

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER's THESIS No. 1413

**Identification of 1D-Signal Types Using
Unsupervised Deep Learning**

Jan Tomljanović

Zagreb, June 2017.

Zagreb, 3 March 2017

MASTER THESIS ASSIGNMENT No. 1413

Student: **Jan Tomljanović (0036473882)**
Study: Computing
Profile: Computer Science

Title: **Identification of 1D-Signal Types Using Unsupervised Deep Learning**

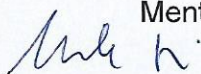
Description:

During the de novo genome assembly process, an important step is overlap of sequenced reads. False overlaps impede successful sequence reconstruction. It is possible to construct a per-base coverage graph for each read. From this graph, we can obtain a 1D-signal that might be further analysed. Different signal types represent different types of reads that might produce false overlaps (i.e., false overlaps due to chimeric or repeat reads).

The task is to develop a method for identification of these signal types based on deep learning. In the first step use a feature extractor such as autoencoder to obtain compressed signal representation for each signal instance. In the second step perform clustering of the obtained representations using standard clustering methods such as k-means or spectral clustering methods. Perform visualization of obtained representations and clusters using t-Distributed Stochastic Neighbor Embedding (t-SNE) and evaluate achieved results. Implement method using TensorFlow library. The code should be documented and hosted on a publicly available Github repository.

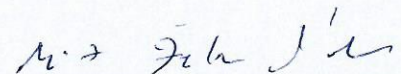
Issue date: 10 March 2017
Submission date: 29 June 2017

Mentor:



Associate Professor Mile Šikić, PhD

Committee Chair:



Full Professor Siniša Srbljić, PhD

Committee Secretary:



Assistant Professor Tomislav Hrkać, PhD

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1413

Pristupnik: **Jan Tomljanović (0036473882)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Identifikacija tipova 1D-signala pomoću nenadziranog dubokog učenja**

Opis zadatka:

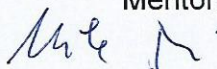
Tijekom de novo sastavljanja genoma, važan korak je pronalaženje preklapanja među očitanjima. Pogrešno detektirana preklapanja onemogućuju ispravno sastavljanje originalne sekvence. Za svako očitavanje moguće je izgraditi graf pokrivenosti svake baze. Takav graf daje nam 1D-signal koji se može dodatno analizirati. Različiti tipovi signala predstavljaju različite tipove očitavanja koja mogu rezultirati pogrešnim preklapanjem (npr. pogrešna preklapanja uzrokovana kimernim očitanjima ili očitanjima s ponavljanjem).

Zadatak ovog rada je koristeći tehnike dubokog učenja razviti metodu za identifikaciju tipova signal koji uzrokuju nepostojeća preklapanja. U prvom koraku treba koristiti ekstraktor značajki (npr. autoenkoder) da bi se dobio sažeti prikaz signala. U drugom koraku treba obaviti grupiranje sažetih prikaza koristeći neku od standardnih metoda za grupiranje kao što je k-means ili spektralno grupiranje. Dobivene grupe vizualizirati pomoću metode t-SNE (using t-Distributed Stochastic Neighbor Embedding) i procijeniti kvalitetu. Implementirati metodu pomoću programske biblioteke TensorFlow. Programski kod treba biti dokumentiran i javno dostupan preko repozitorija GitHub.

Zadatak uručen pristupniku: 10. ožujka 2017.

Rok za predaju rada: 29. lipnja 2017.

Mentor:



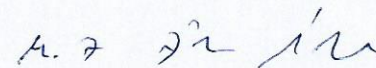
Izv. prof. dr. sc. Mile Šikić

Djelovođa:



Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za
diplomski rad profila:



Prof. dr. sc. Siniša Srbljić

Contents

Introduction.....	1
1. Overview.....	2
1.1. DNA sequencing.....	2
1.2. Overlap-layout-consensus.....	2
1.3. Types of reads.....	3
1.3.1. Regular read.....	3
1.3.2. Chimeric read.....	4
1.3.3. Repeat read.....	6
2. Dataset.....	8
2.1. Finding overlaps.....	8
2.2. Balancing the dataset.....	8
2.3. Heuristic algorithm.....	9
3. Methods.....	11
3.1. Machine learning.....	11
3.2. Supervised Learning.....	12
3.3. Artificial neural network (ANN).....	14
3.4. Training the ANN.....	16
3.5. Deep learning.....	16
3.6. Autoencoder.....	17
3.7. Variational autoencoder.....	18

3.7.1. Manifold hypothesis.....	18
3.7.2. Decoder.....	20
3.7.3. Encoder.....	22
3.7.4. Training the variational autoencoder.....	22
3.7.5. Reparametrization trick.....	26
3.8. Denoising Autoencoder.....	27
3.9. K-means clustering algorithm.....	28
3.10. Spectral clustering.....	30
3.11. Overview of approach.....	33
4. Results and discussion.....	34
4.1. K-means along variational autoencoder.....	34
4.1.1. Results.....	34
4.1.2. Discussion.....	39
4.2. K-means along denoising autoencoder.....	41
4.2.1. Results.....	41
4.2.2. Discussion.....	46
4.3. Spectral clustering along variational autoencoder.....	48
4.3.1. Results.....	48
4.3.2. Discussion.....	52
4.4. Comparison.....	54
4.5. Application.....	55
Conclusion.....	56
Bibliography.....	57

Introduction

In bioinformatics, de novo genome assembly is a process of creating a genome sequence out of small fragments (Šikić et al, 2013). This process is needed because DNA sequencing technology cannot read the whole genome, but rather only a small part of it. To reconstruct the whole DNA sequence, many of these small parts must be put together to form the whole genome. In this process, one of the steps is observing overlaps between these small fragments. Using overlaps, it is possible to chain fragments together. For each read (small fragment gathered using DNA sequencer) it is possible to construct a coverage graph in which it can be seen how well each segment of that read overlaps with others. This coverage graph can be analysed as a 1D-signal. Multiple types of signals are known, as they represent different types of reads. Knowing which read is of which type is important during the process of chaining reads together.

After generating signals, it is possible to approach classification of signals into different classes (types) in many different ways. Often algorithms based on heuristics are used since this is a novel problem and there is no general agreement on which approach works best. In this thesis, the idea is to explore how well clustering, i.e. unsupervised learning, works to separate different types of signals. This way, we hope clusters of signals which naturally emerge from the data will correspond to the types we know. In the case they don't, perhaps new types of reads could be discovered.

Chapter 1 gives an overview of the topic and explains known types of reads in detail. The topic of Chapter 2 is data used in this thesis and how it is obtained. Chapter 3 describes methods used to encode the data and then cluster it. Results along with discussion are presented in Chapter 4.

1. Overview

1.1. DNA sequencing

DNA sequencing is a process whose goal is to obtain the order of nucleotides within a DNA molecule. DNA sequencer is an instrument that can extract reads (small fragments of the DNA sequence) from a molecule as textual data.

Throughout last couple of decades three generations of DNA sequencers appeared. The first generation used Sanger sequencing method (Metzker, 2005) which was used in humane genome project in 2001. Today, sequencers which use this type of technology can achieve read length of around 1 000 bp (base pairs) with 99.99% accuracy. Cost of these sequencers is about \$0.5 per kilobase (Shendure et al, 2008). There are multiple second generation DNA sequencing technologies and their read length is much lower, ranging from 13 bp to 350 bp (Shendure et al, 2008). But they are much cheaper then the previous technology as price per megabase ranges from \$1 to \$60. Recently, third generation DNA sequencers have been created, such as SMRT (Single molecule real time sequencing). Using this technology, obtained reads are longer, ranging from about 1 000 bp to 7 000 bp.

Since the goal of DNA sequencing is to produce the full genome, which is usually longer than 100 000 bp, reads which are obtained by any of the DNA sequencers mentioned still need to be assembled.

