), ff-LMU$_o$ to refer to equations (7)-(8), and ff-LMU$_m$ (m for minimal) to refer to equation (7).

Additionally, since the input to the LTI system in this case is a vector, $\boldsymbol{u} \in \mathbb{R}^{1 \times d_u}$, we have that $\boldsymbol{m}_t \in \mathbb{R}^{d \cdot d_u}$, and equation (7) can be thought of as implementing the following:

$$\boldsymbol{m}_t = \text{reshape}(\boldsymbol{A}\ \text{reshape}(\boldsymbol{m}_{t-1}, (d,\ d_u)) + \boldsymbol{B}\boldsymbol{u}_t,\ d \cdot d_u). \tag{9}$$

Table 1: Complexity per layer and minimum number of sequential operations of various architectures. n is the sequence length, d_x is the input dimension, and d is the order. First three rows are as reported in Vaswani et al. (2017).

Layer Type	Complexity per Layer	Sequential Operations
RNN	$O(n \cdot d_x^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d_x^2)$	$O(1)$
Attention	$O(n^2 \cdot d_x)$	$O(1)$
ff-LMU$_m$ (recurrent)	$O(n \cdot d^2 \cdot d_x)$	$O(n)$
ff-LMU$_m$ (convolution - frequency)	$O(n \cdot \ln n \cdot d + n \cdot d_x \cdot d)$	$O(1)$

The motivation for this new architecture is two-fold: 1) we believe that a model consisting of just the LTI system, which implements temporal compression, along with dense layers is sufficient to carry out a variety of tasks; this is demonstrated by the experiments in section (3); and 2) writing the LMU this way makes $\boldsymbol{m}_t$ a function of itself at a previous instant in time and $\boldsymbol{x}_t$ (through its dependence on $\boldsymbol{u}_t$) only, and since $(\boldsymbol{A}, \boldsymbol{B})$ are pre-defined and given access to $\boldsymbol{x}_t$, this architecture allows us to compute equation (7) in one shot using the convolution integral expressed in equation (10). That is, the state of a LTI system can be directly computed at any moment in time, without the need to simulate the dynamical system explicitly to know the future value of its state given the input.

$$\boldsymbol{m}(t) = \int_0^t \exp(\boldsymbol{A}(t-\tau))\boldsymbol{B}\boldsymbol{u}(\tau)\, d\tau. \tag{10}$$

Consequently, this reformulation allows us to express the RNN layer (equation (7)) as a feedforward layer. That is, instead of training an RNN where we feed the inputs sequentially, we can instead construct a feedforward network that corresponds to the RNN model, and thus make the training procedure completely feedforward, although with an additional memory cost. Furthermore, this formulation shows how feedforward training still allows us to do inference in a sequential manner (using equation (7) instead of (10)) for applications where the amount of available memory is limited, or where data is streamed in an online fashion.

Moreover, in special scenarios where the ff-LMU$_m$ layer is the first layer in a model – or when all the preceeding layers are frozen during training – it is possible to pre-compute the ouputs of this layer and train a network on this pre-computed dataset rather than the original dataset. This is similar to how pre-trained models with frozen weights are used to pre-compute the outputs in order to speed up training. We show in section (3.5) that doing so in the case of psMNIST results in training times that are 80x faster than that of the LSTM.

2.3 Complexity

Taking n, d_x, and d to represent the sequence length, input dimension and order, complexity per layer for various algorithms is presented in Table (1). We notice that the ff-LMU$_m$, like the Attention mechanism, depends more strongly on the length of the sequence, n, than on the dimension of the input, d_x. Thus, as is argued in Vaswani et al. (2017), ff-LMU$_m$ layer is faster than recurrent layers for applications where n is smaller than d_x, which is a common occurrence in NLP applications. For example, standard word or sub-word level neural machine translation systems work with sequences containing about 50 time steps and inputs that are 300 to 512 dimensional. More specifically, in our NLP experiments we found the setting $d = 1$ and $d_x \in \{100, 300\}$ to work well; configurations such as these, according to Table (1), are computationally advantageous for our model.

