Machine Learning - Final Project

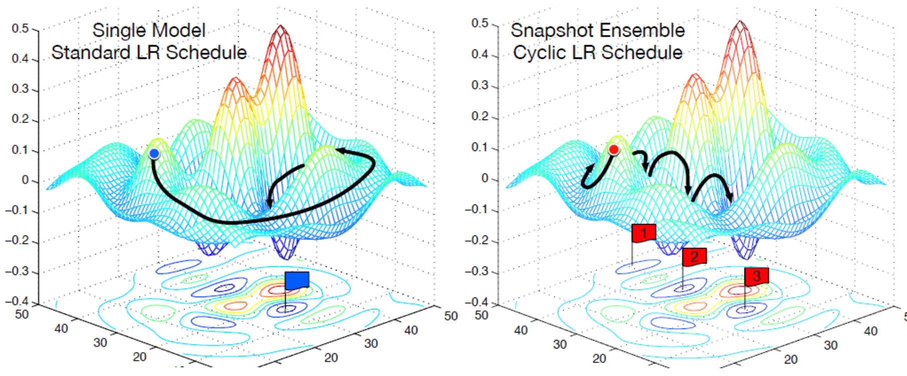
Maor Sagi 205502636

Introduction

In this assignment we requested to evaluate the performance of Deep Learning ensemble methods that are beyond the scope of the course. I will describe an algorithm that uses a few local minimum points models as different models, combining all to one decision in an efficient way with a single neural network.

The paper: Huang, Gao, et al. "Snapshot ensembles: Train 1, get m for free." *arXiv preprint arXiv:1704.00109* (2017).

Although preventing the model from reaching local minimum points is considered positive, this paper argues with this approach and claims local minimum points can be useful. The authors elaborate about the advantages of different networks with different parameters initialization convergence points, but they claim local minima often give similar results.



This illustration is taken from the original paper, on the left you can see the model converges to one final minimum point as we all know. On the right you can see the checkpoints of the number of local minimum points like the approach presented in the paper. They call it "*Snapshot Ensembling*" because they save a snapshot at each local minima they reach during the training process.

Method

Deep learning ensembles are mostly computationally expensive. Neural networks are naturally resource consuming and running different networks is costly. The approach the paper presents reduces training cost by training a single model and saves different checkpoints of local minimum convergences as different models.

Instead of training M networks independently, the authors let the single network SGD converge M times to local minimum points. For each convergence point they save the model parameters as a model for the final ensemble model. When the SGD got close to the minimum point the learning rate reduced, and after saving the weights increased again to escape the current local minimum point.

The paper approach is inspired by the idea of Loshchilov et al., dropping the learning rate in an early stage, rather than hundreds of epochs, has a minor effect on the final test error. The method Loshchilov et al. proposed for converging to multiple local minimum points, called *cyclic annealing schedule*. The first learning rate decay starts after 50 epochs, and repeats several times. Formally, the learning rate is: $\alpha(t) = f\left(\mathrm{mod}\left(t-1, \lceil T/M \rceil\right)\right)$.
t - the iteration number.
T - the total number of iterations.
M - the total number of minimum local points required (number of models in the ensemble).
f - monotonically decreasing function.
After convergence and model weights saved, the learning rate reset to the initial large value (defined as $\alpha = f(0)$).
f function defined to be the same as proposed by Loshchilov et al.:

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mathrm{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right)$$

α_0 - the initial learning rate.
Note that the update of the learning rate depends on the iteration number and not by epoch number because it seems to improve the convergence in short cycles even when a large learning rate is used.

As described before, for each local minima, model weights are saved. After M models are saved the training process is over.

At test time, all the *M* models from the snapshot are being tested and the result is the averaged of all their predictions. the number of models to be ensembled *m* , can be any number while *m*<=*M*. The final prediction is: $h_{\text{Ensemble}} = \frac{1}{m} \sum_0^{m-1} h_{M-i}(\mathbf{x})$.

$h_i(\mathbf{x})$ - the softmax output value of test sample *x* , of model *i*.