1.2. Overlap-layout-consensus

Overlap-layout-consensus (OLC) method is used for genome assembly which works with reads provided by the DNA sequencer (Šikić et al, 2013). As the name says, it consists of three steps:

1. Overlap – overlaps between reads are calculated and overlap graph is built
2. Layout – overlap graph is simplified
3. Consensus – remaining uncertain parts of the graph are decided

Complications during the first step of the method are the motivation for this thesis. Multiple different types of reads can result in different coverage graphs, which are built to show number of overlaps (per each base in the read) between the read in question and others. Some types of reads can cause coverage graphs which can significantly complicate the overlap graph and/or even make it completely incorrect. The matter in question is how to identify those reads using their coverage graphs.

1.3. Types of reads

Here, reads are classified into types according to their coverage graphs, therefore this is also the classification of the coverage graphs themselves.

1.3.1. Regular read

Regular reads are the “good” type of reads, meaning there are no specific problems regarding them. Regular read occurs when DNA sequencer reads a part of the genome, as can be seen in Figure 1.1.

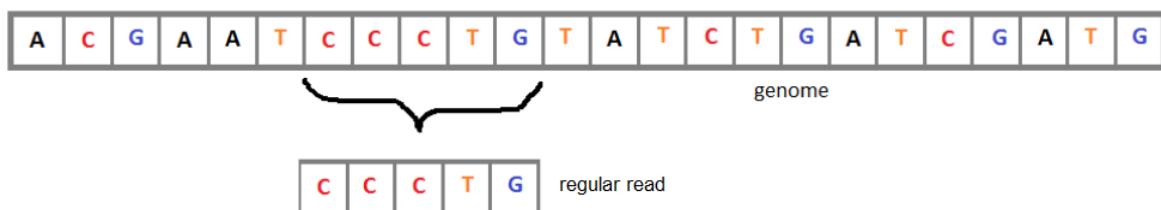


Figure 1.1. Regular read

When generating the coverage graph of a regular read, on average, all parts of the read will overlap with about the same number of other reads, making the graph more or less flat. Example of a coverage graph for a regular read is presented below in Figure 1.2. X-axis represents the read, for instance first position represents the first base of the read. Value on the y-axis represents a number of overlaps which contain the corresponding base. Values on axes are intentionally left behind since the focus is on the shape of the signal, values themselves have little meaning in this context.

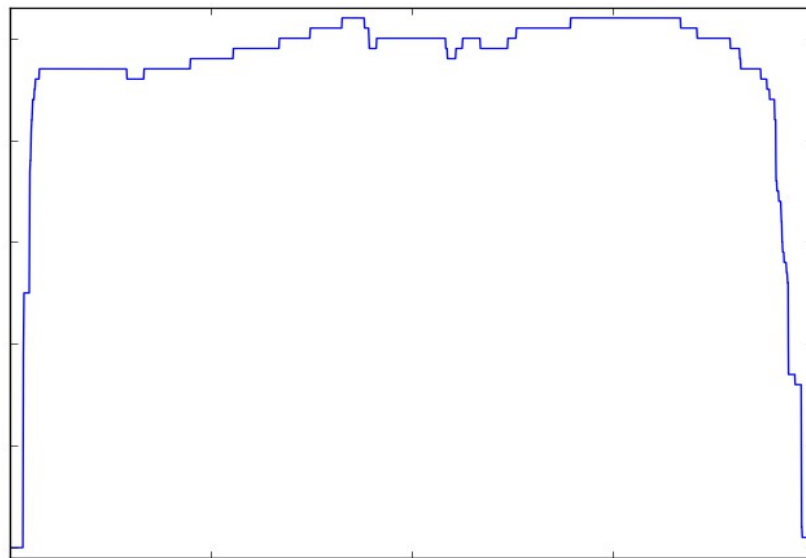


Figure 1.2. Coverage graph of a regular read

1.3.2. Chimeric read

Chimeric reads can pose a significant problem when creating the overlap graph. Chimeric read is created when DNA sequencer unexpectedly skips a part of the genome and continues reading. This jump creates a read which has two parts which map to different parts of the genome. Creation of a chimeric read is shown in Figure 1.3.

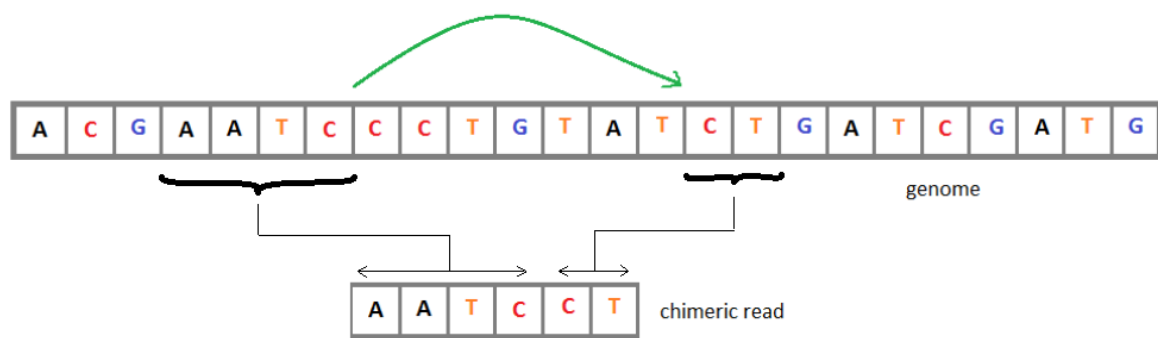


Figure 1.3. Chimeric read

Chimeric read can lead to problems when linking reads together, as it connects two parts of the genome that shouldn't be connected. Fortunately, chimeric reads are moderately rare, for instance in some datasets it is estimated they make up only 0.2% of the data. Nonetheless, detection of chimeric reads is necessary since each occurrence can lead to errors.

Coverage graph of a chimeric read is usually split in two ways by a very steep drop of the signal followed by an immediate steep rise. This occurs since left and right part of the read overlap with other reads. But almost no overlap is found along the part of the graph which connects left and right part. Example of the coverage graph of the chimeric read is given in Figure 1.4.

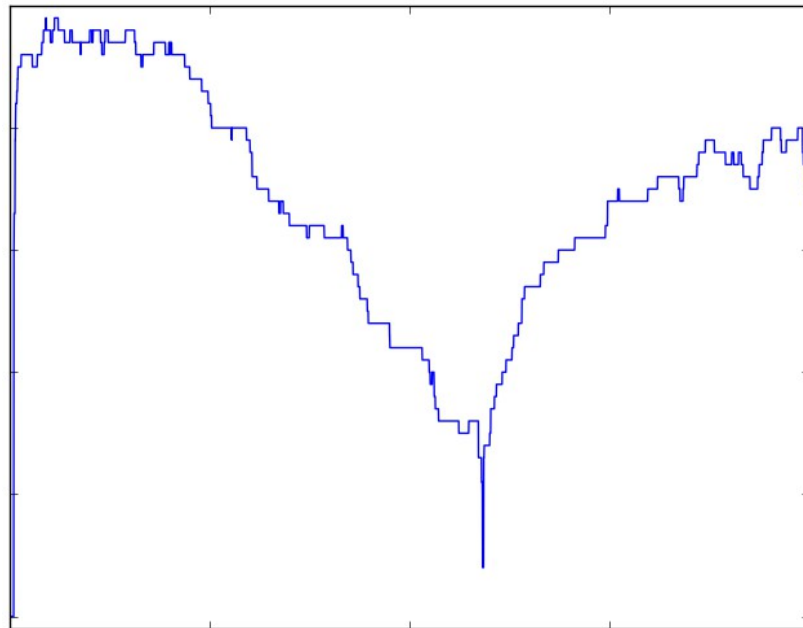


Figure 1.4. Coverage graph of a chimeric read

1.3.3. Repeat read

Repeat read is no different than a regular read regarding how it is created. It is a valid part of the genome. But what makes it special is the fact that a significant part of that read, in most cases its left or right side, is repeated throughout the rest of the genome and therefore can be found in other reads. Figure 1.5. shows how a left repeat is produced.

