

SEMI SUPERVISED LEARNING

131129.301 2977 27.05860990326578 0.0305 0.0307
131173.792 2978 27.051805288605614 0.0303 0.0297
131218.21 2979 27.055281491702193 0.0302 0.031
131262.62 2980 27.042746063167538 0.0301 0.0307
131306.95 2981 27.047667698690837 0.03 0.03
131351.26 2982 27.044890637429503 0.0312 0.0306
131395.62 2983 27.04937428217574 0.0306 0.03
131440.00 2984 27.02953340458087 0.0304 0.0303
131484.25 2985 27.066224279740208 0.0298 0.0299
131528.77 2986 27.0458703190473 0.0302 0.0293
131573.23 2987 27.038823049895218 0.0305 0.0302
131617.47 2988 27.056954942357788 0.0303 0.0296
131661.86 2989 27.03262982465121 0.0287 0.0298
131706.20 2990 27.022691164653565 0.0298 0.0301
131750.48 2991 27.038230920911133 0.0296 0.0298
131794.83 2992 27.065048913319455 0.0303 0.0302
131839.24 2993 27.049289813677387 0.0306 0.0305
131883.71 2994 27.038168365813572 0.0305 0.0305
131928.01 2995 27.051252114398626 0.0305 0.0305
131972.63 2996 27.0475399018 0.0305 0.0305
132016.89 2997 27.035111111111111 0.0305 0.0305
132061.21 2998 27.035111111111111 0.0305 0.0305
132105.47 2999 27.035111111111111 0.0305 0.0305

BETTER COMPATIBILITY

With Python 3 becoming the new standard we decided that it would be best to port the existing code to this newer version.

Bringing with it a bunch of benefits:

- Improved Legibility
- Better support in the future

With this we will also try to improve:

- Overall Code Quality

REPRODUCING RESULTS

We have reproduced the Semi Supervised Learning results from the paper, and we have gotten a pretty substantial improvement on the result of the paper.

From 3.33 ± 0.14 to $2.95 \pm ??$. There was no mention on how the Stdev was calculated, and there is no time to do a lot of runs. The Stdev will be updated once we have done more runs over the weekend.

This improvement can have various reasons and is something that we are discussing in our blog.

PORTING TO PYTHON 3

BETTER COMPATIBILITY


With Python 3 becoming the new standard we decided that it would be best to port the existing code to this newer version.

Bringing with it a bunch of benefits:

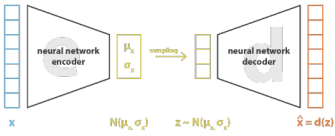
- Improved Legibility
- Better support in the future

With this we will also try to improve:

- Overall Code Quality




2021



$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_z, \sigma_z), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_z, \sigma_z), N(0, I)]$$

SEMI SUPERVISED

AVERAGE RUNTIMES



In order to test this program we have to run the M1+M2 test since that is the only test that is semi supervised. This test takes us nearly 48 hours to completely finish with 3000 epochs.

We did notice that after 2500 epochs the error did not decrease a lot. With a "good" result already showing up at around 2000 epochs.

SANDER VAN LEEUWEN
TOM SAVEUR
[HTTPS://GITHUB.COM/SANDERVANL/DEEP-SEMI-SUPERVISED-LEARNING](https://github.com/SandervanL/deep-semi-supervised-learning)

Reproduction of Semi-Supervised Learning with Deep Generative Models

<https://github.com/SandervanL/deep-semi-supervised-learning>

By: Sander van Leeuwen & Tom Saveur

Introduction

We decided to reproduce [Semi-supervised Learning with Deep Generative Models by Kingma et al.](#) The paper discusses two models, namely M1 and M2, and the combination of both. During our reproduction we decided to focus on the M1+M2 model, since this is the only model that is Semi Supervised according to the existing Code Repository. This paper mainly uses these models to evaluate the performance on a dataset called [MNIST](#), which is a dataset containing around 60.000 handwritten digits. M1+M2 will then try to evaluate each of these numbers to the correct class.



An Example of the MNIST dataset

In this blog we will try to reproduce the results of the paper, after which we ported the existing (outdated) codebase from Python 2 to Python 3, making it more approachable for users. And we will end up with a test on what happens when you change the hyperparameters of the program.

