SparseLang

MIE324 Project Final Report

prepared by

Qingyang Guo (1003216557)
Yongchao Zhou (1003282592)

University of Toronto, Engineering Science Division, Machine Intelligence

December 2, 2018

Word count: 1999

Goal and Motivation

The Recurrent Neural Network (RNN) [1] is widely used in many state-of-the-art models to solve a variety of problems. However, RNN models are sometimes hard to deploy due to the compute and storage requirement as well as the training time it takes. Research [2] has suggested that sparsity technique can be used to reduce compute and memory requirement of the neural network by pruning the redundant connections between neurons. Inspired by that, we hypothesized that the size of RNN models can be reduced by implementing sparse structure, and we explored our hypothesis in the context of a language modeling task. The goal of this project is to reduce the number of parameters of the model and hence computation load of training by employing sparsity in the connectivity of an RNN-based Language Model [3], while maintaining model's performance. Moreover, we also aimed to explore how different level of sparsity would affect the performance of the model.

Source of Data

The dataset we used is Penn Treebank dataset [4], which is a common benchmarking corpus. It contains 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation. The raw datasets were downloaded from Tomas Mikolov's webpage [5] and were pre-processed through data pipelines. After tonization and indexify, the dataset was chucked up into batches of examples and was ready to be fed into model. There was not data labeling required for this project, since the language model tries to predict the most likely word that will come up next in the sequence given the previous words. The labels were collected by delaying the input data one time step. An example was shown below:

Inputs : "I" "went" "to" "school" …
Labels: "went" "to" "school" …

We have pre-processed the data using two different methods, one way was to truncate the dataset into samples with fixed number of tokens, and the other method was to truncate the dataset at the end of each sentence, so that each sample has different number of tokens.

Overall Software Structure

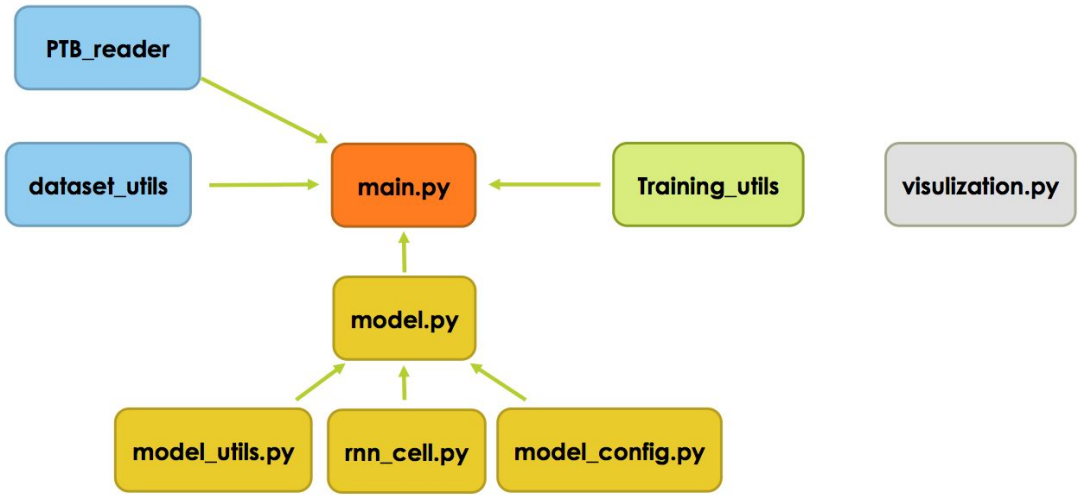


Figure 1 - Block diagram of the SparseLang's overall software structure

The overall structures of software consist of five submodules, which are model, dataset, main, training utils and visualization. The "model" module is the most crucial one, since it contains the implementation of the sparse LSTM cell as well as the definition of embedding layer and output layer. During the training, we used model_config.py file to specify the hyperparameters we used for the training. In terms of the "dataset" module, it contains data pipeline which was used to preprocess the dataset and make it ready to be fed into model. As for the "main" file, it contains the training and evaluation loop and logging information during the training. The "training utils" contains helper functions for training like learning rate warm up and decay scheme. Finally, we used "visualization" file to understand our training results better, for example, we visualized the values and the distribution of the weight matrices' parameters using histogram and heatmap respectively.

