Artificial Neural Networks with small Datasets. A practical Approach

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science in Engineering (M.Sc.)

Eingereicht bei:

Fachhochschule Kufstein Tirol Bildungs GmbH

Data Science & Intelligent Analytics

Verfasser:

Paul Leitner, BA

1910837299

Erstgutachter : Dr. Johannes Luethi

Zweitgutachter : Lukas Demetz, PhD

Abgabedatum:

31. October 2021

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und in der Bearbeitung und Abfassung keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe. Die vorliegende Masterarbeit wurde noch nicht anderweitig für Prüfungszwecke vorgelegt.

Kufstein, 31. October 2021

Paul Leitner, BA

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

CNN Convolutional Neural Network

GB Gradient Boosting

nn Neural Network

ml Machine Learning

SMOTE synthetic minority oversampling technique

GAN Generative Adversarial Network

DCGAN Deep Convolutional GAN

FH Kufstein Tirol
Data Science & Intelligent Analytics

Abstract of the thesis: **Artificial Neural Networks with small Datasets. A practical Approach**

Author: Paul Leitner, BA
First reviewer: Dr. Johannes Luethi
Second reviewer: Lukas Demetz, PhD

After giving a summary on the literature and history of neural networks, I elucidate the trade-offs between deep learning and other machine learning approaches. I show that machine learning approaches such as Gradient Boosting (GB) mostly trade increased data requirements in favor of data scientist worktime in data preparation and feature engineering. I then investigate whether more complicated Neural Networks (nns) may be used by synthetically enlarging the training data present and thereby achieving comparable accuracy while saving data preparation time, effectively trading processing time (synthetic data enlargement being resource-intensive) for manual feature-engineering time by creating a nn model and benchmarking it against a GB reference model on a standard Machine Learning (ml) dataset with small data, the diabetic retinopathy dataset.

insert result - how much better does this perform? tradeoffs!

note - synthetic data Hittmeir et al. (2019)

31. October 2021

1. Introduction

In 2012 Krizhevsky and his colleagues entered and won the ImageNet classification contest with a deep convolutional neural network Convolutional Neural Network (CNN), outperforming other models by a significant margin Krizhevsky et al. (2012). This marked a turning point in machine learning in general, and in perceptual tasks specifically.

Pereira et al. (2009) is often invoked as a shorthand to the core problem of Machine Learning, the fact that a larger training corpus would always be preferable.

Currently, data scientists spend a significant amount (how much? sources!) of their time, when solving 'shallow' machine learning tasks (such as???) in feature engineering / preprocessing. Source! examples! This is due to the fact that shallow approaches such as decision trees, GBM and SVM models require features that 'directly' connect the prepared data to the searched-for outcome. (source)

Deep learning (neural networks) create intermediate representations via **stacked layers** at the cost of increased training data (source). Thereby enabling a more abstract understanding of patterns within the data.

Shearer (2000)

1.1 Problem Situation

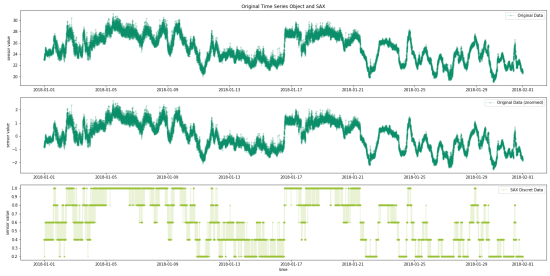

Figure 1: Sax approximation of a time series

As can be seen in Figure 1 . . .

1.2 Objectives

1.3 Methods

1.4 Structure

1.5 Tables

Table 1 shows an example table.

Table 1: This is a table

Column 1	Column 2	Column 3
A	B	C
D	E	F
G	H	I

1.6 Source Code

Listing 1: Hello World in Java

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Listing 1 shows the classic Hello World in Java.