After running this program on our own computers we tried to obtain our free google cloud credits, but unfortunately something went wrong resulting in us not getting these credits, and having to think of another way to test this program, since running it on our machines alone would take too long to get any valuable results. We ended up getting 100\$ for free on [Microsoft Azure](#) from the [Github Edu Pack](#), which we used to reproduce 16 results in total. But you can read more about this in our section on Changing Hyperparameters.

Setting up initial project

As the paper focuses heavily on semi-supervised learning, approximate inference and variational autoencoders, it was difficult for us to get up to speed. We tried implementing our own version, but very soon noticed that this may cost hundreds of hours. Because of this we switched to trying to

reproduce the project with the code provided by the paper¹. While we agree this is not an independent reproduction and as such there might be a systematic error in our findings, this was the only option in the time budget set for this project.

Although Arxiv points out there are 12 other code implementations, we wanted to stay as close to the paper as possible and tried to reproduce the code of the authors. As the code was written in 2014, getting it to run was not an easy task, because of the following reasons:

- As this project uses Theano (a predecessor of PyTorch and TensorFlow), we had to rely on Anaconda to set up our environment. We had no prior experience with this framework.
- Package- and external dependencies were not given. We had to figure out on which packages and which versions of those packages the code depended. Also the ffmpeg.exe was a dependency, which was not stated.
- The project was written in Python 2.7, when Python 3 was not widely adopted yet. This caused some confusion at the start.
- No information was given on which environment variables to set. An environment variable of ML_DATA_PATH had to be set to the 'models' directory of the project.
- There existed compile-time and run-time bugs in the code, which had to be fixed for the code to run.

Porting to Python 3

To make this project future proof and easier to work with, we decided to port the project to Python 3. This port ended up not being as straightforward as we thought, a lot of the code relied on older Python tricks, and packages that are no longer well supported. And while this was not the hardest thing to do, we did end up having to spend quite some time on debugging and checking whether every error was removed from the code. Furthermore we had to run tests to ensure we had not introduced a systematic error in the training algorithm so that the Python 3 version would output different values than the Python 2 version.

After porting it to Python 3 we decided to try and tackle some of the code smells and style issues within the code. We tried our best in updating the code style a bit, but unfortunately we did not end up having enough time to fix all of the problems with code style, since there were over 1000 PEP8 errors, and a lot more warnings if you also factored in typo's, duplicated code, and other code smells.

Deployment

To test the results of the ported version of the code compared with the original version, we ran the code on our own computer. Because of this, it quickly became evident that we would not be able to run the tests on our own computer. Running one full test of 3000 epochs would take around 48 hours.

Because of the steep learning curve we were later than the other groups with starting our project's tests, and by this time the Google Coupon Code for running tests on the Google Cloud was not available anymore. After searching around, we were able to obtain \$100,- free Azure credit. As Azure

¹ <https://github.com/dpkingma/nips14-ssl>

limits the number of virtual cores to six per region, the most powerful machines we could hire were the Standard_DS3_v2 VM's, with four cores and 14 GiB of RAM. These machines were not as fast as our laptops, but we could scale horizontally very well with this new setup.

For both of us, this was a new environment. Because of this we wanted to automate testing our solutions, with a master PC automatically firing up servers all around the world to take full advantage of the limitation of six cores per region. This automatic setup could then in the future be used for further testing. This code is provided in the *runtests* folder. Using this code, another number of servers and different testing setups can easily be created.

Reproducing Results

The results from the paper can be found in the table below. Since the paper does not go into too much detail on what each of these numbers mean we had to follow our own interpretation, in consultation with Prof. Loog. In our interpretation N is equal to the number of labels used for training, where the total size of the training set is 50000. We tried to reproduce the numbers in the right-most column, which state the test error in percentages, with their 95% confidence bounds.