The last *m* models would always be the ones to be average because they tend to have lower test error.

Data and Preprocessing

The datasets have been chosen after running Initial experiments to recognize interesting patterns and determining various properties of the datasets. I focused on binary classification tasks because I expect to find interesting patterns I could not see while running multi-class classification tasks as well. The properties examined are:

- The number of categorical features, if there are any.
- Size of dataset - large/small.
- Number of features - large/small.
- Features Ranges - similar features scale, and top-bottom of a feature range similar or not (e.g 1-1000, and 1-10,000 ranges are different scales, 1-1000 and 1000-2000 are not normalized to the same axis).

The 20 datasets I used in my experiments are:

1. analcatdata_boxing1
2. blood
3. bodyfat
4. breast-cancer
5. chatfield_4
6. cloud
7. diabetes
8. diggle_table_a2
9. kidney
10. meta
11. no2
12. pima
13. plasma_retinol
14. pm10

15. prnn_synth
16. socmob
17. statlog-australian-credit
18. statlog-heart_
19. veteran
20. visualizing_livestock

Preprocessing:

I converted all binary classes to [0,1] classes (instead of [P,N] for example). I used Label Encoder for the categorical strings features to change all string labels to integers. In addition, I drop ID columns and drop all rows with null values and drop “cancor2”, ”fract2” which have a high missing ratio in the “meta” dataframe.

Here you can see detailed information of the datasets after preprocessing, while small numbers mean under the average and vice versa.

Dataset Name	# Rows	# Features	# Categorical Features	Columns Ranges	Columns Data Types
Analcatdata Boxing	120 (small)	4 (small)	3 (large)	['0-9', '0-1', '1-12']	['object', 'object', 'int64', 'int64']
Blood	748 (large)	5 (small)	0 (small)	['-1.17-7.97', '-0.77-7.62', '-0.77-7.62', '-1.32-2.61']	['float64', 'float64', 'float64', 'float64', 'int64']
Bodyfat	252 (small)	15 (large)	1 (small)	['0.99-1.11', '22-81', '118.5-363.15', '29.5-77.75', '31.1-51.2', '79.3-136.2', '69.4-148.1', '85.0-147.7', '47.2-87.3', '33.0-49.1', '19.1-33.9', '24.8-45.0', '21.0-34.9', '15.8-21.4']	['float64', 'int64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Breast Cancer	286 (small)	10 (large)	0 (small)	['-2.63-2.31', '-2.74-0.91', '-2.32-2.43', '-0.46-6.5', '-3.53-0.51', '-1.42-1.29', '-0.94-1.06', '-1.79-2.36', '-1.79-0.56']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Cloud	108 (small)	7 (small)	1 (small)	['0-1', '0.0-6.0', '0.06-6.93', '0.0-9.42', '0.02-4.02', '0.04-7.84']	['object', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Chatfield	235 (small)	13 (large)	1 (small)	['1749-1983', '0.0-217.4', '0.0-182.3', '0.0-190.7', '0.0-196.0', '0.0-238.9', '0.0-200.7', '0.0-191.4', '0.0-200.2', '0.0-235.8', '0.0-253.8', '0.0-210.9']	['int64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Diabetes	768 (large)	9 (small)	6 (large)	['0-17', '0-199', '0-122', '0-99', '0-846', '0.0-67.1', '0.08-2.42', '21-81']	['int64', 'int64', 'int64', 'int64', 'int64', 'float64', 'float64', 'int64', 'int64']