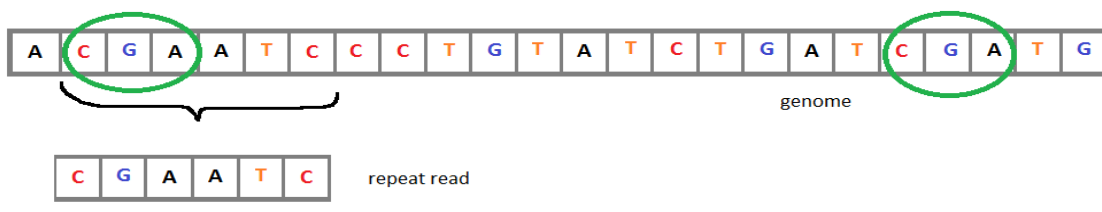


Figure 1.5. Repeat read

When coverage graph of a repeat read is generated, coverage will be higher on one side of the graph since more overlaps with other reads are found for that area. Example of a coverage graph of a left read is shown in Figure 1.6.

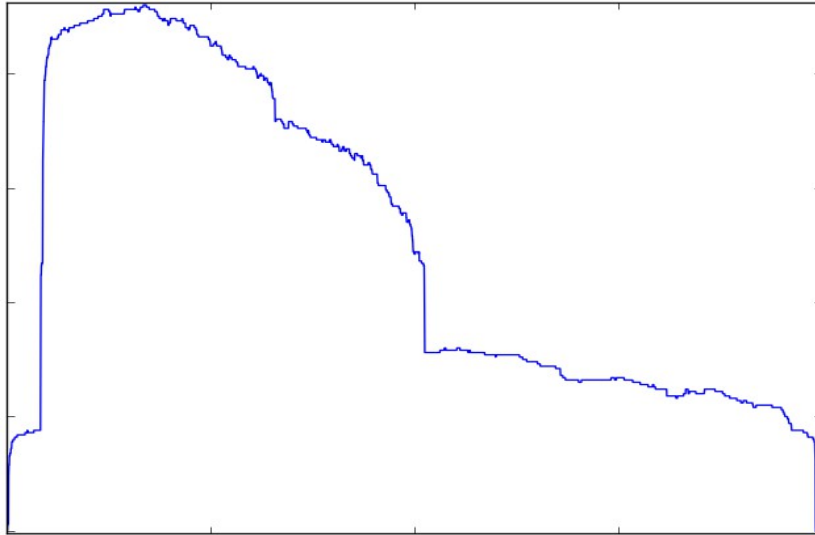


Figure 1.6. Coverage graph of a repeat read

Repeat reads can cause problems in the overlap graph as well. For instance, it is difficult to connect the left part of the left read with the rest of the genome, since there are multiple places where it can potentially fit. Repeat reads are less common than regular reads, but much more common than chimeric reads.

2. Dataset

2.1. Finding overlaps

Raw data that was available for this thesis were reads from many different bacteria. First step in analysis was getting coverage graphs generated from these reads. For that, graphmap tool (Sović et al, 2015) set to owler mode was used. This step provided about 800k coverage graphs, to which it can also be referred to as signals. As stated above, signals of regular, chimeric and repeat reads are not equally represented in the dataset. Since this thesis uses deep learning methods, balanced dataset is preferred (Dalyac et al, 2014) if we hope to identify coverage graphs of regular, chimeric and repeat reads as different classes of signals.

2.2. Balancing the dataset

Balancing the dataset starts from the assumption about the classes which we would like to find. Even though unsupervised algorithm is used that does not mean we don't have any idea about which classes of signals we want to find. Supervised approach would make more sense for the classification task, but that would require manually labelling a huge amount of data. Using unsupervised approach, the goal is to try to produce clusters we already know about, but with unlabelled data and also to check whether it is possible to find some clusters in the data we are not aware of. Based on visual identity four classes are identified:

- regular signals
- chimeric signals
- left repeat signals
- right repeat signals

where signals are named after the type of read they are based upon.

The original dataset, using a heuristic algorithm developed for this thesis (Section 2.3.), was classified into those four classes thus dividing the dataset into four parts. One of the goals of this thesis is to provide a much better classifier than that algorithm whose accuracy can only be estimated since data is not labelled (accuracy estimations range from 20% to 70%). Of the produced four parts of the dataset, the one with the least amount of signals was taken whole and the rest were reduced to having a similar number of signals as that part. That way, new dataset containing 21 173 signals was created and it can be assumed that it is somewhat balanced.

2.3. Heuristic algorithm

Heuristic algorithm was used as a classifier which decided whether signal is regular, chimeric, left repeat or right repeat. Simple Pseudocode 2.1. follows:

```
input: signal
output: type of signal
classify (signal) :
    if condition_one:
        return determine_if_left_or_right_repeat(signal)
    else if condition_two:
        return "chimeric"
    else:
        return "regular"
```

Pseudocode 2.1. Classifier

Before applying `classify` function, each signal was preprocessed. First, edges of the signal were removed, since signals often have a steep rise at the left side and a steep drop at the right side, which are not specific for any type of signal, so they are not useful for classification. After that, since signals differ in sizes on x-axis (read length) and y-axis (overlap number) sampling and vector normalization are

applied, in that order. This way, only the shape of the signal is considered and same criteria can be applied to all reads.

For determining whether signal is a repeat signal, `condition_one` was used. That condition took the following values into account:

- number of different values in the signal
- local maximum drop and rise
- average signal value left and right of the local maximum drop and rise
- position of the maximum value in the signal
- position of the local maximum drop and rise

Conditions regarding these values were adjusted according to a very small number of hand labelled data and put together as `condition_one` in Pseudocode 2.1. Deciding if a repeat signal is a left or right repeat was done simply by checking the average values of the left and right part of the signal.

For determining whether signal should be classified as a chimeric signal, `condition_two` was used. That condition took the following values into account:

- number of different values in the signal
- position of the local maximum drop
- steepness of the local maximum drop
- steepness of the possible rise that followed local maximum drop

Conditions regarding those values were adjusted in the same way as above.

3. Methods

3.1. Machine learning

Machine learning is a field within artificial intelligence that studies algorithms which can learn how to make decisions based on data. Machine learning tasks can be categorized in many ways.

Machine learning tasks can be split by the type of the feedback (or perhaps equivalently, structure of data used for learning) which is provided. There are three categories (Russell et al, 2009):

- supervised learning – algorithm learns how to make decisions based on available input-output pairs
- unsupervised learning – algorithm tries to find hidden structure in input data
- reinforcement learning – algorithm actively interacts with the environment and learns based on rewards and punishments

Both supervised and unsupervised learning are important for this thesis, reinforcement learning is not discussed further.

Machine learning can also be categorized depending on the type of output which is expected. Four categories are relevant for this thesis:

- classification – algorithm learns to assign class labels (from a certain set) to inputs, therefore deciding which input is a part of which class; supervised approach
- regression – unlike classification, where output is discrete, regression produces continuous output
- clustering – algorithm divides data into clusters, but in an unsupervised fashion as classes are previously unknown (contrary to classification)

- dimensionality reduction – input data is mapped into a lower dimensional space

3.2. Supervised Learning

Goal of supervised learning is to train a certain model to learn the connection between input and output data samples. It can be either a classifier connecting input to a class from a predefined set of classes, or a trained model which does regression connecting input to a continuous output. In this context, model is a tool that has the possibility of learning input-output connections. There are many different models which have their own advantages and limitations. Model learns how to produce the output value based on an input value.

In order to learn these connections, model must be presented with examples, i.e. data. Table 3.1. shows how data is structured.