Table 1: Benchmark results of semi-supervised classification on MNIST with few labels.

| N | NN | CNN | TSVM | CAE | MTC | AtlasRBF | M1+TSVM | M2 | M1+M2 |
|------|-------|-------|-------|-------|-------|---------------------|----------------------|----------------------|----------------------------|
| 100 | 25.81 | 22.98 | 16.81 | 13.47 | 12.03 | 8.10 (± 0.95) | 11.82 (± 0.25) | 11.97 (± 1.71) | 3.33 (± 0.14) |
| 600 | 11.44 | 7.68 | 6.16 | 6.3 | 5.13 | – | 5.72 (± 0.049) | 4.94 (± 0.13) | 2.59 (± 0.05) |
| 1000 | 10.7 | 6.45 | 5.38 | 4.77 | 3.64 | 3.68 (± 0.12) | 4.24 (± 0.07) | 3.60 (± 0.56) | 2.40 (± 0.02) |
| 3000 | 6.04 | 3.35 | 3.45 | 3.22 | 2.57 | – | 3.49 (± 0.04) | 3.92 (± 0.63) | 2.18 (± 0.04) |

After careful consideration we decided to only reproduce the values for M1+M2. In the code no semi-supervised implementation was given to test M1 and M2 on their own. After researching the possibility to develop this ourselves, we came to the conclusion that this would be outside of the scope, as the most questionable (and therefore subject to reproducibility) would be the results of the joint M1+M2 models.

After running this once on our own computers we realized that it took too long to test the project reliably on time on our own PC's. The first and only full test we executed on our own PC's took about 48 hours for a test setup of $N=100$. This test led to an improvement of 0.39 percent-point to a test error of 2.94.

We try to explain this improvement in three ways:

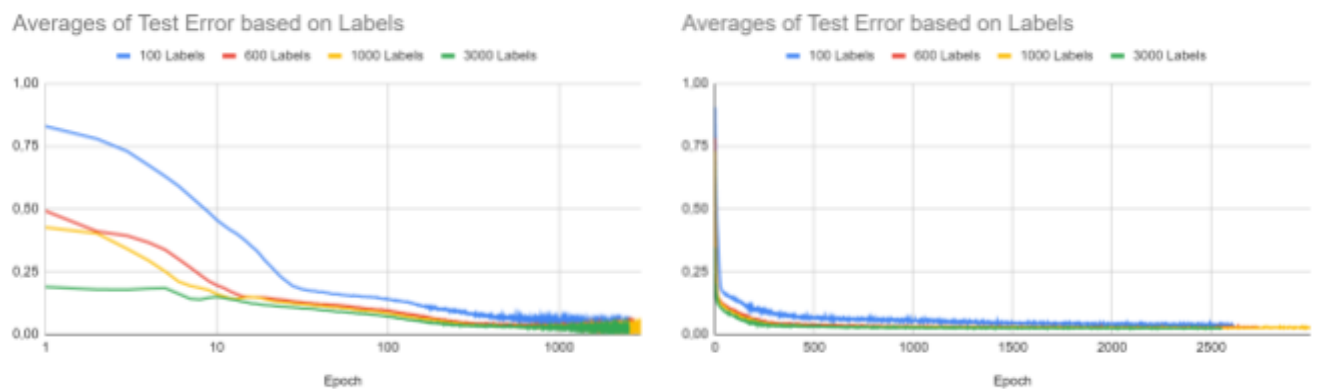
1. This could be a random deviation from the result stated in the paper, as this was still with the original Python 2.7 version of the code. However, this seems unlikely as the result is far from the confidence bound. (If the 0.14 means a confidence bound).
2. We might have run it for longer than the authors of the paper did. The paper did not state for how many epochs their models were trained. From our results we could see that we hit the 3.33 mark at around 2000 epochs. The code states that the results are only marginally improving after 1500 epochs. We ran for 3000 epochs, as implemented by the code.
3. Given that batching was used, the updating of the neuron weights would be stochastic. The program provided a way to give a random seed. While we believe changing the seed may not necessarily result in a massive performance boost, it is something to keep in mind. We will talk more about the seed in Changing Hyperparameters.

We reproduced the result for each N twice, once with 1000 as seed and once with 2000 as seed. This led to the following results.

| N | Test Error Seed 1000 At Epoch 2500 | Test Error Seed 2000 At Epoch 2500 | Average Test Error |
|------|------------------------------------|------------------------------------|--------------------|
| 100 | 4,52 | 3,15 | 3,84 |
| 600 | 2,8 | 2,92 | 2,86 |
| 1000 | 2,78 | 2,51 | 2,65 |
| 3000 | 2,39 | 2,39 | 2,39 |

Table with reproduction results after 2500 epochs with the average test error.