3 Experiments

In this section we consider the performance of the ff-LMU and LSTM models on sentiment classification, semantic similarity, natural language inference, and image classification. For all the ff-LMU models, we stick to a simple architecture involving dropout, and train all the models using the Adam optimizer (Kingma & Ba, 2014) with the default parameter settings. Thus, the order, number of units

Table 2: IMDB test accuracy. The first four rows are from Gu et al. (2020), where all the models are reported to use 256 hidden units.

Model	Accuracy
RNN	67.4 ± 7.7
LSTM	87.3 ± 0.4
LMU	87.7 ± 0.1
HiPPO-LagT	88.0 ± 0.2
LSTM (our impl.)	87.29
ff-LMU$_m$	**89.10**

in the output and inputs states, and occasionally additional dense units, are the only hyperparameters that we tune.

With LSTM models, wherever possible, to avoid the risk of misrepresentation, we use results found in published work, even when they make use of more sophisticated architectures, learning schedules, or regularization methods. Otherwise, we construct models with the same constraints as above.

3.1 Sentiment Classification

Task The IMDB dataset (Maas et al., 2011) is a standard sentiment classification task containing a collection of 50k highly polar reviews, with the training and testing sets containing 25k reviews each. We set aside 2.5k sentences from the training set to be used as the validation set. Given a review, the task is to classify it as expressing a positive or negative sentiment. Hence, this is a binary classification problem. We use the standard pre-processed dataset available from the Keras website,[1] consider a vocabulary of 20k words, and set the maximum sequence length to 500 words.

Models We evaluate LSTM and ff-LMU$_m$ model accuracies as a function of the number of parameters. We consider models ranging from about a 100 parameters to 100k parameters (excluding the embedding parameters). We set the dimension of the embedding layer to be 100. Both models make use of dropout with keep probability set to 0.5. We do not make use of any pre-trained word embeddings.

Results & Discussion We found that the LSTM (73 units) and ff-LMU$_m$ ($d = 1$ and $\theta = 500$) models achieve peak accuracies of 87.29% and 89.10% at 50k and 101 parameters, respectively. The ff-LMU$_m$ model thus outperforms the LSTM model, while using 500x fewer parameters. We also report scores from Gu et al. (2020) in Table (2). All of their implementations (reported in the first four rows) use 256 hidden units. We first note that our implementation of LSTM is comparable to theirs, although ours uses only 73 hidden units. We see that our ff-LMU model, despite using only 101 parameters, outperforms several complex models that are orders of magnitude larger. This prompts us to propose 89.10% as the new baseline for RNN models on the IMDB dataset.

3.2 Semantic Relatedness

We consider two datasets here: Quora Question Pairs[2] and Microsoft Research Paraphrase Corpus (Dolan & Brockett, 2005).

QQP For question and answer websites such as Quora and Stack Overflow, it is extremely useful to be able to automatically identify duplicate questions. In order to aid research in this direction, Quora released a dataset, called the Quora Question Pairs (QQP), where, given a pair of sentences, the task is to identify whether the two sentences are semantically similar. The labels are binary, with 0 indicating semantic similarity and 1 otherwise. In this case, we experiment on two train/dev/test splits: 390k/8k/8k like in Shen et al. (2018), which we refer to as split$_1$, and 280k/80k/40k like in

[1] https://keras.io/api/datasets/imdb/.

[2] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

Table 3: QQP test accuracies. Split_1 and split_2 LSTM scores are as reported in Shen et al. (2018) and Sharma et al. (2019), respectively. The number of parameters for split_2 (800k) is not reported but is a conservative estimate.

	Parameters		Accuracy	
Model	split_1	split_2	split_1	split_2
LSTM	-	800k	82.58	81.4
ff-LMU_m	1201	1201	**86.95**	**85.36**

Sharma et al. (2019), which we refer to as split_2. We use a vocabulary of 20k words and truncate the sentences to be less than 25 words.

