Evaluating the Robustness of Generative Teaching Networks

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, Jeff Clune

Kurt Willis

July 4, 2021

TU Berlin

Overview

1. Generative Teaching Networks

2. Robustness Performance Tests

3. Conclusion

Generative Teaching Networks

Meta-Learning

Meta-Learning is about ”learning to learn”.

There are generally 3 components to Machine-Learning..
The **environment**, the learner **model** and the learning **algorithm**.

Generative Teaching Networks

Generative Teaching Networks (GTN) aim to learn an environment and algorithm parameters via meta-learning for rapid training of a variety of different kinds of deep neural network architectures.

Generative Teaching Networks (GTN) aim to learn an environment and algorithm parameters via meta-learning for rapid training of a variety of different kinds of deep neural network architectures.

traditional ML
environment: *fixed*
algorithm: *fixed*
model architecture: *fixed*
model parameters: *learned*

Generative Teaching Networks

Generative Teaching Networks (GTN) aim to learn an environment and algorithm parameters via meta-learning for rapid training of a variety of different kinds of deep neural network architectures.

traditional ML	Generative Teaching Networks
environment: *`fixed`*	environment: *`learned`*
algorithm: *`fixed`*	algorithm: *`learned`*
model architecture: *`fixed`*	model architecture: *`random`*
model parameters: *`learned`*	model parameters: *`learned`*

GTN Algorithm

Algorithm Generative Teaching Networks

initialize $G_{\boldsymbol{\omega}}$
for $\mathtt{2000\ times}$ **do**
  sample $D_{\boldsymbol{\theta}}$
  for $\mathtt{64\ times}$ **do**
    $\mathsf{z} = (\mathsf{z}_x, \mathsf{z}_y) \sim$ sample randomly
    $\mathsf{x} \leftarrow G_{\boldsymbol{\omega}}(\mathsf{z})$
    $\hat{\mathsf{y}} \leftarrow D_{\boldsymbol{\theta}}(\mathsf{x})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} \mathcal{L}(\hat{\mathsf{y}}, \mathsf{z}_y)$
  end for
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \gamma \nabla_{\boldsymbol{\omega}} \mathcal{L}(D_{\boldsymbol{\theta}}(\mathsf{x}_{train}), \mathsf{y}_{train})$
end for

Meta-Loop

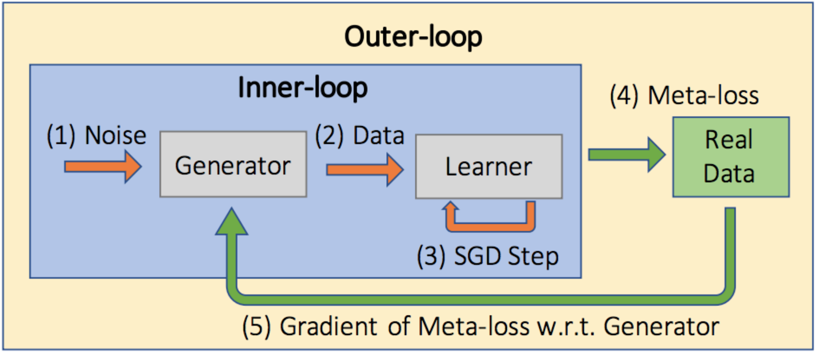


Figure 1: Generative Teaching Networks meta-loop [1]

Full Curriculum:
In addition to the weights of the generator G, the full ordered set of 64 latent vectors $\mathbf{z}^{(i)} \in \mathbb{R}^{138}$ and learning rates $\lambda^{(i)}$ (and weight decay parameters) are learned for $i \in [1, \ldots, 64]$.