Graph relating (cost of) number of labels to the gained accuracy



For all sizes of N, the results that we obtained after 2500 epochs were higher than what the results that the author got in their paper. However, as these values are in the same order of magnitude, this paper may be considered reproduced.²

This means that, while we didn't think so before, the authors of the paper did run more than 2500 epochs. Since we did not have enough time to run 3000 epochs for all we can't say if that would've given us the same results as the authors of the paper. But what we can see from the results is that the seed might have a big influence on lower values of N. As you can see the test error is drastically different for N=100 when looking at seed 1000 and seed 2000.

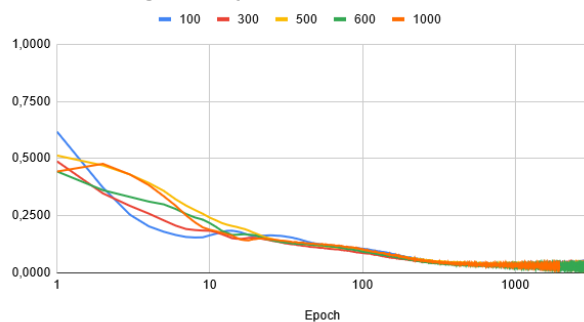
² More information: Raff, E. (2019): A Step Toward Quantifying Independently Reproducible Machine Learning Research

Changing Hyperparameters

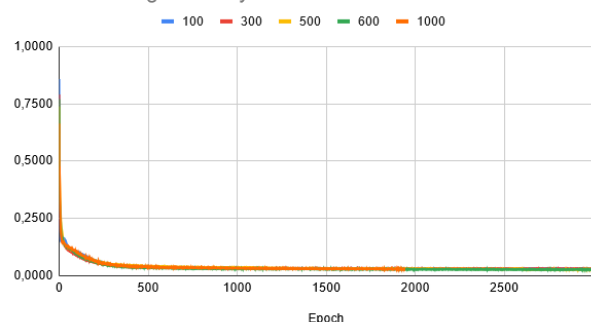
To get a more thorough understanding of the models we were dealing with, we researched the impact of changing a hyperparameter of the neural network models: the number of hidden neurons in the network. We used the levels 100, 300, 500, 600 and 1000 with 600 labels of the 50000 training samples. We suspected that a model with 1000 hidden neurons would be able to lower the test error, but this feat would take a significant amount of resources to train compared to its counterparts with not as many hidden neurons.

| | 100 neurons | 300 neurons | 500 neurons | 600 neurons | 1000 neurons |
|-----------------|-------------|-------------|-------------|-------------|--------------|
| Run 1 (s: 1000) | 3.40 | 3.03 | 3.56 | 2.67 | 2.82 |
| Run 2 (s: 2000) | 3.10 | 3.42 | 2.74 | 2.98 | 2.73 |
| Average | 3.25 | 3.23 | 3.05 | 2.84 | 2.77 |

Neurons testing efficiency



Neurons testing efficiency



In the table above, it can be seen that adding neurons to the deep neural network has a positive effect on lowering the test error. The results in this table are generated from 2000 epochs, as training deep neural networks with 1000 hidden neurons takes a long time. However, with all these models, the test error does drop fairly fast and remains stable after 1500 epochs.

Conclusion

We regard this paper as reproduced. While not fully independent as we have not implemented the code ourselves from the paper, the results of the paper could be reproduced with the code. We found an improvement on the results given in the paper, but when running the code on Azure we unfortunately did not manage to get the same results after 2500 Epochs. If we had more time, and thus managed to run 3000 Epochs we believe that our results will be about the same as the results of the authors.

More hidden neurons contribute positively to the model, but impact the training time a lot. Going from 100 to 1000 neurons yields a 15% improvement.

Furthermore, we hope that further research might be conducted more easily with this port from Python 2 to Python 3. We have also written a tutorial on how to properly set up this project for testing.

Reflection

During our reproduction and porting of this Paper we learned a lot of things about this paper in particular, but also about reproducing papers as a whole. And the importance of reproducibility for papers. We believe that with the knowledge obtained from the paper we could do a rudimentary reproduction, but the author of the paper did not go into enough detail on the various parameters needed to have a 100% accurate reproduction. No runtime or specs were mentioned as well, making it harder to do a time comparison between the Python versions.

But all in all we learned a lot about what it takes to reproduce a paper, and that it is a lot more than just pulling the code from github and running it a bunch of time. It requires an extensive understanding of the paper in itself, and what it is trying to achieve.