Table 3.1. Data for supervised learning

Input	Output
$(x_{11}, x_{12}, \dots, x_{1n})$	$(y_{11}, y_{12}, \dots, y_{1r})$
$(x_{21}, x_{22}, \dots, x_{2n})$	$(y_{21}, y_{22}, \dots, y_{2r})$
...	...
$(x_{m1}, x_{m2}, \dots, x_{mn})$	$(y_{m1}, y_{m2}, \dots, y_{mr})$

There are m examples, and each consists of input vector and output vector (often output is only a single value). The simplest form of learning is presented in Pseudocode 3.1., assuming the model tries to learn a single example at a time, which isn't always the case.

```
while some condition:
    for input, output in examples:
        model.learn(input, output)
```

Pseudocode 3.1. Simple learning

Now, before addressing the stopping condition for learning, the question is how to evaluate how well did the model learn the connection between input and output. Would it be able to produce correct output for an input it hasn't seen before, thereby showing that it "understands" the data? That is usually achieved by separating a part of the available data and using it as a test set. Rest of the data is used as a training set. Model is being learned only using the training set. After learning, model is presented with input vectors of the test set. It produces the output vector for each input vector. The error which model produces is calculated by comparing those output vectors to known correct output vectors from the test set.

Now the question of stopping condition for learning can be discussed. Is it the case that more iterations of learning always produce better results on the test set? The answer relies upon the model that is used, but from that it can be inferred that generally answer is no. It can be the case that less iterations can produce better results. That is the case if the used model is very "powerful" and if given enough iterations it will learn the training set "by-heart", instead of learning the general connection between input and output. In that case, it is said the model has been overfitted. That model applied on the test set will produce bad results. This problem is often depicted in a graph, shown in Figure 3.1. X-axis represents the number of iterations of learning, and y-axis represents the error. Red curve shows the error on the test set, while the blue curve shows the error on the training set.

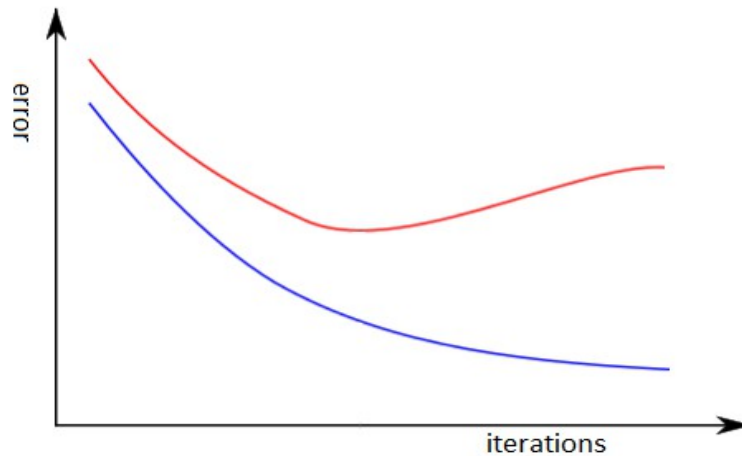


Figure 3.1. Bias – variance trade off graph

In order to find the spot on the graph where the model produces the smallest error, one additional set of examples is used, called validation set. Therefore, for a single learning run, data is split into three sets: training set, validation set and test set. While showing the training set to the model, at some rate, model is tested using the validation set. When it is noted that error on the validation set starts to rise instead of drop, training is stopped. In that case, the red curve in the Figure 3.1. is actually the error on the validation set. Model can then be tested using the test set which the model had never seen before.

3.3. Artificial neural network (ANN)

Artificial neural network is a model used for machine learning. It imitates the structure of neurons in a human brain. Single artificial neuron is shown in Figure 3.2. Neuron performs a function of the provided input. Input to the neuron is a vector (x_1, x_2, \dots, x_n) . Each component is multiplied by a corresponding factor w_i , often referred to as weight. Sum of the results and the bias component is made. Activation function F is applied to the sum and the resulting value is the output of the neuron. Activation function is non-linear, thus making the neuron a non-linear function.

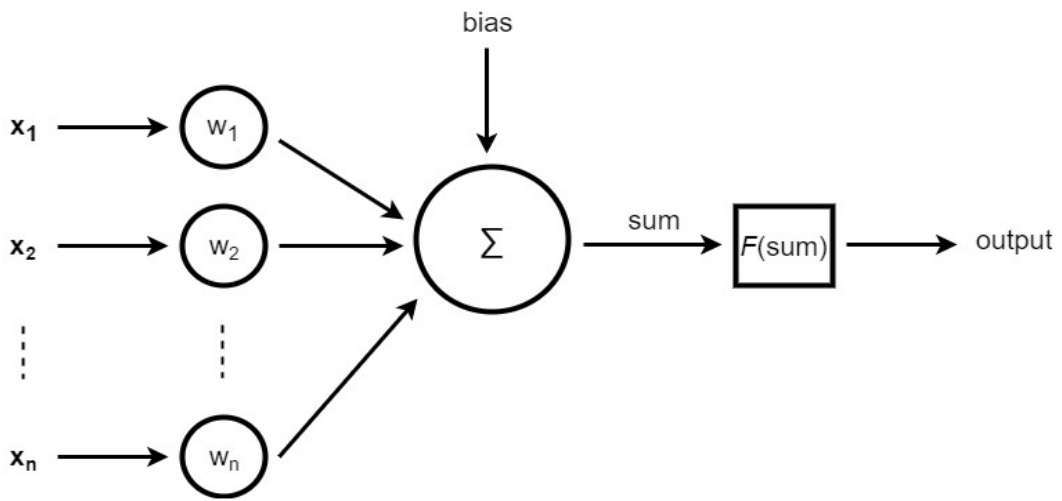


Figure 3.2. Artificial neuron structure

These neurons are combined together in a structure to form a fully connected neural network, as shown in Figure 3.3. The figure is shown only as an example, number of layers can vary, as well as the number of neurons in each layer. As a machine learning model used for supervised learning, artificial neural network operates in a following way. Input vector acts as the first layer, thus providing the data to the next layer. Output of the last layer is considered to be the output of the model for the given input.

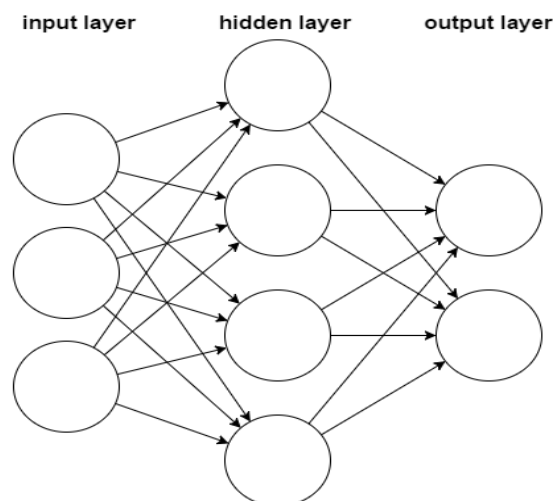


Figure 3.3. Artificial neural network

3.4. Training the ANN

In order for an ANN to learn the connections between the input and the output, parameters of the network must be adjusted. Each neuron has the following parameters:

- weights w_1, w_2, \dots, w_n
- bias

It is assumed activation function is fixed and has no parameters. Training the neural network is done as follows. First, some number of examples is chosen for a single training step. All examples (whole training set) can be chosen, a batch of examples or a single example. Single training step is explained as follows. For each chosen example (input-output pair), neural network calculates its output based on the input vector of the example. Error of the network is calculated by comparing the output of the network and the correct output for chosen examples. Since the error is indirectly the function of the parameters of the network, gradient of the error with respect to each of the parameters of neurons in the network can be found. At that moment, a step of a variation of the gradient descent method¹ is applied to parameters, thus adjusting them to try to decrease the error. This process is called backpropagation².