GTN - Results

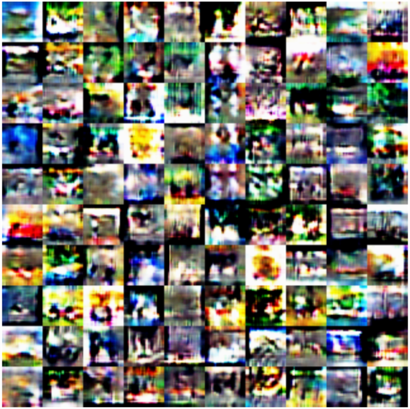


Figure 2: CIFAR10 Generator examples [1]

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)
- pooling layer doesn’t seem to work
- CIFAR10 performance too low

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however …

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work
- CIFAR10 performance too low
- attempt to recreate inner training loop failed

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work
- CIFAR10 performance too low
- attempt to recreate inner training loop failed
- custom autograd (required for gradient checkpointing)

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however …

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work
- CIFAR10 performance too low
- attempt to recreate inner training loop failed
- custom autograd (required for gradient checkpointing)
- custom convolution operation, only works on GPU

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work
- CIFAR10 performance too low
- attempt to recreate inner training loop failed
- custom autograd (required for gradient checkpointing)
- custom convolution operation, only works on GPU
- extremely convoluted, interdependent code (functions inside of functions inside of functions)

Experience With Codebase

Hope: simply train a GTN, then use that to train custom models, however ...

- limited architecture choices (weight normalization necessary)
- pooling layer doesn't seem to work
- CIFAR10 performance too low
- attempt to recreate inner training loop failed
- custom autograd (required for gradient checkpointing)
- custom convolution operation, only works on GPU
- extremely convoluted, interdependent code (functions inside of functions inside of functions)

$\rightarrow$ debugging & interacting with code was very difficult.

Training Models

GTN trained on *'base_larger'*.

learner type
'base'
'base_fc'
'base_larger'
'base_larger2'
'base_larger3'
'base_larger3_global_pooling'
'base_larger4'
'base_larger4_global_pooling'
'linear'

Training Models

GTN trained on `'base_larger'`.

Criteria: mean accuracy over 5 independently trained networks should be ≥ 0.7.

learning algorithm / *learner type*	`10_gtn`	`vanilla`	`gtn`
`'base'`	0.16	0.95	0.22
`'base_fc'`	0.97	0.98	0.98
`'base_larger'`	0.97	0.98	0.98
`'base_larger2'`	0.12	0.96	0.16
`'base_larger3'`	0.72	0.97	0.91
`'base_larger3_global_pooling'`	0.11	0.96	0.12
`'base_larger4'`	0.09	0.11	0.09
`'base_larger4_global_pooling'`	0.10	0.11	0.10
`'linear'`	0.62	0.96	0.61

Training Models

GTN trained on `'base_larger'`.

Criteria: mean accuracy over 5 independently trained networks should be ≥ 0.7.

learning algorithm / *learner type*	`10_gtn`	`vanilla`	`gtn`
`'base'`	0.16	0.95	0.22
`'base_fc'`	0.97	0.98	0.98
`'base_larger'`	0.97	0.98	0.98
`'base_larger2'`	0.12	0.96	0.16
`'base_larger3'`	0.72	0.97	0.91
`'base_larger3_global_pooling'`	0.11	0.96	0.12
~~`'base_larger4'`~~	0.09	0.11	0.09
~~`'base_larger4_global_pooling'`~~	0.10	0.11	0.10
`'linear'`	0.62	0.96	0.61

Training Models

GTN trained on *'base_larger'*.

Criteria: mean accuracy over 5 independently trained networks should be ≥ 0.7.

learning algorithm / *learner type*	*10_gtn*	*vanilla*	*gtn*
'base'	0.16	0.95	0.22
~~*'base_fc'*~~	0.97	0.98	0.98
'base_larger'	0.97	0.98	0.98
'base_larger2'	0.12	0.96	0.16
'base_larger3'	0.72	0.97	0.91
'base_larger3_global_pooling'	0.11	0.96	0.12
~~*'base_larger4'*~~	0.09	0.11	0.09
~~*'base_larger4_global_pooling'*~~	0.10	0.11	0.10
'linear'	0.62	0.96	0.61

Training Models

GTN trained on `'base_larger'`.