Diggle	310 (small)	9 (small)	5 (large)	['76-84', '2-51', '33-442', '257-499', '306-791', '1.08-1.6', '5.55-6.21', '5.72-6.67']	['int64', 'int64', 'int64', 'int64', 'int64', 'float64', 'float64', 'float64', 'int64']
Kidney	76 (small)	6 (small)	5 (large)	['2-562', '0-1', '10-69', '0-1', '0-3']	['int64', 'int64', 'int64', 'object', 'object', 'int64']
Visualizing Livestock	130 (small)	3 (small)	2 (small)	['0-4', '0-25']	['object', 'object', 'int64']
Veteran	137 (small)	8 (small)	7 (large)	['1-2', '1-4', '0-1', '10-99', '1-87', '34-81', '0-10']	['int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64']
Statlog Heart	270 (small)	14 (large)	0 (small)	['-2.79-2.48', '-1.45-0.69', '-2.29-0.87', '-2.09-3.84', '-2.39-6.08', '-0.42-2.39', '-1.02-0.98', '-3.4-2.26', '-0.7-1.42', '-0.92-4.5', '-0.95-2.3', '-0.71-2.47', '-0.87-1.19']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Statlog Australian Credit	690 (large)	15 (large)	0 (small)	['-1.5-4.11', '-0.96-4.67', '-1.78-2.87', '-1.73-1.8', '-1.85-2.16', '-0.66-7.85', '-1.05-0.95', '-0.86-1.16', '-0.49-13.28', '-0.92-1.09', '-3.11-3.58', '-1.07-10.55', '-0.2-19.0', '-0.89-1.12']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
Socmob	1156 (large)	6 (small)	4 (large)	['0-16', '0-16', '0-1', '0-1', '0.0-746.3']	['object', 'object', 'object', 'object', 'float64', 'int64']
Prnn Synth	250 (small)	3 (small)	0 (small)	['-1.25-0.86', '-0.19-1.09']	['float64', 'float64', 'int64']
PM10	500 (large)	8 (small)	1 (small)	['0.69-5.39', '3.81-8.35', '-19.0-21.9', '0.3-9.9', '-5.0-4.0', '5.0-358.0', '1-24']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64', 'int64']
Plasma Retinol	315 (small)	14 (large)	7 (large)	['19-83', '0-1', '0-2', '16.33-50.4', '0-2', '445.2-6662.2', '14.4-235.9', '3.1-36.8', '0.0-203.0', '37.7-900.7', '214-9642', '30-6901', '0-1415']	['int64', 'object', 'object', 'float64', 'object', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64', 'int64', 'int64', 'int64']
Meta	504 (large)	20 (large)	8 (large)	['0-20', '270-20000', '270-58000', '6-180', '2-26', '0-43', '0-1', '1.03-4.0', '0.05-0.75', '0.5-0.99', '0.15-1.0', '0.18-6.1', '0.99-160.31', '0.29-4.7', '0.81-6.55', '0.02-1.31', '1.56-160.64', '1.76-159.64', '0-23']	['object', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'object', 'int64']
NO2	500 (large)	8 (small)	1 (small)	['1.22-6.4', '4.13-8.35', '-18.6-21.1', '0.3-9.9', '-5.4-4.3', '2.0-359.0', '1-24']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64', 'int64']
Pima	768 (large)	9 (small)	0 (small)	['-1.14-3.9', '-3.78-2.44', '-3.57-2.73', '-1.29-4.92', '-0.69-6.65', '-4.06-4.45', '-1.19-5.88', '-1.04-4.06']	['float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'float64', 'int64']
AVERAGE:	406.15	9.3	2.6		

** Python notebook of profiling and analyzing data submitted with the report.*

Architecture

The original experiments from the paper research were executed with images datasets. I thought it would be interesting to test the method with different data - structured tabular data.

For this purpose, I have not used the implementation of the authors - https://github.com/gaohuang/SnapshotEnsemble.
I created a network aligned with tabular data, based on Keras official code examples for structured data neural networks for classification tasks (code github and other sources present in the next section). The main important thing to pay attention to is that our network should deal with categorical features in addition to numerical features. The solution was to convert integer inputs to one-hot encoded integers.

the structure of the neural network is simple and depends on the input size (number of features):

1. Preprocessing Layers - IntegerLookup layer for categorical features , Normalization to numerical features.
2. Concatenate Layer - concatenate all input layers into one input.
3. Dense Layer (size = 2.5* input_size)
4. Dropout Layer (rate 0.2)
5. Dense Layer (size = 1, binary classification)

The loss function is binary cross entropy and the model compiled with an SGD optimizer.

