Latent Representation of Topographic Classes in Remote Sensing Image Data using Autoencoders

Hannes Stärk

August 12, 2019

Contents

1 Vorwort

2 Kurzfassung

3 Abstract

4 Introduction

4.1 Topic Overview

4.2 Research Questions

5 Background

This section contains basic knowledge that is necessary for the understanding of the rest of the thesis. It briefly touches on artificial neural networks and convolutional neural networks in the beginning. The subsection after that goes on to describe autoencoders in general. Next up, Kullback-Leibler divergence is explained in detail as it is an essential part for the variational autoencoders as they are used in this work. The difference between those variational autoencoders and the standard autoencoder is covered in the next subsection. At last, the dimensionality reduction methods principal component analysis and t-distributed stochastic neighbor embedding is brought up since they are used in this work to gain an understanding of the high dimensional latent space of the autoencoders.

5.1 Remote Sensing Imagery

5.2 Artificial Neuron

Inspired by biological neurons an artificial neuron has one or more weighted inputs that are summed together and after that passed through a non-linear function called activation function. The output models the axon of a biological neuron to which other biological neurons can connect via their dendrites which is similar to an artificial neuron using the output of another artificial neuron as its input. By adjusting the separate weights of the inputs a single neuron is able to model simple functions like the logic *and* or the logic *or*.

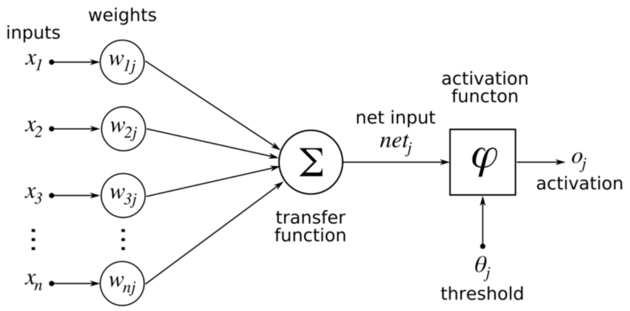

Figure 1: An artificial neuron. Image taken from (Chrislb 2005)

5.3 Artificial Neural Network

5.4 Convolutional Neural Networks

5.5 Autoencoders

5.6 Kullback-Leibler Divergence

5.6.1 Entropy

A part of the loss function used in the variational autoencoder is based on Kullback-Leibler divergence. To understand Kullback-Leibler divergence it seems necessary to explain entropy from the field of information theory. In short, entropy is a measure for the minimum average size an encoding for a piece of information can possibly have.
Suppose there is a system $S1$ that can have four different states a, b, c, d and every one of those states is equally likely to occur, that means the probability $P(x)$ of each state x is $1/4$. Now the goal is to losslessly transmit all information about that system with the minimum average amount of bits. That can be done with only two bits for example like this

$$a : 00 \quad b : 01 \quad c : 10 \quad d : 11$$

However, if $P(a) = 1$ and $P(b) = P(c) = P(d) = 0$, zero bits will suffice to encode the information since it is always certain that the system is in state a. So the entropy of the system clearly depends on the probabilities of each state. To see in which way, one can consider the system $S2$ with $P(a) = 1/2,\ P(b) = 1/4,\ P(c) = 1/8$ and $P(d) = 1/8$. In that case it would be best to encode the state with the highest probability with as few bits as possible since it has to be transmitted the most often. That means a is encoded with one bit as 0. When decoding the information there must be no ambiguities so while the encoding for b has to start with a 1 it cannot be 1 since we need to encode two more states so $b : 10$. Additionally if $c : 11$ there would be no space left for d: say $d : 111$ then if the transmitted information is $111111\ldots$ it could either be decoded to $ccc\ldots$ or $dd\ldots$. So c should rather be encoded as 110 which way $d : 111$ works. In the end a valid encoding that can transmit all information with the minimum average amount of bits is

$$a : 0 \quad b : 10 \quad c : 110 \quad d : 111$$

Here the states c, d are encoded with three bits instead of the two bits in the first example. But c and d are transmitted far less often than a which now only needs one bit. To be more precise half of all transmissions have one bit. Additionally a quarter of all transmissions have two bits. The sum of those probabilities multiplied with the respective amount of bits is the average amount of bits needed to transfer the information in a given encoding. So in the example, with $f(x)$ as the number of bits that encode a state x, that turns out to be $P(a)f(a) + P(b)f(b) + P(c)f(c) + P(d)f(d) = 1.75$. That means for $S2$ on average you only need 1.75 bits to encode a state and since that is also the minimum 1.75 is the entropy of $S2$.
In general in an optimal encoding $f(x)$ is the same as $\log_2 \frac{1}{P(x)}$. For $S1$ $P(x)$ is $1/4$ so the number of bits for x is $\log_2(4) = 2$ what matches the two bits the first encoding uses for the states of $S1$.
The entropy H of a system with a set of discrete events X and the probability distribution $P(x)$ for each $x \in X$ is