Listing 2: Hello World in Python

```python
# This is a comment
print('Hello World')
```

Listing 2 shows the classic Hello World in Python.

2. Synthetic Data in Privacy

cite –> paper from source, different models on synthetic data!

2.1 Synthetic Data for model performance

When training nns for image classification, (source) a common practice is **data augmentation**, a range of random transformation applied to images in order to synthetically increase the breadth of data that the model is exposed to. Such operations include

- rotation
- shearing
- zoom
- height & width shift

effectively, these operations transform an Image while preserving the underlying signals in the data. However, with other types of data this might be possible. Attributes of another dataset may not be feasibly 'shifted' in one direction or another without fundamentally changing the signal and misleading the model.

note - the infeasibility of pretraining on non-image datasets - representations of the visual world

2.2 Deep Learning

3. Comparison with other solutions to the small data problem

- synthetic minority oversampling technique (SMOTE)
- crossvalidation (k-fold, single holdout)
- transfer learning (word embeddings, image filter layers)
- wholesale synthetic data approaches, Hittmeir et al. (2019) **more sources needed**

3.1 Data Enhancement for image data

4. Technical Application

4.1 Theoretical applicability

In their landmark paper in 2014, Goodfellow et al. (2014) demonstrated the viability of Generative Adversarial Networks (GANs) on creating image data on the classic MNIST dataset (Deng (2012)), by generating - among other things - convincing handwritten digits. As mentioned in 3.1, some of the architecture specifics and evaluation are quite specific to image data in that

- the data contains a notion of locality, as neighboring data points (i.e. pixels) are strongly dependent
- dimensionality of the generated data is higher than the **latent space**
- results lend themselves to visual quality inspection by humans (it is easy to see even degrees of quality between different architectures)

specifically the former points are strongly relevant to GAN architecture, as will become obvious shortly.

4.2 Technical implementation steps

Since the goal of this paper is to evaluate whether or not GAN may be used to not only generate more data of a small non-image dataset (which is fairly trivial) but whether or not this data actually serves to **boost model performance** of models trained on the resulting data, a small, well-understood standard dataset was used to develop the initial architecture; Farag and Hassan (2018). Specifically, the iconic titanic dataset constitutes a binary classification problem, which facilitates quick model evaluation and ameliorates some of the more typical difficulties of training GANs - see below.

The first attempts to create a basic, dense GAN actually failed to converge for a significant number of experiments with different amounts of layers, neurons and size of the latent space. Somewhat unsurprisingly, achieving the classic Nash Equilibrium between discriminator and generator was fairly difficult and the initial models all proved unstable. GANs provide several unique challenges, and/or failure modes:

- mode collapse Che et al. (2017)
- oscillation and general instability of the model Liang et al. (2018)
- catastrophic forgetting McCloskey and Cohen (1989)

Mode collapse is especially relevant in a task like MNIST, where there are multiple classes to be generated, and the generator becomes increasingly proficient in generating one class explicitly - thankfully, this is less of an issue in a binary classification task.

4.2.1 Network Architecture

The other failure modes, however **did** all make an appearance at one time or another, after the initial data preparation. It was fairly clear that the initial network, with one layer each for the generator and the discriminator each, and 64 neurons had insufficient representational power to converge on creating convincing samples as can be seen in 2:

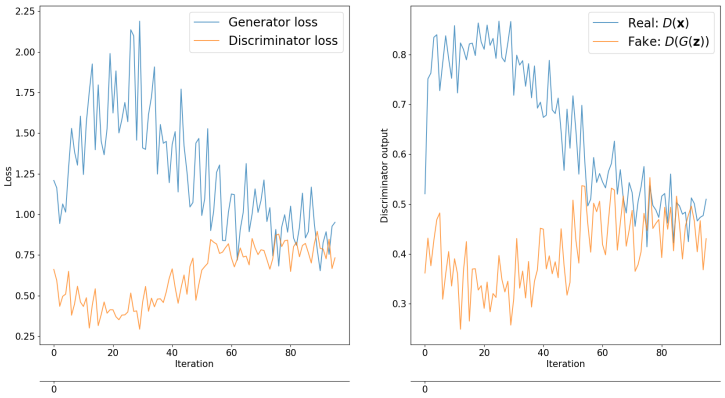

Figure 2: Initial simple dense GAN - left side shows the losses of generator and discriminator, right side shows the probabilities assigned to real and fake samples by the discriminator