Criteria: mean accuracy over 5 independently trained networks should be ≥ 0.7.

learning algorithm / *learner type*	`10_gtn`	`vanilla`	`gtn`
~~`'base'`~~	0.16	0.95	0.22
~~`'base_fc'`~~	0.97	0.98	0.98
`'base_larger'`	0.97	0.98	0.98
~~`'base_larger2'`~~	0.12	0.96	0.16
`'base_larger3'`	0.72	0.97	0.91
~~`'base_larger3_global_pooling'`~~	0.11	0.96	0.12
~~`'base_larger4'`~~	0.09	0.11	0.09
~~`'base_larger4_global_pooling'`~~	0.10	0.11	0.10
~~`'linear'`~~	0.62	0.96	0.61

Robustness Performance Tests

Robustness Performance Tests

Overview of robustness tests:

- Noise corruption
- Blurring
- FGSM-attack
- LBFGS-attack

Robustness Performance Tests

For all experiments, two GTN models (based on *'base_larger'*, *'base_larger3'*) have been trained. For each GTN model and for each learner type, 5 independent models have been trained for the full 64 cycles (*gtn*). The same has been done for only 10 cycles for comparison (*10_gtn*). Also, each learner type has been trained for one epoch ($\sim$390 update steps) on the training set (*vanilla*). All tests are performed on unseen test data.

Noise Corruption

$$\tilde{x} = \text{clip}[\,x + \beta z\,]\,, \quad z \sim \mathcal{N}(0, I)$$

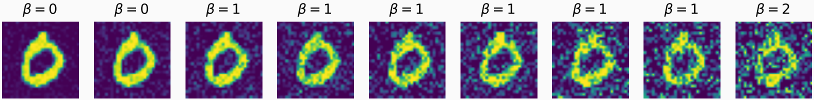
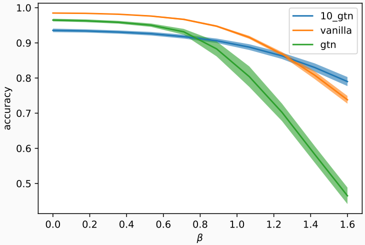


Figure 3: Noise corruption of varying strength β.

Noise Corruption - Results

GTN trained with *'base_larger'*



GTN trained with *'base_larger3'*

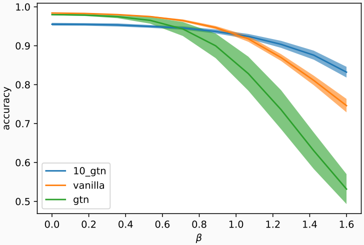


Figure 4: Accuracies of trained models under varying input noise corruption.

Blurring

$$\tilde{x} = \text{gaussian\_blur}_{\beta}(x)$$

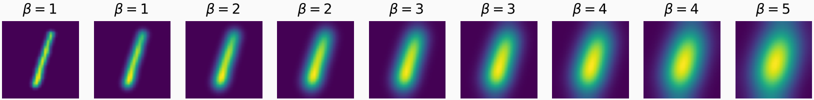


Figure 5: Gaussian-blur filter applied with varying Gaussian kernel std β.

Blurring - Results

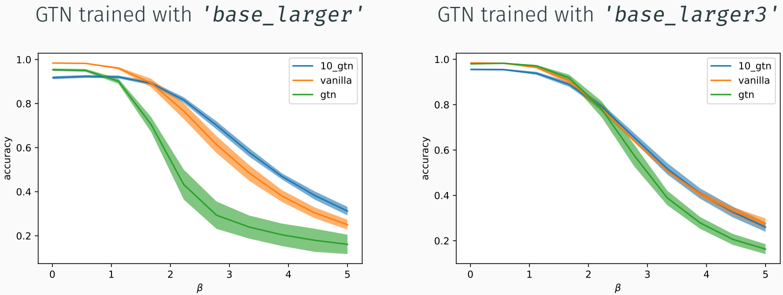


Figure 6: Accuracies of trained models under varying Gaussian-blur filters.