Baseline Model: Standard LSTM Model

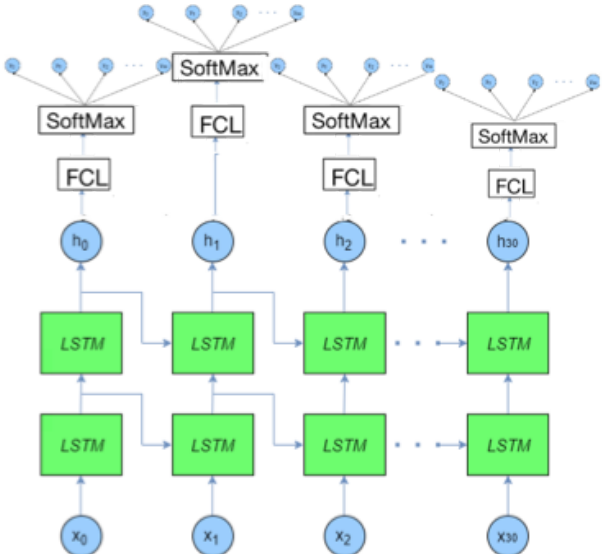
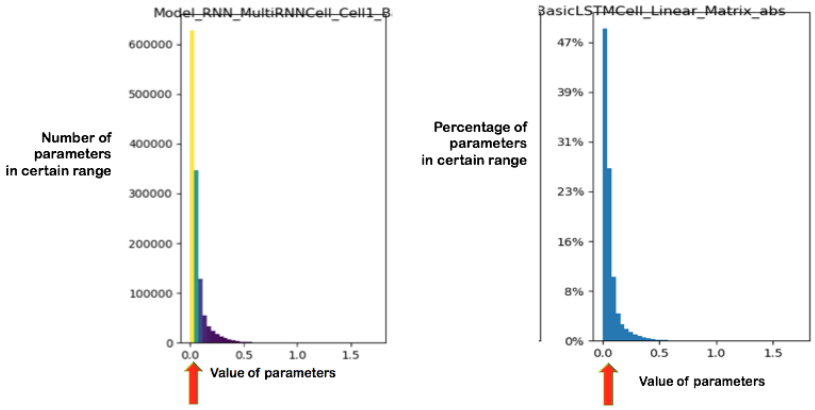


Figure 2 - Visual representation of baseline model structure

The standard multi-layer LSTM model was used as the baseline model [6]. The inputs to the model are batches of indexes. First of all, the inputs are converted to word vectors by a word embedding layer, which is trained to create meaningful words vectors. Then the word vectors are treated as the input to the first layer of the LSTM cell whose outputs are then fed into the second layer of the LSTM cell. The outputs from the second LSTM cell are flattened by a linear layer and then fed into the softmax classification layer, which computes the probability of every word in the vocabulary.

After training the baseline model, we visualized the values and the distribution of the parameters of weight matrices. As Figure 3 shown, a large portion of parameters (around 75%) has weights less than 0.07. This observation implies sparsity in the weight matrices, suggesting that sparse implementation is a promising direction to further pursue.



Observation: After training, large portion of parameters of baseline model are close to 0.

Figure 3 - Visualization of the values and distributions of the parameters of the baseline model.

Furthermore, we have trained the model on the two datasets that were pre-processed differently, as Figure 4 shown, the training results have no significant difference.