3.5. Deep learning

Deep learning is a part of machine learning which studies artificial neural networks which contain more than one hidden layer. Today, deep learning approaches have produced state-of-the-art results on many different tasks. Training of a neural network with many hidden layers is done the same way as with a simple ANN, as described above. There are additional challenges when dealing with deep neural networks, like the choice of the activation function. Simple ANNs often used the

1 https://en.wikipedia.org/wiki/Gradient_descent

2 <https://en.wikipedia.org/wiki/Backpropagation>

sigmoid function³ as the activation function. But as more layers are added to the network, backpropagation algorithm used for training the network becomes extremely inefficient if sigmoid function is used. Therefore new activation functions needed to be found. Today, perhaps the most commonly used activation function is Rectified linear unit⁴ (ReLU) which improves the performances of the training algorithm significantly.

3.6. Autoencoder

Autoencoder is an artificial neural network typically used for the purpose of dimensionality reduction. Main idea of the autoencoder is to provide data as the input to the ANN and to get the reconstructed input as the output of the network. If the ANN has a layer where number of neurons is smaller than the dimension of the input data, or if special learning techniques are used, information in that layer will be a compressed representation of the input vector. That way dimensionality reduction is achieved.

Figure 3.4. shows an example of the ANN used as an autoencoder. Vector $\mathbf{x} = (x_1, x_2, x_3, x_4)$ is the input data. Vector $\mathbf{x}' = (x_1', x_2', x_3', x_4')$ is the output of the network and reconstruction of the input data. Error of the network is zero if $\mathbf{x} = \mathbf{x}'$ and that is the best possible result. Training the ANN is done same as before using the backpropagation algorithm, where the desired output is exactly the same as input. But valuable information that can be gathered from this autoencoder are actually stored in vector $\mathbf{z} = (z_1, z_2)$, since, ideally, it contains enough information to reconstruct vector \mathbf{x} , thus it is a compressed representation of vector \mathbf{x} . Part of the network which produces vector \mathbf{z} , i.e. encodes \mathbf{x} into \mathbf{z} , is called encoder. Part of the network which takes \mathbf{z} as input and tries to reconstruct \mathbf{x} is called decoder.

3 https://en.wikipedia.org/wiki/Sigmoid_function

4 [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

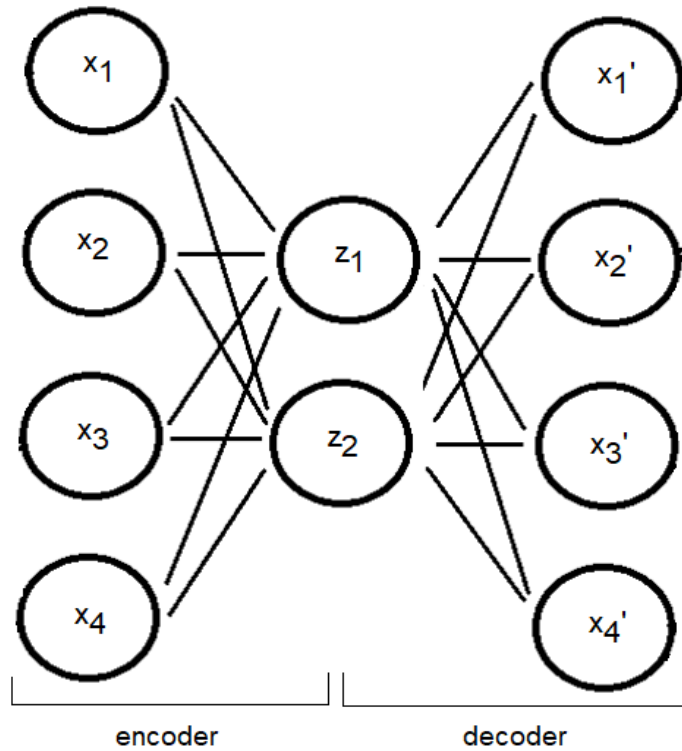


Figure 3.4. Autoencoder

In this thesis two different variants of autoencoders are used. They are used as a tool to reduce the dimensionality of coverage graphs from the dataset described in Section 2.

3.7. Variational autoencoder

3.7.1. Manifold hypothesis

Goal of the variational autoencoder is to successfully generate data which is very similar to the data in available dataset. By reaching that goal, with certain assumptions, variational autoencoder can then be used as a tool for dimensionality reduction of data.

In order to generate data, it is assumed that data samples, which are high dimensional vectors, are concentrated near a low dimensional manifold. This

assumption is called the manifold hypothesis. To demonstrate the assumption, example of 2D data is shown in Figure 3.5.

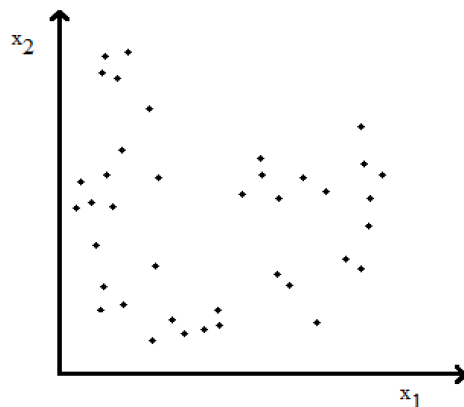


Figure 3.5. Example of a dataset

There is no obvious way in how these data points are generated. But it is possible to notice that they are distributed along a 1D manifold which is shown in Figure 3.6.

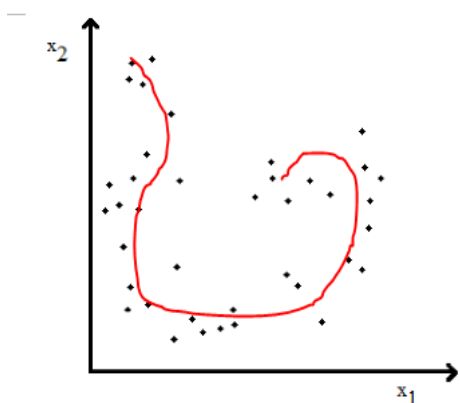


Figure 3.6. Data distributed along 1D manifold (red)



Figure 3.7. Number line

That 1D manifold is nothing but a number line shown in Figure 3.7.

Let the random variable which generates a data sample be represented as \mathbf{x} and random variable which generates a point on a manifold as \mathbf{z} . Variational autoencoder therefore states that there is a probability distribution $p_{\phi}(\mathbf{x}|\mathbf{z})$ which describes variable \mathbf{x} if variable \mathbf{z} is given, and conversely distribution $p_{\phi}(\mathbf{z}|\mathbf{x})$ which

describes variable \mathbf{z} is \mathbf{x} is given. These distributions are not independent and are parametrized with parameter ϕ . Idea of a variational autoencoder is depicted in Figure 3.8. There is a distribution $p_{\phi}(\mathbf{z}|\mathbf{x})$ by which it is possible to generate a point \mathbf{z} on a manifold with the help of a given data sample \mathbf{x} . Given that point \mathbf{z} it is possible to generate a data sample according to a distribution $p_{\phi}(\mathbf{x}|\mathbf{z})$.

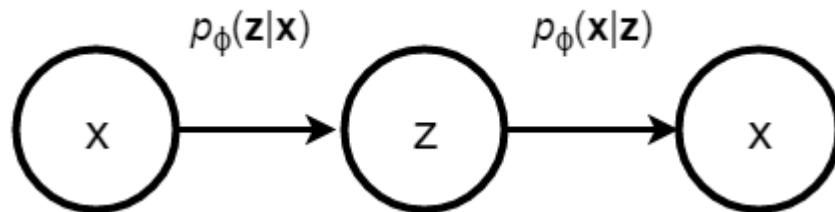


Figure 3.8. Variational autoencoder idea

3.7.2. Decoder

Decoder needs to act according to a distribution $p_{\phi}(\mathbf{x}|\mathbf{z})$. Since this probability distribution is very complex and unknown, the idea is to approximate it using a combination of a neural network and a normal distribution. What this approximation actually does is assume that $p_{\phi}(\mathbf{x}|\mathbf{z})$ can be described as a series of normal distributions distributed along the low dimensional manifold. Figure 3.9. shows this idea. For each point on the low dimensional manifold parameters of the normal distribution can be found. That normal distribution is used to generate data. To make decoder design fairly simple, normal distribution is assumed to have a diagonal covariance matrix.