$$H(P) = \sum_{x \in X} P(x) \log_2 \frac{1}{P(x)} = -\sum_{x \in X} P(x) \log_2 P(x) \tag{1}$$

This is often written as the expectation for a given state x under the distribution P.

$$H(P) = E_{x \sim P}[-log_2 P(x)]$$

Intuitively if a system has high entropy, the size of the encodings are high on average and many states have small probabilities. This means it is hard to predict what state the system will be in at a given time since there is no state that can be guessed with high confidence. If entropy is low, zero for example, one can be confident that the system is in a certain state like in the previous example with $P(a) = 1$.

5.6.2 Cross Entropy

If the real distribution P of a system is unknown an estimate distribution Q could be guessed and encoding sizes $-log_2 Q(x)$ can be produced which will not be optimal for the true distribution P. Now with some data gathered and P known the used encoding sizes can be cross-checked with the expectation under the actual distribution resulting in the cross entropy $H(P, Q)$

$$H(P, Q) = E_{x \sim P}[-log_2 Q(x)] = -\sum_{x \in X} P(x) \log_2(Q(x)) \tag{2}$$

In machine learning tasks regarding classification this is often used as a loss function since the label of a piece of data gives us a distribution P with absolute certainty and $H(P) = 0$. With the inaccurate distribution Q that the model estimates $H(P, Q)$ will be greater than zero unless $P = Q$ where $H(P, Q) = H(P, P) = H(P) = 0$. So the learning algorithm can try to minimize $H(P, Q)$.

5.6.3 Kullback-Leibler Divergence

Having computed the entropy $H(P)$ and cross-entropy $H(P, Q)$ of two distributions P, Q it is possible to compare those distributions by comparing $H(P)$ and $H(P, Q)$ through subtraction

$$\begin{aligned} D_{KL}(P \parallel Q) &= H(P, Q) - H(P) \\ &= E_{x\sim P}[-log_2 Q(x)] - E_{x\sim P}[-log_2 P(x)] \\ &= E_{x\sim P}[-log_2 Q(x) + log_2 P(x)] \\ &= E_{x\sim P}[log_2 \frac{P(x)}{Q(x)}] \\ &= \sum_{x\in X} P(x) log_2 \frac{P(x)}{Q(x)} \end{aligned} \tag{3}$$

where $D_{KL}(P \parallel Q)$ is called the Kullback-Leibler divergence of P and Q. This works because $D_{KL}(P \parallel Q)$ is zero if Q and P are the same since that means $H(P, Q) = H(P)$. On the opposite if Q is different from P then $H(P, Q)$ is greater than $H(P)$ and therefore the KL divergence is greater than zero proportional to how different Q and P are. In summary Kullback-Leibler divergence is a measure of how different two probability distributions are that is zero if they are the same and greater than zero if not.

5.7 Variational Autoencoders

5.8 Principal Component Analysis

5.9 t-Distributed Stochastic Neighbor Embedding

6 Related Work

7 Methodology

In this section the subsection environment will explain what tools were used to program the autoencoders and what hardware the developed models were trained and written on. The subsection Datasets contains information about the remote sensing image data that was analyzed using the autoencoders. The architecture subsection lays out the design of the different models regarding the layers and loss functions. In the last subsection latent space there are the techniques used to visualize and understand the latent code between the encoder and decoder with comparisons of the latent space between normal autoencoders and variational autoencoders.