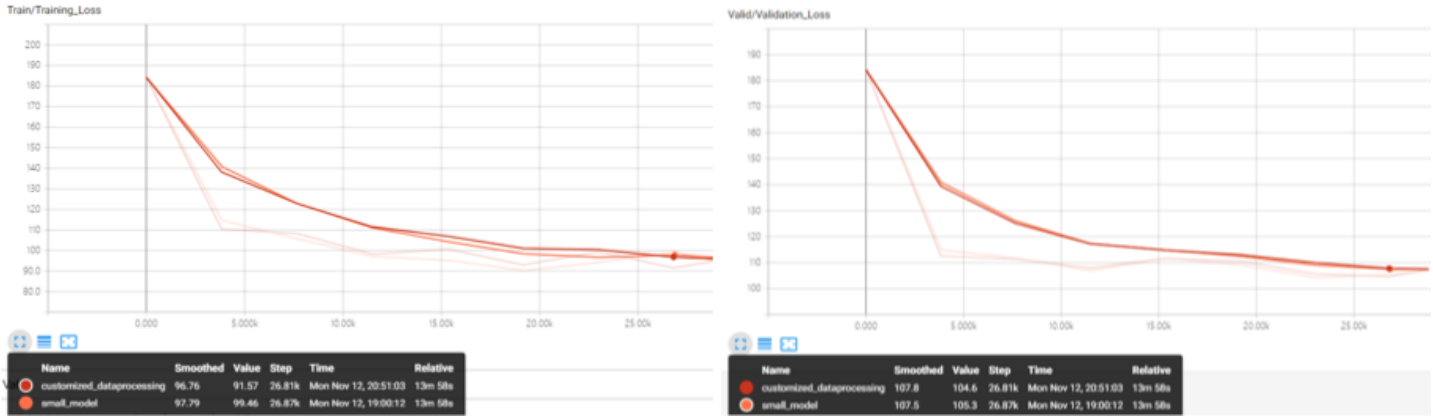


Figure 4 - The plots of the training loss(left) and validation loss(right)

Sparse LSTM Model

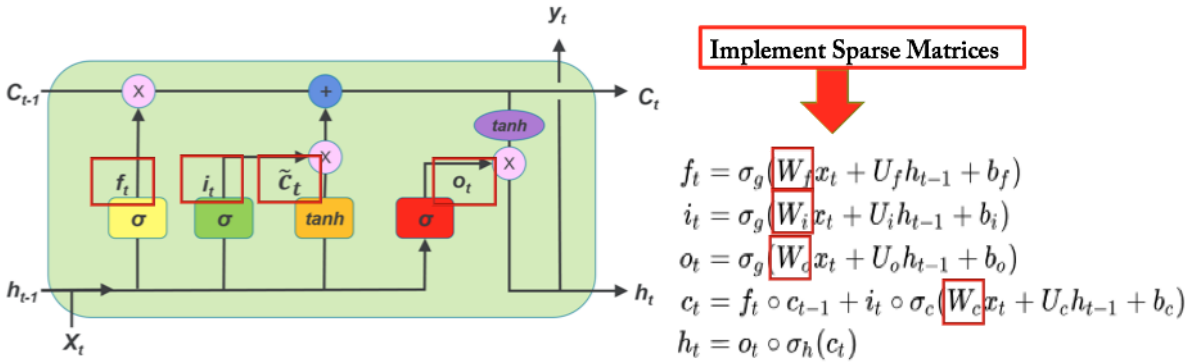


Figure 5 - Weight matrices in LSTM cell that are replaced by sparse matrices.

The Sparse LSMT model was developed based on the standard LSTM model. The sparse structure was implemented by replacing full-size weight matrices that were used to calculate the gates of the LSTM with sparse weight matrices. More specifically, to replace regular weight matrices, we first constructed sparse matrices by selecting some random indices from the full matrix and then initialized the selected parameters by uniform distribution initializer. (Note: these initialized parameters will be updated during training) These newly created sparse matrices were represented by a special tensor called SparseTensor, which only stores the non-zero elements' indices, values and the shape of the dense matrix. By utilizing SparseTensor, we could apply sparse matrix multiplication to all the weight matrices instead of normal matrix multiplication. According to TensorFlow official guide [7], the more sparse the matrix is, the more the sparse matrix multiplication outperforms regular matrix multiplication.
The reason we chose to use two layers LSTM cells is not only because it's a standard choice in language modeling task, but also it's based on the experiments we have done. We observed that

with the same number of parameters, model with two LSTM layers always outperformed the model with a signal layer. Meanwhile, two-layer model has similar performance with three-layer model but it becomes difficult to train the model when more and more layers were stacked up.

Training, Validation and Test Results

In order to explore how would the different level of sparsity (eg. 1%, 0.5%, 0.2%, 0.1%, 0.05%) affect the performance of the model, we have designed and conducted two controlled experiments. For the first experiment, we varied the sparsity level by fixing the size of the weight matrix while changing the number of non-zero parameters. For the second experiment, we fixed the number of non-zero parameters and changed the size of the weight matrix, therefore varying the level of sparsity. It is worth mentioning that the hidden size of the LSTM remained the same throughout the first experiment, whereas for the second experiment, the hidden size increased so that the weight matrix became more and more sparse.