Experiments

The algorithm I chose to compare with is Random Forest. I chose a classic Machine Learning model to see if a complex neural network provides better performance than a simple Random Forest ensemble. This model consists of a number of decision trees trained on sub-samples of the data. The ensemble method to combine the decision of all trees is averaging.

Experiments Flow:

```
datasets = datasets from the classification_datasets given to us for this project.
results = empty dataframe
1.  for dataset in datasets:
2.      ds = load_and_preprocessing(dataset)
3.      for train_set ,test_set in 10-fold cross validation(ds):
#           Snapshot algorithm
4.          best_params = random_search(3-cross validation(train_set))
5.          build_and_train(best_params ,train_set)
6.          models = load_models(dataset )
7.          result = predict(models ,test_set)
8.          results.append(result)
#           Random Forest
9.          best_params = random_search(3-cross validation(train_set))
10.         rf = build_and_train(best_params ,train_set)
11.         result = predict(rf, test_set)
13.         results.append(result)
14.     statistical_tests(results)
```

For the training I used 10-fold cross validation using stratified K-fold, then I ran hyper parameter tuning on each fold, and after I got the best parameters for this fold I built, trained and tested the model.

I ran hyperparameters tuning using Random Search monitored by accuracy measurement, I ran it with 50 iterations, for each iteration I estimated the parameters with 3-fold cross validation train and test loop.
The Ensemble Snapshot constant parameters are batch size 32 and a learning rate of 0.01.
The parameters I chose to tune are the main parameters that impact the model training process the most -

1. The number of cycles - 15, 20. This parameter together with the next one, are the ones that determine the number of epochs the model trained in total.
2. Epochs cycle - 20, 30, 50, 70, 100. The total number of epochs is epochs cycles times the number of cycles.
3. The number of models - 3,5,7,10,12. The number of chosen models from the last checkpoints.

The cycles' configuration would determine the whole training process. The last parameter has tradeoffs - a high number of models would generalize the model but also can decrease the accuracy. We select the last models, and as long as we increase the number of the models, we consider early models to impact the final decision, early models means model from the start of the training process.

The hyperparameters of the Random Forest I chose to tune, are the the ones determine the model structure and as an outcome have large influence on the training process:

1. N estimators - 2, 5, 10, 50, 100. The number of models.
2. Max depth - None, 10, 20, 50, 70. The maximum depth of the trees.
3. Max features - “sqrt”, “log2”. The branching factor method.

For the statistical tests I used Mann Whitney U test, for each dataset I tested whether the accuracy performance differences are statically significant. For all datasets I test whether Snapshot Ensemble metrics are better than Random Forest and vice versa.

** Github link for the source code of the paper: https://github.com/MaorSagi/Snapshot-Ensamble-Network*
** Random Forest used - sklearn.ensemble.RandomForestClassifier (LINK)*
** Additional Sources :*
- Keras Blog - Structured data classification from scratch
- Kaggle tutorials - Snapshot ensemble tutorial with keras

Evaluation Results and Discussion

In this section I present to you the statistical tests results for each dataset. I only describe tests that rejected the null hypothesis. The Less column informs us the direction of the one sided test (while positive, Random Forest value is higher).

Dataset Name	Metric	p-value	Reject H0	Less
Analcatdata Boxing	TPR	0.001	1	1
Analcatdata Boxing	FPR	0.034	1	1
Analcatdata Boxing	Precision	0	1	1
Analcatdata Boxing	Training Time	0	1	0
Analcatdata Boxing	Inference Time	0	1	0
Blood	Accuracy	0.033	1	0
Blood	TPR	0.048	1	1