7.1 Environment

7.1.1 Hardware

Training autoencoders with larger images of multiple channels takes a lot of computation especially if the features that should be learned are as complicated and abstract as the topography of remote sensing data. As this is very time consuming the training of the models was mainly done on a remote machine from the Leibniz University in Hannover that could run 24/7. Only the programming of the models and tests with small datasets took place on a local personal computer that had no GPU that was capable of training models in the given scenario. The following are the specifications of the used systems:

Personal Computer

AMD Ryzen 7 1700, 16 GB RAM, NVIDIA GeForce GTX 760 2GB

Remote Machine

Lenovo Legion Y520T-25IKL, Intel i7-7700, 8GB RAM, NVIDIA GeForce GTX 1060 3GB

7.1.2 Software

The programming language used for the work is Python with the development environment PyCharm locally and and Jupyter Notebooks on a remote machine. The machine learning framework Tensorflow was used which is developed by Google and offers cross-platform support, running on most CPUs and GPUs (Google n.d.). Whenever possible Tensorflows implementation of the Keras API specification was used which is a high level API for training machine learning models.

7.2 Datasets

The available data are 1024x1024 images of the two United States cities Jacksonville in Florida and Omaha in Nebraska taken from the US3D Dataset that was partially published to provide research data for the problem of 3D reconstruction (Bosch et al. 2019). The images for each recorded area cover one square kilometer and can be divided into four categories with the first one being multispectral satellite images with eight channels (MSI). From the MSI data three channels were extracted and used as red, green and blue intensities (RGB). Thirdly there are digital surface models (DSM) and Lastly semantic labeling with five different categories.

The MSI data was collected by the WorldView-3 satellite of Digital Globe from 2014 to 2016. Thereby the images were taken in different seasons and times of day leading to great differences in their appearance regarding for instance shadows, reflections, overall brightness or clouds. This is an advantage for training models that are capable of processing data with similar differences in appearance. In the MSI dataset a single picture consists of eight channels for eight different bands of the spectrum with a ground sample distance of 1.3 meters. The eight channels of the imagery correspond to the following wavelengths:

- Coastal: 400 - 450 nm
- Blue: 450 - 510 nm
- Green: 510 - 580 nm
- Yellow: 585 - 625 nm

1. Red: 630 - 690 nm
2. Red Edge: 705 - 745 nm
3. Near-IR1: 770 - 895 nm
4. Near-IR2: 860 - 1040 nm

Three of those channels were extracted and used as RGB data. Each pixel of an image is described by three bytes representing the intensity of the wavelengths of the reflected light corresponding to either red, green or blue.
The DSM data was collected using light detection and ranging technology (Lidar) which measures the distance to points of earths surface. This distance is proportional to the value of the single channel that each pixel of the DSM has. Lastly there are semantic labeled pictures with one channel of a single byte that encodes one of five different topographic classes. Those classes are vegetation, water, ground, building and clutter. The semantic labeling was done automatically from lidar data but manually checked and corrected afterwards. Those four categories of data all cover one square kilometer in each image. Additionally they contain a lot of oblique view on buildings and other valuable features like clear shadows making the data ideal for training models that should detect those features.
In the dataset there are $2,783$ 1024×1024 RGB images available. Since the autoencoder should be able to distinguish between categories like shadows and vegetation each image is split into 64 128×128 pictures resulting in $178,112$ total training images. Those smaller picture sections are more likely to have a dominant feature like containing mainly shadows or only vegetation.

7.3 Architecture

The first general structure of the vanilla autoencoder and the variational autoencoder is a combination of the convolutional autoencoder and the variational autoencoder presented in Hans-On Machine Learning (Geron n.d.). That model is adjusted to work with 128×128 RGB images

7.3.1 The Loss Function

Intuitively the loss function defines a goal that the model should reach when training by minimizing the loss function. In the variational autoencoder one of those goals is that the distribution of the latent space is similar to a normal distribution since that makes it possible to sample latent variables from a normal distribution. From those sampled variables the decoder can generate an output that resembles the real distribution.
To define that goal in a loss function Kullback-Leibler divergence is used to force the distribution of the latent space to resemble a normal distribution.
The second goal is of course that the output of the variational autoencoder is similar to the input. The loss function used for this purpose is the sum of the absolute differences between each input and each output pixel.
The final loss function is the sum of both, the Kullback-Leibler divergence and the sum of the absolute errors.

7.4 Latent Space

8 Conclusion and Future Work

9 Conclusion

10 Future Work

References

Bosch, Marc et al. (2019). “Semantic Stereo for Incidental Satellite Images”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV).* IEEE, pp. 1524–1532.

Chrislb (2005). *Diagram of an artificial neuron.* [Online; accessed August 08, 2019][CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)].

Geron, Aurelien (n.d.). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O’Reilly UK Ltd. ISBN: 9781492032649.

Google (n.d.). *Tensorflow.*