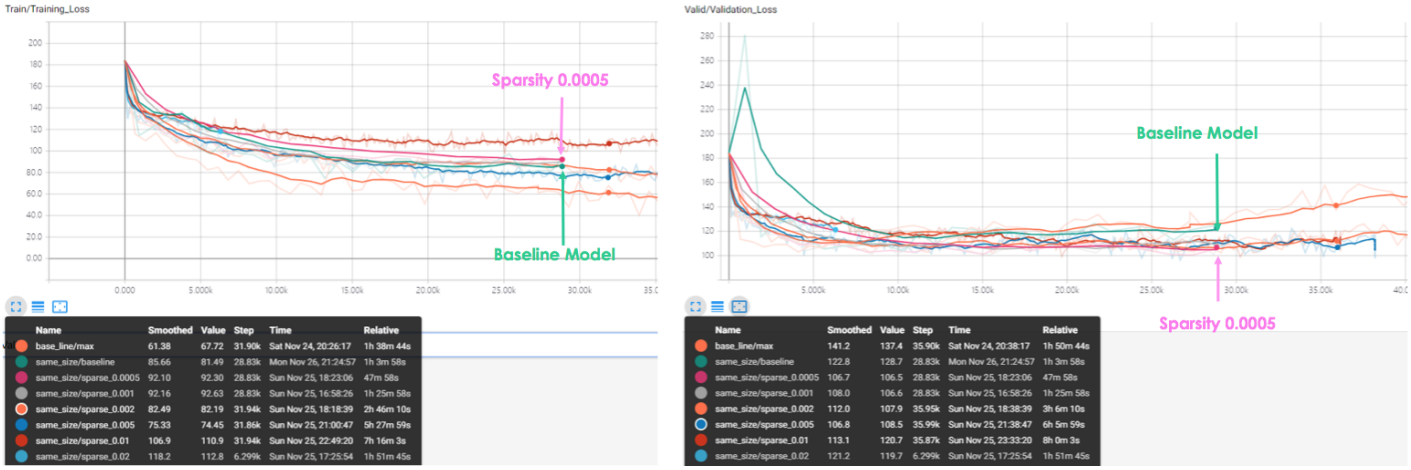


Figure 6 - The plots of training loss(left) and validation loss(right) of Experiment 1 have shown that sparse models have better performance than the baseline model

Spasity	# of Parameters
0.0005	50k
0.001	95k
0.002	180k
0.005	420k
0.01	810k
1	81020k
Large	107.9M

Table 1 - The number of parameters of the baseline model and different sparse models

The training and validation loss plots were shown in Figure 6. From the training loss plot, we observed no significant difference among sparse models with different sparsity levels and the baseline model. However, in the validation loss plot, all of the sparse models outperformed the baseline model. As the table above shown, the baseline model contains 80 million parameters, and the most sparse model (sparsity = 0.0005) contains only 50 thousand parameters, whereas

the sparse model had achieved better performance. The observation of the most sparse model outperformed the baseline model was very surprising and it was out of our expectation.

The results of the second experiment were similar to the first experiment but not as impressive, the train and validation loss plots can be found in the Appendix. All sparse models except the one with sparsity 0.01 performed better than the baseline model, but there is no obvious difference among all sparse models with different sparsity.

Although the training results seem to suggest that the sparse LSTM works pretty well, we were very cautious about making that statement, considering there are still many other factors may affect the performance of the model.

1) Regularization: Since we didn't apply any regularization techniques such as dropout, L2 regularization during the training, so it is possible that the baseline model was overfitted to the training set, resulting in a slightly worse performance in the validation set. But the sparse model seems to generalize well, so we hypothesized that the sparse structure has intrinsic regularization effect, but it needs to be further explored.
2) Embedding layer and output layer: Since our experiments focused on the LSTM, we did not pay much attention to the impact from embedding layer and output layer, maybe these two layers are so powerful that they capture most of the information needed to perform well on the language modeling task. In that case, LSTM cell only plays an insignificant role in the model, so whether it is strong or not does not have a huge impact on the performance of the model.
3) Dataset and Task: Maybe the dataset is not challenging enough, there is not enough information to learn, so it is too easy for the sparse model to perform well. Moreover, maybe the language modeling task is too "suitable" for the sparse model, and a different task may reveal the deficiency of the sparse model.
4) Initialization: It was observed that the sparse models with different sparsity have pretty similar performance, and there is no monotone relationship between sparsity and performance. We hypothesized that this might due to the random initialization of the weight matrix, resulting in some trials have a better "starting point" than the others.