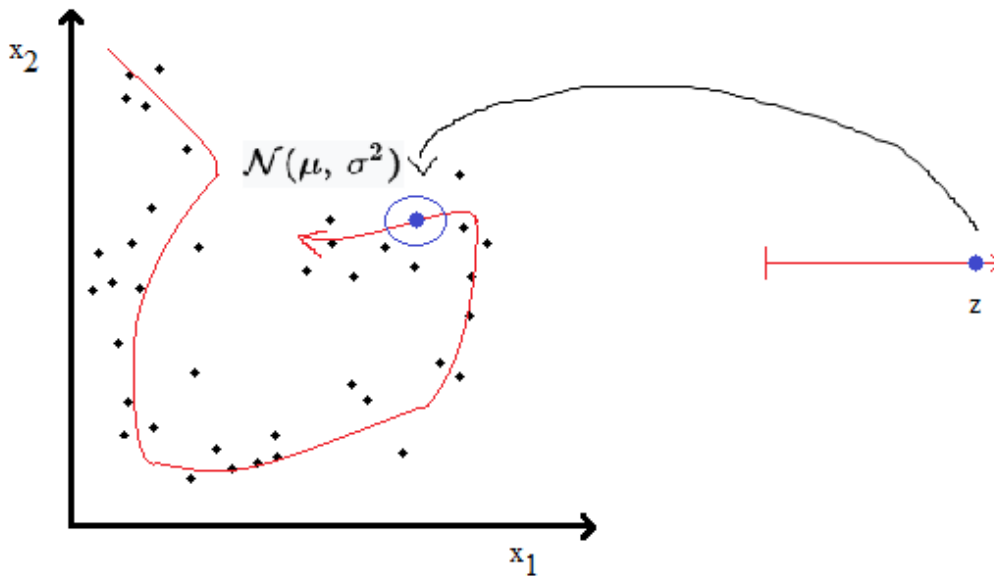


Figure 3.9. Approximation of a distribution – parameters of a normal distribution which generates data can be calculated for each point on the manifold

Decoder is shown in Figure 3.10. Number of neurons in each layer (including the first and the last) is chosen only as an example (here and in following figures) and is very different (larger) in practice. Purpose of the neural network is to calculate the parameters of the normal distribution given the point on the manifold. Sampling from the normal distribution can then be done to acquire a data sample.

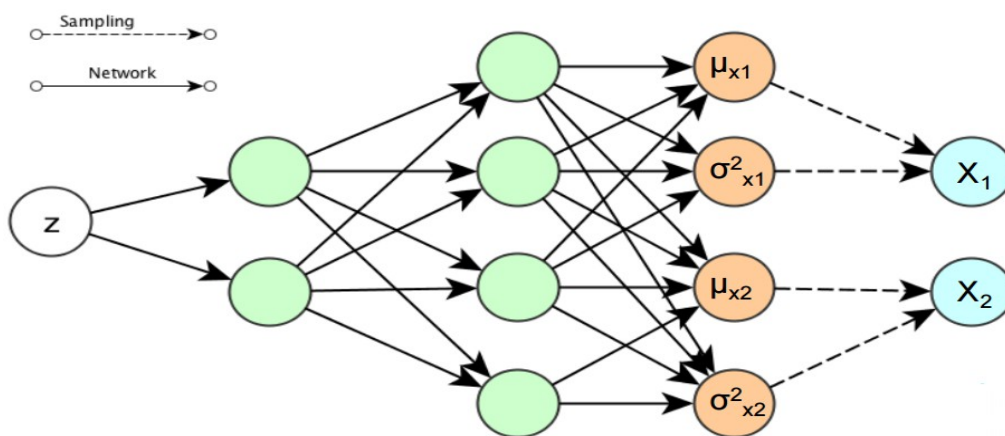


Figure 3.10. Decoder (Dürr, 2016)

3.7.3. Encoder

Task of the encoder is to find the point on the low dimensional manifold which corresponds to the data sample. That point can be considered to be the latent representation of that data sample. Vector space of the manifold is called latent space. Probability distribution $p_{\phi}(\mathbf{z}|\mathbf{x})$ describes this relationship. But again, this distribution is very complex and is approximated as a series of normal distributions which can generate points in latent space. Data sample is used to find parameters of the normal distribution. Basically, the approach is the same as with the decoder, but reversed.

Neural network is used to calculate the parameters and then latent representation can be found via sampling. Outlook of the encoder is shown in Figure 3.11.

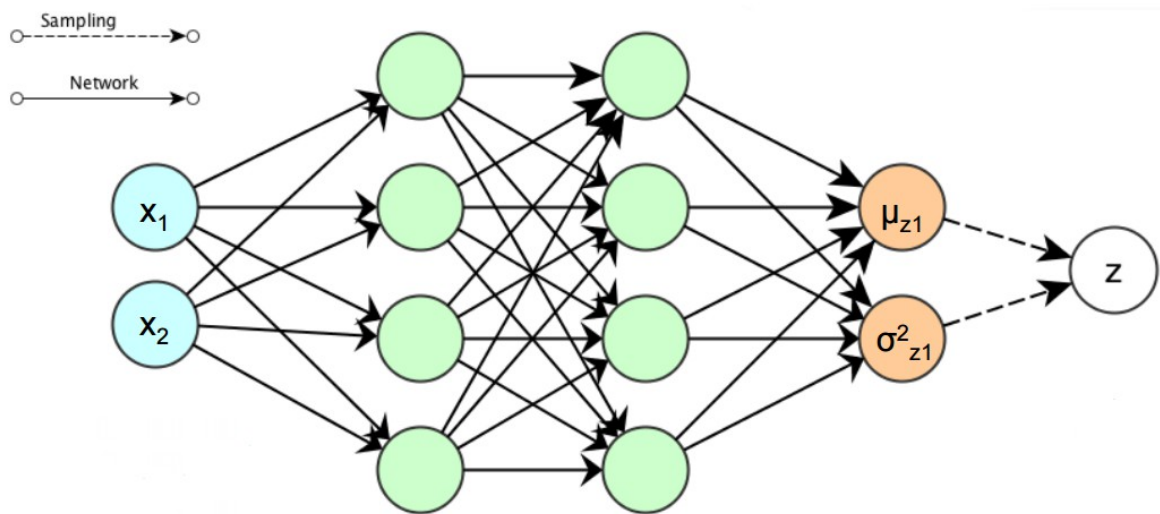


Figure 3.11. Encoder (Dürr, 2016)

3.7.4. Training the variational autoencoder

Variational autoencoder is trained in a supervised fashion like a neural network using the backpropagation algorithm, described in Section 3.4. The whole network is shown in Figure 3.12. However, gradient cannot propagate backwards through random sampling. Solution to that problem is presented later in Section 3.7.5.