Blood	FPR	0.003	1	1
Blood	AUC	0	1	0
Blood	AUPRC	0	1	0
Blood	Training Time	0	1	0
Blood	Inference Time	0	1	0
Bodyfat	Accuracy	0	1	1
Bodyfat	TPR	0	1	1
Bodyfat	Precision	0	1	1
Bodyfat	AUC	0	1	1
Bodyfat	AUPRC	0	1	1
Bodyfat	Training Time	0	1	0
Bodyfat	Inference Time	0	1	0
Breast Cancer	Training Time	0	1	0
Breast Cancer	Inference Time	0	1	0
Chatfield	AUC	0.005	1	0
Chatfield	AUPRC	0.019	1	0
Chatfield	Training Time	0	1	0
Chatfield	Inference Time	0	1	0
Cloud	Accuracy	0	1	0
Cloud	FPR	0	1	1
Cloud	AUPRC	0.006	1	0
Cloud	Training Time	0	1	0
Cloud	Inference Time	0	1	0
Diabetes	Accuracy	0	1	1
Diabetes	TPR	0	1	1
Diabetes	FPR	0	1	1
Diabetes	Precision	0	1	1
Diabetes	AUC	0	1	1
Diabetes	AUPRC	0.007	1	1

Diabetes	Training Time	0	1	0
Diabetes	Inference Time	0	1	0
Diggle	Accuracy	0.028	1	1
Diggle	TPR	0.039	1	1
Diggle	Training Time	0	1	0
Diggle	Inference Time	0	1	0
Kidney	Accuracy	0	1	1
Kidney	FPR	0	1	0
Kidney	Precision	0.002	1	1
Kidney	AUC	0.001	1	1
Kidney	AUPRC	0	1	1
Kidney	Training Time	0	1	0
Kidney	Inference Time	0	1	0
Meta	Accuracy	0.003	1	0
Meta	TPR	0.008	1	0
Meta	FPR	0.023	1	0
Meta	AUC	0.019	1	0
Meta	Training Time	0	1	0
Meta	Inference Time	0	1	0
NO2	Accuracy	0	1	1
NO2	TPR	0.048	1	1
NO2	Precision	0.013	1	1
NO2	AUC	0	1	1
NO2	Training Time	0	1	0
NO2	Inference Time	0	1	0
Pima	AUC	0	1	0
Pima	Training Time	0	1	0
Pima	Inference Time	0	1	0
Plasma Retinol	TPR	0	1	0

Plasma Retinol	FPR	0	1	0
Plasma Retinol	Precision	0.043	1	1
Plasma Retinol	AUC	0.032	1	1
Plasma Retinol	Training Time	0	1	0
Plasma Retinol	Inference Time	0	1	0
PM10	Accuracy	0.001	1	1
PM10	TPR	0.004	1	0
PM10	FPR	0	1	0
PM10	Precision	0.001	1	1
PM10	AUC	0.001	1	1
PM10	AUPRC	0.009	1	1
PM10	Training Time	0	1	0
PM10	Inference Time	0	1	0
Prnn Synth	AUC	0.009	1	0
Prnn Synth	Training Time	0	1	0
Prnn Synth	Inference Time	0	1	0
Socmob	AUC	0	1	0
Socmob	AUPRC	0	1	0
Socmob	Training Time	0	1	0
Socmob	Inference Time	0	1	0
Statlog Australian Credit	TPR	0.004	1	0
Statlog Australian Credit	FPR	0	1	0
Statlog Australian Credit	Precision	0.006	1	1
Statlog Australian Credit	AUPRC	0	1	1
Statlog Australian Credit	Training Time	0	1	0
Statlog Australian Credit	Inference Time	0	1	0
Statlog Heart	AUC	0.022	1	0
Statlog Heart	Training Time	0	1	0
Statlog Heart	Inference Time	0	1	0

Veteran	TPR	0	1	0
Veteran	FPR	0.001	1	0
Veteran	Training Time	0	1	0
Veteran	Inference Time	0	1	0
Visualizing Livestock	TPR	0.023	1	0
Visualizing Livestock	AUC	0.004	1	0
Visualizing Livestock	AUPRC	0.043	1	0
Visualizing Livestock	Training Time	0	1	0
Visualizing Livestock	Inference Time	0	1	0

** The detailed results csv files submitted with the report.*

As you can see the Snapshot Ensemble (SE) training and inference time (for 1000 samples) is significantly higher than the Random Forest (RF) algorithm. It seems in all datasets the time is lower for the RF, and we might generalize and conclude that the time would probably be lower for every dataset given. A detailed discussion of each dataset result presented below.