Ethical Issues

Since in this project we have worked closely with the language models, it is necessary for us to consider the ethical issues related to its applications. One of the most widely used applications of language modeling is speech recognition [8],which can be applied in all kinds of scenarios. For example, speech recognition technology has allowed visually impaired and disabled community to interact with computers and phones. The technology is also used to assist professional interpreters during simultaneous interpretation. The list of useful applications of speech recognition technology can go on and on, however, we should also consider some potential risks associated with it. Before speech recognition technology was developed, many people were already concerned about government wiretapping their phone calls. As technology develops, many smart devices have voice-controlled features, people start to worry about the technology empowers government to easily surveille general public and some companies might take advantages of the audio records, creating customer profiles. Many technologies themselves are neutral, but how we use and develop them is not, so it is very important for machine learning engineers to be aware of the possible consequences and impact of their work could have on people and society.

Key Learnings

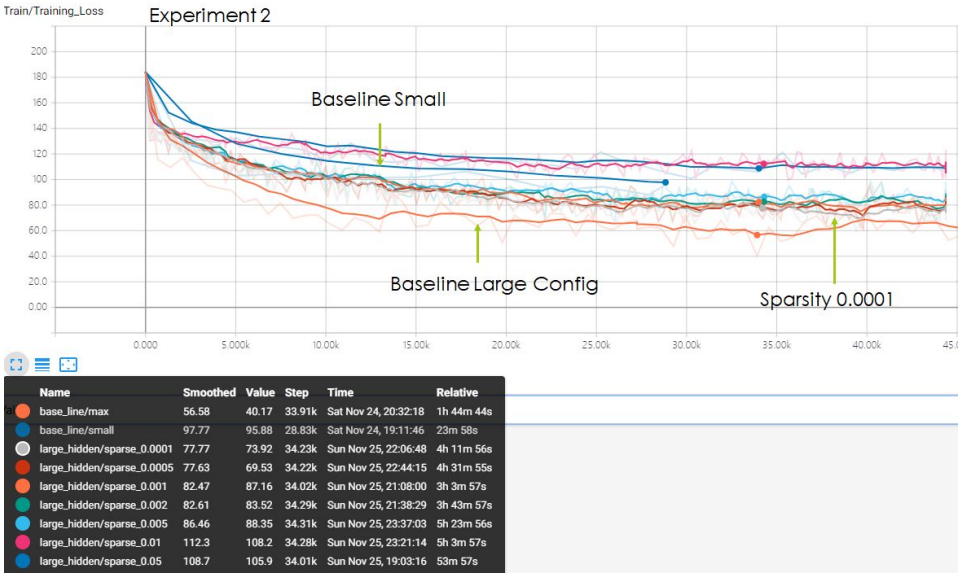
This project provided a valuable learning experience for both team members. Besides from learning all about the technical machine learning knowledge, we think the key learning from this project is how to properly and efficiently convey our ideas to others. For our proposal presentation, we failed to deliver our idea to the audience in a clear and understandable manner, it was mainly because we didn't efficiently link the technical details and the application together, and we didn't properly express the goal of our project. As Prof. Rose pointed out, lack of clarity implies of lack of thinking and understanding. After the presentation, we dug in more and tried to grasp a deeper understanding of the project. When we were preparing for the progress presentation, we were we cautiously designed the presentation, hoping it's more understandable. During the preparation process, we found that by presenting the project to people who are not familiar with it and asking them to repeat what the project is about turns out to be a very good method to verify whether or not the presentation makes sense. Learning how to effectively convey an idea to others is hard, but we are glad that we are aware of it and are making effort to improve on it.