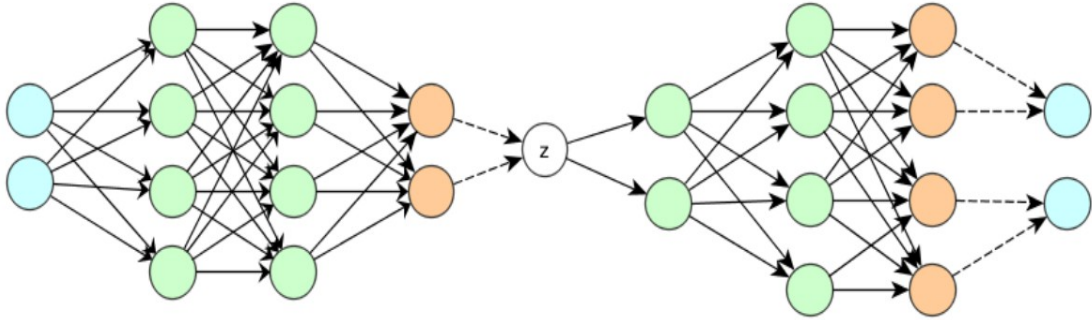


Figure 3.12. Variational autoencoder (Dürr, 2016)

Question remains, how is the error of the network calculated. Idea of the network is to maximize the probability that the sample shown to the network will be generated as output. This approach is well known in statistics and is called MLE (maximum likelihood estimation). By using MLE, error function of the network is found. Error is also referred to as cost, and accordingly, error function is the cost function.

First, log-likelihood of the random variable which represents the data sample \mathbf{x} (from the training set) being the output of the network is specified in equation (1) (logarithm is applied because it makes the calculation easier). Goal of training the neural network is to maximize that likelihood.

$$L = \log(p(x)) \quad (1)$$

Distribution which approximates $p_\phi(\mathbf{z}|\mathbf{x})$ is denoted as $q(\mathbf{z}|\mathbf{x})$. Parameters of distribution q are now considered independent from the parameters of distribution $p_\phi(\mathbf{x}|\mathbf{z})$. Equation (1) can be multiplied by the integral over the entire space of $q(\mathbf{z}|\mathbf{x})$, therefore, by one.

$$= \int_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log(p(x)) \quad (2)$$

Then, a series of simple transformations are done:

$$= \int_z q(z|x) \log\left(\frac{p(z, x)}{p(z|x)}\right) \quad (3)$$

$$= \int_z q(z|x) \log\left(\frac{p(z, x) q(z|x)}{q(z|x) p(z|x)}\right) \quad (4)$$

$$= \int_z q(z|x) \log\left(\frac{p(z, x)}{q(z|x)}\right) + \int_z q(z|x) \log\left(\frac{q(z|x)}{p(z|x)}\right) \quad (5)$$

$$= L^V + D_{KL}(q(z|x) || p(z|x)) \quad (6)$$

The first term L^V is called lower variational bound of the likelihood. Second term D_{KL} is the Kullback-Leibler divergence⁵ which measures the similarity of behaviour between two distributions. Here, it measures how well does $q(\mathbf{z}|\mathbf{x})$ approximates $p(\mathbf{z}|\mathbf{x})$. To maximize the likelihood it is necessary to maximize the lower variational bound. Therefore, lower variational bound is further analysed.

$$L^V = \int_z q(z|x) \log\left(\frac{p(z, x)}{q(z|x)}\right) \quad (7)$$

$$= \int_z q(z|x) \log\left(\frac{p(x|z)p(z)}{q(z|x)}\right) \quad (8)$$

$$= \int_z q(z|x) \log\left(\frac{p(z)}{q(z|x)}\right) + \int_z q(z|x) \log(p(x|z)) \quad (9)$$

$$= -D_{KL}(q(z|x) || p(z)) + \mathbb{E}_{q(z|x)}(\log(p(x|z))) \quad (10)$$

The left term in (10) is the Kullback-Leibler divergence which measures the similarity between $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$. Distribution $p(\mathbf{z})$ can be freely chosen but is usually normal distribution with zero mean and unit variance. In this context this term acts as a regularization term. Second term is the reconstruction quality of the autoencoder. It measures how well the approximation of $p(\mathbf{x}|\mathbf{z})$ produces data sample from the given latent state. Maximization of both of those terms is the goal of training the network.

5 https://en.wikipedia.org/wiki/Kullback-Leibler_divergence

First term $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ can be calculated using equation (11) (Dürr, 2016), since $q(\mathbf{z}|\mathbf{x})$ is a normal distribution with parameters $\boldsymbol{\mu}_z$, $\boldsymbol{\sigma}_z$ (vectors) produced by the encoder. Variable J is the size of the latent space, thus the dimension of mentioned vectors.

$$-D_{KL}(q(z|x)||p(z)) = \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_{z_j}^2) - \mu_{z_j}^2 - \sigma_{z_j}^2) \quad (11)$$

In order to calculate the second term sampling of latent variable is needed. After sampling the latent variable many times, the average of log probability over all the samples should be calculated to estimate the expectation. But usually, as (mini) batch learning is used, only a single sample is enough for training to work well. Therefore, only the log probability remains to be calculated using the sample taken from the latent space. The calculation can be done in the following way by using the expression for the probability distribution of a normal distribution $p(\mathbf{x}|\mathbf{z})$ (Dürr, 2016):

$$\log(p(x|z)) = - \sum_{j=1}^D \left(\frac{1}{2} \log(\sigma_{x_j}^2) + \frac{(x_j - \mu_{x_j})^2}{2\sigma_{x_j}^2} \right) \quad (12)$$

Parameters $\boldsymbol{\mu}_x$, $\boldsymbol{\sigma}_x$ (vectors) are produced by the decoder and D is the dimension of the data.

Both expressions on the right hand side in (11) and (12) need to be maximized in order maximize the likelihood of the data sample. Thus, the cost function is the sum of those two expressions multiplied by minus one:

$$-\frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_{z_j}^2) - \mu_{z_j}^2 - \sigma_{z_j}^2) + \sum_{j=1}^D \left(\frac{1}{2} \log(\sigma_{x_j}^2) + \frac{(x_j - \mu_{x_j})^2}{2\sigma_{x_j}^2} \right) \quad (13)$$

Function (13) is a cost function for a single data sample. Usually, for a single step of training, cost function is calculated for a batch of instances from the available training set. Total cost is then calculated as the average cost among the batch.

3.7.5. Reparametrization trick

As mentioned before, it is impossible to calculate the gradient of the cost function with respect to parameters of the encoder because we don't know how to calculate gradient of the random sampling function. To solve this problem the reparametrization trick is used (Kingma et al, 2014).

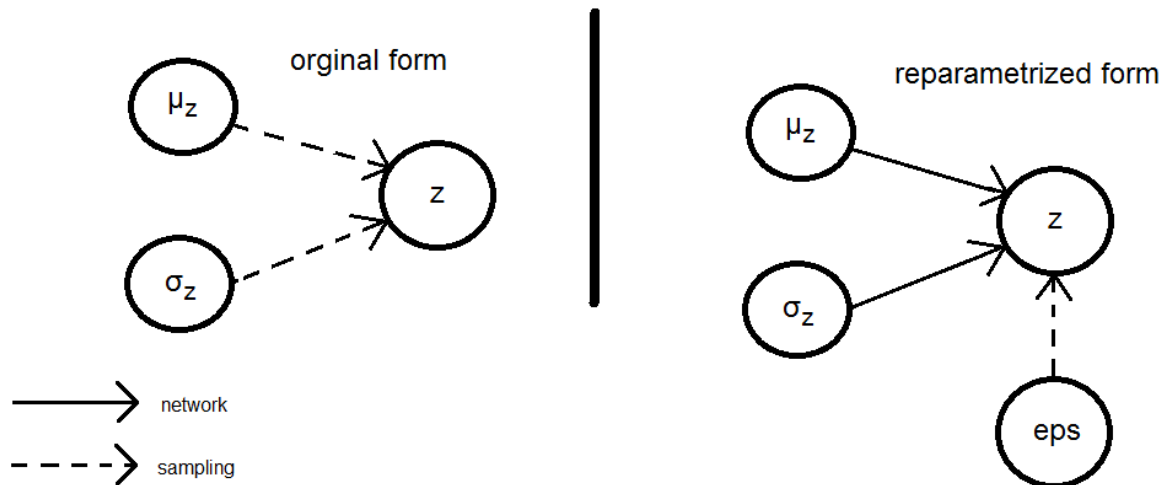


Figure 3.13. Reparametrization trick

Figure 3.13. demonstrates the idea. Instead of calculating latent representation z as the sample taken from the normal distribution, z is calculated as follows:

$$z = \mu_z + \sigma_z * \epsilon, \epsilon \sim \mathcal{N}(0, 1) \quad (14)$$

This way gradient of the cost function with respect to μ_z and σ_z can be calculated and therefore can be propagated further down the encoder network.