MRPC Microsoft Research Paraphrase Corpus is a collection of sentence pairs extracted from online news sources, and similar to QQP, the task is to identify whether a given pair is semantically equivalent. We use the standard 3.7k/300/1.7k split, and set the sequence length to 50.

Models - QQP For QQP, we use scores reported in Shen et al. (2018) and Sharma et al. (2019). Although we are unaware of the particulars of the model used in the former case, we know the architecture of the model considered in the latter one; it uses an LSTM with 300 units to produce two sentence embeddings (one for each sentence). These two embeddings, their absolute difference, and their element wise product are then concatenated and passed onto a single dense layer, followed by a classification layer. They make use of dropout and L2 regularization, and also use 300D GloVe word embeddings (Pennington et al., 2014) which are fine tuned during training. Our model is similar to theirs, with three exceptions: 1) we use an ff-LMU_m ($d = 1$ and $\theta =$ maxlen) instead of an LSTM to obtain the sentence embeddings; 2) we do not use the dense layer, i.e., we pass the concatenated vector directly to a classification layer; and 3) we only make use of dropout (with keep probability of 0.5).

Models - MRPC For MRPC, we use the scores reported in Wang et al. (2018). Their BiLSTM models consist of a two-layer, 1500D LSTM (per direction) with max pooling and 300D GloVe word embeddings. They encode the sentences and pass a concatenated version (same as above) to a classifier, which is an MLP with a 512D hidden layer. Additionally, they augment their model using the ELMo (Peters et al., 2018) and CoVe (McCann et al., 2017) methods of pre-training. They do not report the number of parameters, but we estimate it to be well in excess of 20M. In contrast, we use the same simple model as above, except with an ff-LMU_o layer with $d = 1, \theta = 25$ and $d_o = 10$ (about 12k parameters). In this case too we use GloVe embeddings but we freeze them during training.

Results & Discussion - QQP Results of the experiment are reported in Table (3). We see that the ff-LMU_m outperforms the LSTM model while using significantly fewer parameters ($\sim$650x fewer). We believe this is a new state-of-the-art result for QQP for an RNN.[3] It is also interesting that the next best model[4] achieves an accuracy of 88.0%, but it is a large transformer model (Raffel et al., 2019) with around 60M parameters, or 50,000x more parameters than our model. It should be noted however that their model uses a different dataset split (364k training samples) and is evaluated on a private test set provided by the GLUE benchmark.

Results & Discussion - MRPC Results of this experiment are reported in Table (4). We see that the ff-LMU_o performs comparably to the CoVe BiLSTM model and does better than all the others reported in the table by a good margin. It is also of note that our model uses at least 1000x fewer parameters. Despite the initial success, given the size of the dataset, further progress on this task likely necessitates extensive pre-training or knowledge-transfer across tasks.

[3] See https://paperswithcode.com/sota/question-answering-on-quora-question-pairs.

[4] See https://paperswithcode.com/sota/question-answering-on-quora-question-pairs.

Table 4: Comparison of MRPC results. First six rows are from Wang et al. (2018). We report accuracy/F1 as in that work as well.

Model	MRPC
BiLSTM	69.3/79.4
+ELMo	69.0/80.8
+CoVe	73.4/81.4
+Attn	68.5/80.3
+Attn,ELMo	68.8/80.2
+Attn, CoVe	68.6/79.7
ff-LMU$_o$	**73.5/81.4**

Table 5: Comparison of results for the SNLI dataset. First row is from Bowman et al. (2015).

Model	Parameters	Accuracy
LSTM	220k	77.6
ff-LMU$_m$	3603	**78.85**

3.3 Natural Language Inference

Task The Stanford Natural Language Inference[5] was released to serve as a benchmark for evaluating machine learning systems on the task of natural language inference. Given a premise, the task is to determine whether a hypothesis is true (entailment), false (contradiction), or independent (neutral); it is thus a 3-way classification task. We use the standard 550k/10k/10k split, consider a vocabulary of 20k words, and set the maximum sequence length to 25 words.