Reference

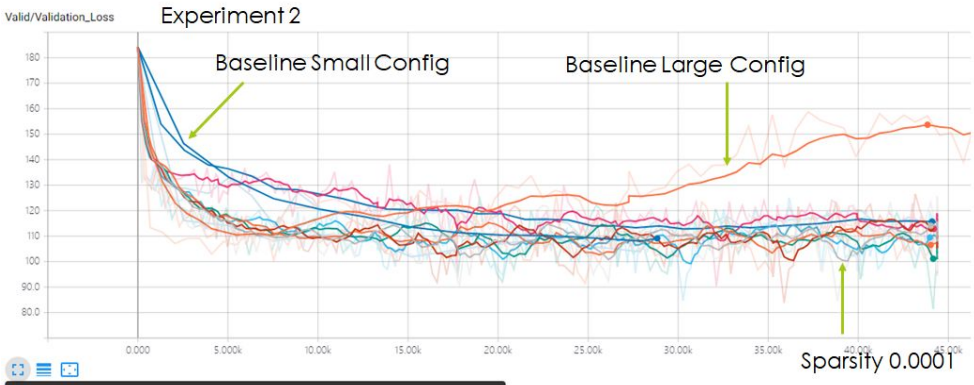
[1] J. L. Elman, “Finding structure in time,” Cognitive science, vol. 14, no. 2, pp. 179–211, 1990.
[2] M. Thom, Sparse Activity and Sparse Connectivity in Supervised Learning, 14th ed. Journal of Machine Learning Research, 2013, pp. 1091-1143 [Online]. Available: http://www.jmlr.org/papers/volume14/thom13a/thom13a.pdf. [Accessed: 03- Dec- 2018]
[3] T. Mikolov, M. Karafiat, L. Burget, J. Cernock ´ y, and S. Khudanpur, “Recurrent neural network based language model.” ` in Interspeech, vol. 2, 2010, p. 3
[4] Treebank-3. (n.d.). Retrieved from https://catalog.ldc.upenn.edu/ldc99t42
[5] Fit.vutbr.cz. [Online]. Available: http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz. [Accessed: 03- Dec- 2018]
[6] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997
[7] "tf.sparse.matmul | TensorFlow", TensorFlow, 2018. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/sparse/matmul. [Accessed: 03- Dec- 2018]
[8] A. Graves, A.-R. Mohamed, and G. Hinton, ‘‘Speech recognition with deep recurrent neural networks,’’ in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., May 2013, pp. 6645–6649

Appendix

Training results of Experiment 2



Name	Smoothed	Value	Step	Time	Relative
base_line/max	56.58	40.17	33.91k	Sat Nov 24, 20:32:18	1h 44m 44s
base_line/small	97.77	95.88	28.83k	Sat Nov 24, 19:11:46	23m 58s
large_hidden/sparse_0.0001	77.77	73.92	34.23k	Sun Nov 25, 22:06:48	4h 11m 56s
large_hidden/sparse_0.0005	77.63	69.53	34.22k	Sun Nov 25, 22:44:15	4h 31m 55s
large_hidden/sparse_0.001	82.47	87.16	34.02k	Sun Nov 25, 21:08:00	3h 3m 57s
large_hidden/sparse_0.002	82.61	83.52	34.29k	Sun Nov 25, 21:38:29	3h 43m 57s
large_hidden/sparse_0.005	86.46	88.35	34.31k	Sun Nov 25, 23:37:03	5h 23m 56s
large_hidden/sparse_0.01	112.3	108.2	34.28k	Sun Nov 25, 23:21:14	5h 3m 57s
large_hidden/sparse_0.05	108.7	105.9	34.01k	Sun Nov 25, 19:03:16	53m 57s



Name	Smoothed	Value	Step	Time	Relative
base_line/max	153.7	157.2	43.84k	Sat Nov 24, 21:02:18	2h 14m 44s
base_line/small	109.1	113.0	28.83k	Sat Nov 24, 19:11:46	23m 58s
large_hidden/sparse_0.0001	114.1	123.2	44.02k	Sun Nov 25, 23:18:47	5h 23m 56s
large_hidden/sparse_0.0005	112.9	103.0	44.04k	Mon Nov 26, 00:00:15	5h 47m 56s
large_hidden/sparse_0.001	106.6	105.2	44.02k	Sun Nov 25, 22:02:00	3h 57m 57s
large_hidden/sparse_0.002	101.1	81.45	44.16k	Sun Nov 25, 22:40:29	4h 45m 57s
large_hidden/sparse_0.005	109.4	119.6	43.98k	Mon Nov 26, 01:07:03	6h 53m 56s
large_hidden/sparse_0.01	112.9	114.6	44.19k	Mon Nov 26, 00:49:14	6h 31m 57s
large_hidden/sparse_0.05	115.7	114.5	44.10k	Sun Nov 25, 19:19:16	1h 9m 57s

Spasity	# of Parameters
0.0001	548k
0.0005	453k
0.001	313k
0.002	345k
0.005	420k
0.01	358k
0.05	147k
1	2880k
Large	107.9M