<u>Analcatdata Boxing:</u>
The RF algorithm acts better in the precision and the TPR, but we can see that the FPR was significantly higher, a drawback of RF.

<u>Blood:</u>
The SE performed better in this dataset, The accuracy was significantly higher and the AUC, AUPRC as well. The TPR of RF is indeed higher but also the FPR.

<u>Bodyfat:</u>
In this dataset RF performance was significantly better than SE except FPR. I might conclude that it is because the difference in the features' ranges (0-1 in one feature and 10-300 in another) might make the network harder to learn compared to RF decision trees.

<u>Chatfield:</u>
SE have higher scores in AUC and AUPRC, it might be because the features ranges excluding one are pretty similar and networks behave better in these conditions.

Cloud:
In this dataset the accuracy and AUPRC are better in the SE algorithm. Might be because the ranges of features are pretty close in top and bottom values.

Diabetes:
Like the Bodyfat dataset, RF performs better in many metrics (all of them). The ranges are various, it may be the reason for the poor performance of SE.

Diggle:
Higher accuracy and TPR to RF. Various features' ranges.

Kidney:
SE failed in FPR. in addition, RF succeeded in Precision AUC and AUPRC. Very different ranges of features.

Meta:
SE has higher scores in accuracy, TPR and AUC. It was also hard for it to distinguish positive samples, high FPR.

NO2:
RF performed better in accuracy, precision, TPR and AUC. The reason might be the different ranges.

Pima:
SE AUC score is higher.

Plasma Retinol:
SE TPR and FPR score is higher, it is hard for it to distinguish positive and negative. RF performance is better in Precision and AUC.
PM10:
SE TPR and FPR score is higher, it is hard for it to distinguish positive and negative. Precision, AUC and AUPRC are better for RF.

Prnn Synth:
AUC better in SE.

Socmob:
AUC and AUPRC are better in SE.

Statlog Australian Credit:
SE TPR and FPR score is higher, it is hard for it to distinguish positive and negative. Precision and AUPRC are better for RF.

Statlog Heart:
AUC better in SE.

Veteran:
SE TPR and FPR score is higher, it is hard for it to distinguish positive and negative.

Visualizing Livestock:
SE TPR, AUC and AUPRC scores higher significantly.

Conclusions

It is not surprising the snapshot algorithm is time consuming. That is something we can expect from a neuron network and should consider the tradeoff between other metrics important to us like accuracy, FPR etc., depending on the dataset we have.

It seems like in many datasets TPR was significantly higher, also FPR was higher. It might indicate that SE has difficulty distinguishing between positive and negative samples in many cases. Also, it might be concluded that classification tasks of datasets with various features range harder for SE than for RF.

For some datasets RF performance was significantly higher than SE. This is something we should keep in mind while choosing a model for structured data. It would be less time consuming and much simpler. Some tasks are perfect for Neural Network but sometimes the use of it is overkill.

Lets not forget the drawbacks of RF, because RF can’t handle strings categorical features, part of the preprocessing of the data, was changing the strings to encoded integers. Neural networks preprocessing layers include String Lookup Layers that can encode strings. In addition, CNNs compared to Random Forests are time and memory consuming but with large amounts of data would probably get lower error rates in complex tasks.

When our task is image classification we might want to use neural networks ensembles. In this case we should consider the benefits and drawbacks of Snapshot-Ensemble.

It does provide similar results to other deep learning models and it is less time and memory consuming during training, naturally because training a single network instead of number of models. In addition, we should pay attention to the simplicity of the algorithm in contrast to the complexity of running different networks, which might involve different architectures.

But, although it provides similar error rates, it is important to note it is not exactly the same as running different networks with different initialization or maybe different architecture. In this approach we are limited to one architecture in contrast to running a number of networks independently.