Models We use the model described in Bowman et al. (2015) as the baseline LSTM model. Their model uses LSTMs with 100 units to produce two sentence embeddings (for premise and hypothesis) which are then concatenated and fed to a stack of three 200 dimensional dense layers. They use dropout, L2 regularization, 300D GloVe word embeddings and an additional dense layer to project the vectors to a lower dimensional space. Our model, in comparison, is very simple: we use ff-LMU ($d = 1$ and $\theta = 25$) to obtain sentence embeddings, which, along with their absolute difference and element wise product, are concatenated and passed to a softmax classification layer with a dropout layer (keep=0.5) in between. We also make use of the 300D GloVe embeddings.

Results & Discussion We report the results of this experiment in Table (5). We see that the simple ff-LMU model containing 3.6k parameters outperforms the relatively deeper LSTM model containing 220k parameters. In summary, our model improves accuracy while using 60x fewer parameters. To put this in the broader context, the next best model (Bowman et al., 2016) achieves a score of 80.6% (at 3M parameters or 830x more) and the current state-of-the-art (Pilault et al., 2020) achieves a score of 92.1% (at 340M parameters).

3.4 Image Classification

Task The permuted sequential MNIST (psMNIST) dataset was introduced by Le et al. (2015) to test the ability of RNNs to model complex long term dependencies. psMNIST, as the name suggests, is constructed by permuting and then flattening the (28×28) MNIST images. The permutation is chosen randomly and is fixed for the duration of the task, and in order to stay consistent, we use the same permutation as Chandar et al. (2019) and Voelker et al. (2019). The resulting dataset is of the form (samples, 784, 1), which is fed into a recurrent network sequentially, one pixel at a time. We use the standard 50k/10k/10k split. For an RNN to successfully classify such a sequence, it has to capture the essential features of the input that are presented over 784 time-steps, and use this temporal representation to infer the right class.

[5]Dataset and published results are available at https://nlp.stanford.edu/projects/snli/.

Table 6: Comparison of psMNIST results. All but last two rows as reported in Voelker et al. (2019). Second last row as reported in Gu et al. (2020).

Model	Accuracy
RNN-orth	89.26
RNN-id	86.13
LSTM	89.86
LSTM-chrono	88.43
GRU	92.39
JANET	91.94
SRU	92.49
GORU	87.00
NRU	95.38
Phased LSTM	89.61
LMU	97.15
HiPPO-LegS	98.3
ff-LMU$_o$	**98.49**

Models As was pointed out in Voelker et al. (2019), in order to facilitate fair comparison, RNN models being tested on this task should not have access to more the $28^2 = 784$ internal variables. Keeping that in mind, and in order to make direct comparisons to the original LMU model and the Non-saturating Recurrent Unit (Chandar et al., 2019), we consider an ff-LMU$_o$ model with $d = 468$ dimensions for memory. We set the dimension of the output state $d_o = 346$ and use $\theta = 784$. Our model uses 165k parameters, the same as all the models reported in Table (6), except for the original LMU model, which uses 102k parameters, and the HiPPO-LegS model, which is reported to use 512 hidden dimensions (unaware of the number of parameters).

Results & Discussion Test scores of various models on this dataset are reported in Table (6). The ff-LMU$_o$ model not only surpasses the LSTM model, but also beats the current state-of-the result of 98.3% set by HiPPO-LegS (Gu et al., 2020) this Fall. Thus, our model sets a new state-of-the art result for RNNs of 98.49% on psMNIST.