Further experiments, with increased numbers of layers and neurons, produced first a very textbook oscillation pattern, shown in 3:

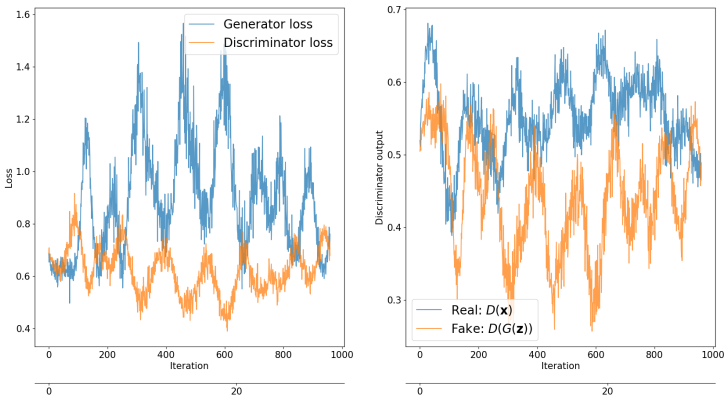

Figure 3: Dense GAN, 3 layers, 64 neurons/layer; left - losses of generator/discriminator right - the probabilities real/fake assigned by the discriminator

Finally, it has to be stressed that finding the ideal combination learning rates, dropout in the discriminator and number of training epochs, is really quite difficult, especially since there appears no good substitute to visually examining the pattern that is produced by a given architecture and then to adjust. A process that has to be iterated for quite a while, and is fairly manual and heavy on trial-and-error.

Ultimately, a a promising architecture appeared to be dense networks with 3 layers each, but a higher number of neurons, and still these networks diverged rather quickly shown here 4:

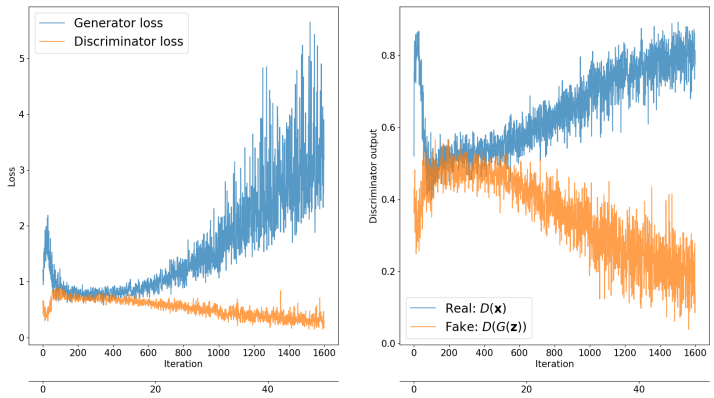

Figure 4: Dense GAN, 3 layers, 128 neurons/layer, reduced learning rate and dropout in discriminator - losses of generator/discriminator right - the probabilities real/fake assigned by the discriminator

Unfortunately, from here on out simply optimizing the number of neurons and learning rate and learning rate scheduling was not enough to mitigate this divergence. Although implementing the popular 1Cycle learning rate decay (described by Smith (2018)) did ameliorate the issue somewhat, it did not fix the network.