FGSM-Attack

$$\mathsf{x} \leftarrow \mathsf{x} + \beta\, \mathrm{sign}\left(\nabla_{\mathsf{x}} \mathcal{L}(D(\mathsf{x}), y)\right)$$
$$\mathsf{x} \leftarrow \mathrm{clip}[\,\mathsf{x}\,]$$

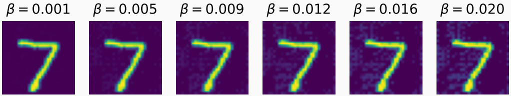


Figure 7: Results after 30 steps of FGSM-attack with varying β.

FGSM-Attack - Results

GTN trained with *'base_larger'* GTN trained with *'base_larger3'*

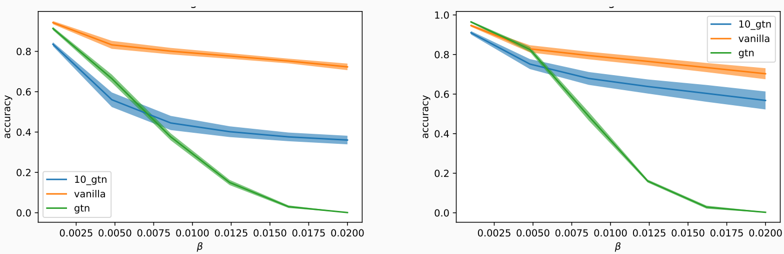


Figure 8: Accuracies of trained models under FGSM-attack with varying strengths.

LBFGS-Attack

$$\mathbf{x} \leftarrow \mathbf{x} + \varepsilon \left(\nabla_{\mathbf{x}} \mathcal{L}(D(\mathbf{x}), y) - \frac{1}{\beta} \nabla_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}^0\|_2 \right)$$

$$\mathbf{x} \leftarrow \mathrm{clip}[\,\mathbf{x}\,]$$

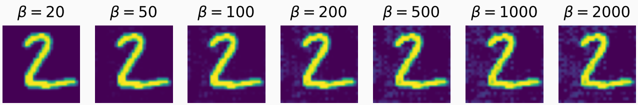
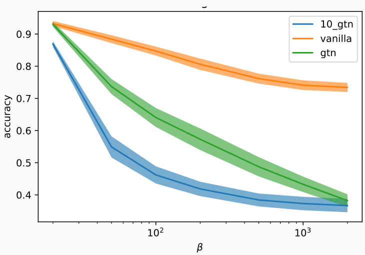


Figure 9: Results after 30 steps of LBFGS-attack with varying β.

LBFGS-Attack - Results

GTN trained with `'base_larger'`   GTN trained with `'base_larger3'`



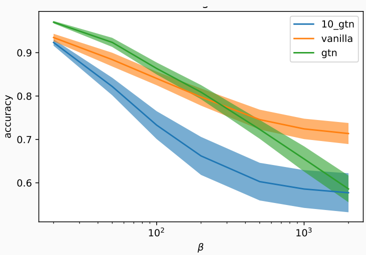


Figure 10: Accuracies of trained models under LBFGS-attack with varying strengths.

Conclusion

Conclusion

- models trained with fewer GTN steps display (mostly) higher robustness

Conclusion

- models trained with fewer GTN steps display (mostly) higher robustness
- models trained with GTNs are less robust than vanilla trained models

Conclusion

- models trained with fewer GTN steps display (mostly) higher robustness
- models trained with GTNs are less robust than vanilla trained models
- however, number of testable architectures was limited due to shortcomings

Conclusion

- models trained with fewer GTN steps display (mostly) higher robustness
- models trained with GTNs are less robust than vanilla trained models
- however, number of testable architectures was limited due to shortcomings
- MNIST is a fairly simple task $\rightarrow$ more datasets are needed to be conclusive

Conclusion

- models trained with fewer GTN steps display (mostly) higher robustness
- models trained with GTNs are less robust than vanilla trained models
- however, number of testable architectures was limited due to shortcomings
- MNIST is a fairly simple task $\rightarrow$ more datasets are needed to be conclusive
- GTNs are likely to improve over time

Questions?

References

Felipe Petroski Such et al. *Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data*. 2019. arXiv: 1912.07768 [cs.LG].