3.5 Training Time

Here we focus on the training times of the LMU, LSTM, and ff-LMU models. In order to ensure fairness, we choose the psMNIST benchmark to carry out this comparison as the three best performing models on this task had similar parameter counts. This dataset also allows us to demonstrate a feature of the ff-LMU that we alluded to in section (2.2): in special scenarios where ff-LMU$_m$ is the first layer in a model, we can save time by training on the pre-computed outputs of this layer instead of the original dataset – note that this is also possible in the case of MRPC, where the embedding weights are frozen during training. We see from Table (7) that using such a strategy reduces the time per epoch to just a second, and the corresponding total time to around 20s, where 15s refers to the time to pre-compute the "new" dataset and 5s refers to the actual training time. This is about 80x faster than the LSTM and 44x times faster than the LMU. Even without the pre-computation, ff-LMU exhibits faster training times, about 11x and 6x faster than the LSTM and LMU models, respectively.

4 Discussion

Based on the results on these five datasets, ff-LMUs provide parameter efficient and accurate models across a variety of problem domains. We demonstrated two new state-of-the-art results for RNNs on the standard RNN psMNIST benchmark, and the QQP dataset. It is interesting that the ff-LMU, despite being simpler than the original LMU, outperforms it on this dataset. This suggests that the main advantage of the LMU over past models is the quality of its temporal memory, implemented by the optimal LTI in the recurrent part of the model. For the latter dataset, the next best model on

Table 7: Training time comparison (in seconds). We use models containing 102k parameters (same as the original LMU model), set the batch size to 100, and use a single GTX 1080 to obtain the following numbers. By "Total Time" we refer to the time it takes for the *training* accuracy to cross 95%.

Model	s/epoch	Total Time(s)
LSTM	32	1600
LMU	220	880
ff-LMU$_o$	30	150
ff-LMU$_o$ (pre-comupted)	1	5 + 15

current leader boards is a transformer that uses 50,000x more parameters. Future work should determine if scaling up the ff-LMU can beat current transform models while preserving this parameter advantage.

On the other three datasets, we demonstrate a consistent advantage of the ff-LMU over LSTMs in terms of performance and parameter counts. While these are not hitting state-of-the-art accuracies, largely because both models are much smaller than networks that do, these experiments demonstrate significantly superior performance with 60-1000x fewer parameters, suggesting a general trend of parameter efficiency.

Finally, one of the main contributions of this approach is that it provides a means to train networks that are recurrent during inference in a feedforward manner. This provides a much better mapping of the training problem to currently commodity hardware. As well, it provides a practical inference advantage, as the final model can be run recurrently, thus providing a natural means of processing data online with minimal latency and efficient use of memory. Future work will focus on realizing this advantage for much larger problems, comparing similarly sized recurrent and transformer models.

5 Conclusion

With the intention of alleviating the long standing problem of speeding up RNN training, we introduce a variant of the LMU architecture, the ff-LMU, that can process data in a feedforward or sequential fashion. When implemented as a feedforward layer, the ff-LMU can utilize parallel processing hardware such as GPUs and thus is suitable for large scale applications, and when run as an RNN, it could be useful in applications where the amount of available memory is a limitation. We demonstrate the effectiveness of this architecture by experimenting on a broad range of tasks, of varying scale and difficulty, and showing that ff-LMU achieves superior performance on them, often requiring far fewer parameters. We also briefly consider the question of computational complexity of our model, and argue for its suitability to large scale applications in the domain of NLP, a direction we will pursue in the future. In sum, we believe that the ff-LMU affords a scalable and light weight solution to many problems without sacrificing accuracy.

References

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

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3280–3287, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. Hippo: Recurrent memory with optimal polynomial projections. *arXiv preprint arXiv:2008.07669*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466, 2018.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6294–6305, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

Jonathan Pilault, Amine Elhattami, and Christopher Pal. Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data. *arXiv preprint arXiv:2009.09139*, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. Natural language understanding with the quora question pairs dataset. *arXiv preprint arXiv:1907.01041*, 2019.

Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *arXiv preprint arXiv:1805.09843*, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 15570–15579, 2019.

Aaron R Voelker and Chris Eliasmith. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural computation*, 30(3):569–609, 2018.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.