What ultimately made the difference is an adaptation of the architecture proposed by Radford et al. (2016). The architecture proposed here for image generation constitutes a **symmetrical** upsampling from the latent space in the generator (in case of images, a **transposed convolution**) and downsampling in the discriminator. As shown by Suh et al. (2019) here:

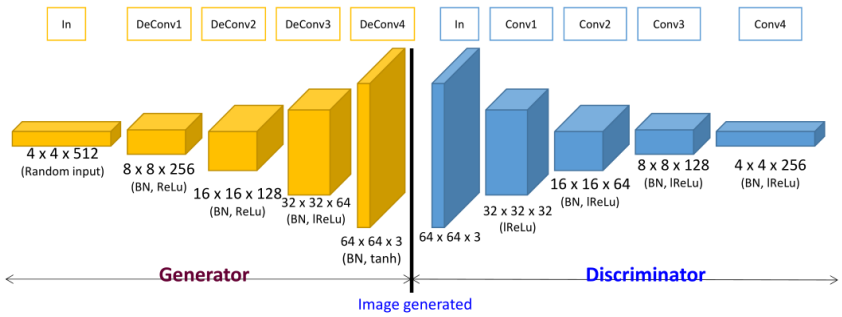

Figure 5: Architecture of Discriminator and Generator

Initially, implementing convolution actually deteriorated performance and completely prevented convergence of the network, a probable explanation would be the fact that convolution and transposed convolution not only upsample the latent space but more fundamentally relate to locality in the data; i.e. multiple convolutional layers over a picture effectively create hierarchical feature extraction. A paper that illustrates the mechanics of this fairly well was Dumoulin and Visin (2018). Effectively, these convolutions would initially find small features in images, subsequent convolutions would assemble these features into feature maps and their presence would indicate the presence of objects in an image. The entire concept of strides and adjacent data points however, does not make sense in the concept of a dataset where an observation consists of a feature vector, in which the order does not convey any information. While 1D convolutions are quite widely used in sequence and time-series processing - which are quite comfortably out of scope of this paper - they fundamentally seem unsuited to a dataset which would not lose any information if the order of its' attributes was permutated.

What **did** make a difference was implementing the symmetry of upsampling and condensing in the generator and discriminator.

Furthermore, Radford et al. (2016) propose other guidelines for building Deep Convolutional GANs (DCGANs) which proved helpful:

- implementing BatchNormalization in the generator and discriminator Ioffe and Szegedy (2015)
- using ReLU activation in all layers in the generator except for the output, which would use tanh
- using LeakyReLU in all layers in the discriminator, except for the output which uses sigmoid

After implementing these guidelines, using Binary Categorical Crossentropy loss, the generator and discriminator actually converged fairly well already, as seen in 6

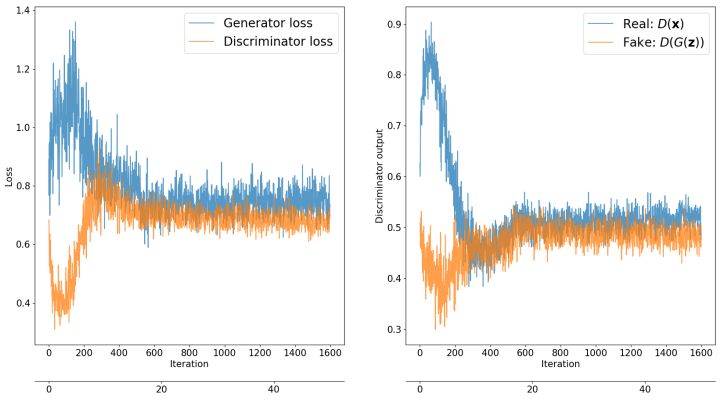

Figure 6: Dense GAN, 2 layers, 32 neurons/layer; left - losses of generator/discriminator right - the probabilities real/fake assigned by the discriminator

Importantly, in this architecture the **generator** model starts with a dense layer containing 32 neurons, which doubles in every layer (once in this case, although

this is a variable parameter). The final dense hidden layer is then downsampled again to reflect the original data - like this:

Listing 3: generator network

```python
import tensorflow as tf


def create_generator_network(
    number_hidden_layers: int = 2,
    number_hidden_units_power: int = 5,
    hidden_activation_function: str = 'ReLU',
    use_dropout: bool = False,
    upsampling: bool = True,
    use_batchnorm: bool = True,
    dropout_rate: float = 0.3,
    number_output_units: int = 12,
    output_activation_function: str = 'tanh') -> tf.keras.Model:

    model = tf.keras.Sequential()
    for i in range(number_hidden_layers):

        if upsampling:
            # implements the guideline DCGAN - upsampling layers in the generator
            model.add(tf.keras.layers.Dense(2 ** (number_hidden_units_power + i), use_bias=False))

        else:
            model.add(tf.keras.layers.Dense(2 ** number_hidden_units_power, use_bias=False))

        if use_batchnorm:
            model.add(tf.keras.layers.BatchNormalization())
        else:
            pass

        model.add(tf.keras.layers.Activation(hidden_activation_function))

        if use_dropout:
            model.add(tf.keras.layers.Dropout(dropout_rate))
        else:
            pass

    model.add(tf.keras.layers.Dense(number_output_units))
    model.add(tf.keras.layers.Activation(output_activation_function))

    return model
```

Listing 3 shows the python function which creates the generator network.

The discriminator implements the exact mirror image of this pattern, beginning with the same amount of neurons after the input layer and downsampling by half each layer;

Listing 4: discriminator network

```python
import tensorflow as tf


def create_discriminator_network(
    number_hidden_layers: int = 2,
    number_hidden_units_power: int = 5,
    hidden_activation_function: str = 'LeakyReLU',
    use_dropout: bool = True,
    upsampling: bool = True,
    use_batchnorm: bool = True,
    dropout_rate: float = 0.3,
    number_output_units: int = 1) -> tf.keras.Model:

    model = tf.keras.Sequential()

    for i in range(number_hidden_layers):

        if upsampling:
            # implements the guideline - downsample in the discriminator network
            model.add(tf.keras.layers.Dense(2 ** (number_hidden_units_power + number_hidden_layers - i - 1)))

        else:
            model.add(tf.keras.layers.Dense(2 ** number_hidden_units_power))

        if use_batchnorm:
            model.add(tf.keras.layers.BatchNormalization())
        else:
            pass

        model.add(tf.keras.layers.Activation(hidden_activation_function))

        if use_dropout:
            model.add(tf.keras.layers.Dropout(dropout_rate))
        else:
            pass

    model.add(tf.keras.layers.Dense(number_output_units, activation=None))

    return model
```

Listing 4 shows the python function which creates the discriminator network.

5. Results

6. Discussion

Bibliography

Che, T., Li, Y., Jacob, A. P., Bengio, Y., and Li, W. (2017). Mode regularized generative adversarial networks.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Dumoulin, V. and Visin, F. (2018). A guide to convolution arithmetic for deep learning.

Farag, N. and Hassan, G. (2018). Predicting the survivors of the titanic kaggle, machine learning from disaster. In *Proceedings of the 7th International Conference on Software and Information Engineering*, ICSIE '18, page 32–37, New York, NY, USA. Association for Computing Machinery.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

Hittmeir, M., Ekelhart, A., and Mayer, R. (2019). On the utility of synthetic data: An empirical evaluation on machine learning tasks. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA. Association for Computing Machinery.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification

with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

Liang, K. J., Li, C., Wang, G., and Carin, L. (2018). Generative adversarial network training is a continual learning problem.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Pereira, F., Norvig, P., and Halevy, A. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(02):8–12.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

Shearer, C. (2000). The crisp-dm model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22.

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay.

Suh, S., Lee, H., Jo, J., Lukowicz, P., and Lee, Y. (2019). Generative oversampling method for imbalanced data on bearing fault detection and diagnosis. *Applied Sciences*, 9:746.

A. List of Interview Partners

B. Code Table