3.8. Denoising Autoencoder

Denoising autoencoder is a neural network whose goal is the same as explained in Section 3.6., generating output as similar as possible to the input. But at the input layer of the network corrupted version of the input is provided. Therefore, in order for the autoencoder to learn the correct output, it must focus on important aspects of the data. Figure 3.14. shows an example of a denoising autoencoder. As with the figures of variational autoencoder, this figure is just an example and number of neurons through layers is different in practice.

The figure shows how noise is added to each component of the input vector. Multiple types of noise can be used and are discussed below. After the input vector is corrupted, it is provided to the neural network which tries to learn how to reconstruct the original non-corrupted data. Error/cost function of the neural network is calculated with respect to difference between the output of the network and the original input vector (before adding noise).

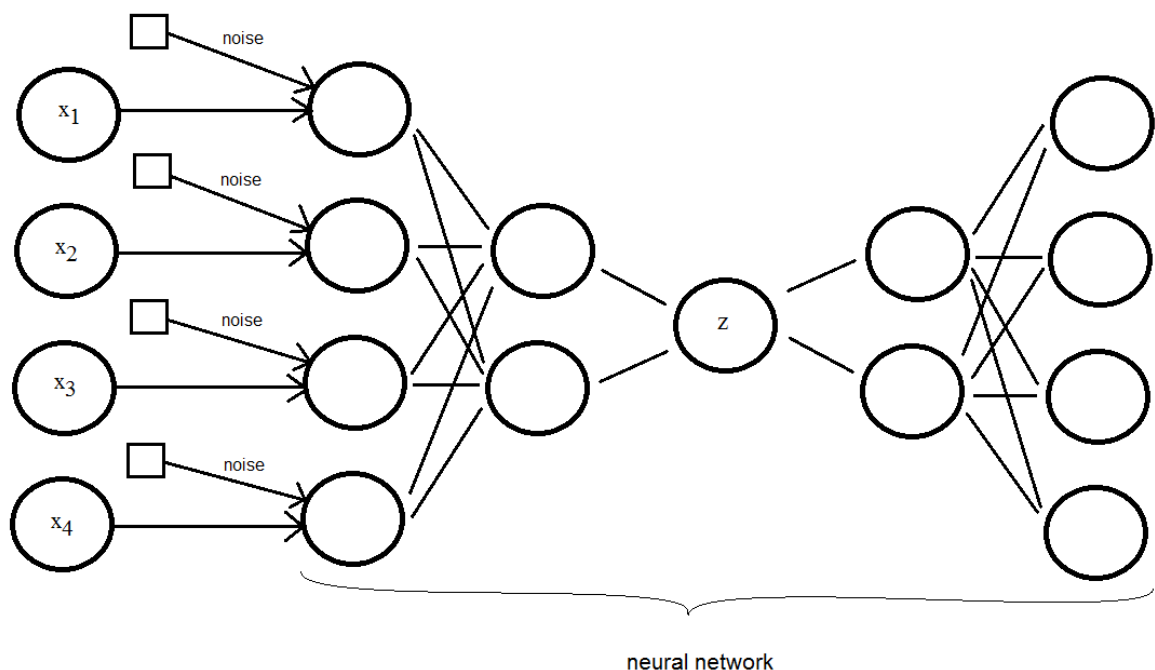


Figure 3.14. Denoising autoencoder

Role of the latent representation of data is assigned to a single hidden layer in the network, usually the one with the least amount of neurons (marked as z in

the figure). Vector whose components are the results of the neurons in that layer is considered to be the representation of the input example, thus providing the effect of dimensionality reduction.

In this thesis, cost function is mean squared error (15) where D stands for dimension of input and output vectors, \mathbf{x} is the original data vector and vector \mathbf{x}' is the output of the neural network.

$$cost = \frac{1}{D} \sum_{j=1}^D (x_j - x'_j)^2 \quad (15)$$

Three types of noise can be used as suggested in (Vincent et al, 2010):

- Gaussian noise: vector whose components are taken from a normal distribution with zero mean and variance σ is added to the original input vector
- Masking noise: fraction c of components (chosen randomly) of the original input vector is set to zero
- Salt-and-pepper noise: fraction c of components (chosen randomly) of the original input vector is set to minimum or maximum possible value (randomly chosen)

Gaussian noise has a hyperparameter σ (variance), while latter two have a hyperparameter c , often referred to as corruption level.

3.9. K-means clustering algorithm

K-means clustering, a typical example of unsupervised learning, is a popular clustering algorithm which divides data into a previously specified number of clusters. Goal of the algorithm is to place each data sample \mathbf{x}_i into one of k clusters in such a way that sum of square differences between data samples and their mean μ_i within each cluster is minimal. Given S_i denotes i -th set which

represents a cluster, algorithm minimizes the following function by changing the placement of data samples along clusters:

$$\sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|^2 \quad (16)$$

Algorithms works as follows (Arthur et al, 2007):

1. Using some technique, pick k points which represent means $\mu_1, \mu_2, \dots, \mu_k$ of k clusters
2. Assign each data sample \mathbf{x} into the cluster with the closest mean (using Euclidean distance)
3. Update the mean of each cluster by calculating it as the average of all points currently placed into the corresponding cluster
4. Go to step two if number of iterations is not yet reached

In step one, it is unspecified how starting means are chosen. There are many different approaches to how those are picked (Celebi et al, 2012). In this thesis k-means++ method is chosen as it is widely available and is known to perform well.

K-means++ works as follows (Arthur et al, 2007):

1. Select the mean of the first cluster uniformly at random from the data samples
2. For every data sample \mathbf{x} , calculate $D(\mathbf{x})$, the distance between \mathbf{x} and the closest already chosen cluster mean
3. Choose one data sample as mean of the next cluster, choosing a data sample \mathbf{x} with a probability proportional to square of $D(\mathbf{x})$
4. Go to step two until k means have been chosen for k clusters

Since k-means++ algorithm for choosing starting means is non-deterministic, results of k-means algorithm can vary. To reduce the impact of non-deterministic behaviour on results, k-means is often carried out a number of times and clusters which produce minimal value of the function (16) are taken as a final result.

By minimizing the function (16), k-means algorithm assumes clusters are spherical. Therefore it works badly on clusters which are elongated or shaped in some other way.

3.10. Spectral clustering

Spectral clustering is a clustering algorithm which has the same purpose as k-means, dividing the data into clusters in an unsupervised way. Just like with k-means, number of clusters is decided prior to running the algorithm. Unlike k-means, spectral clustering does not assume any specific cluster shape. Therefore it usually does better than k-means on datasets where possible cluster shapes are, for example, elongated, circular or other.

Spectral clustering algorithm can be divided into three steps:

1. Construct similarity graph from the given dataset
2. Reduce data dimensionality using the graph adjacency matrix
3. Cluster obtained vectors, usually using k-means

This shows that spectral clustering is a combination of a dimensionality reduction and clustering algorithm. Instead of clustering points in their original space, clustering is performed after applying dimensionality reduction, in hope that clustering in reduced space will be easier and clusters will emerge more naturally. Spectral clustering focuses on data connectivity when deciding on clusters, instead of assuming that a cluster is a set of points which occupy similar area.

The first task is the construction of a similarity graph from the dataset. Each vertex of the graph represents a single data sample from the dataset. There are a couple of popular ways of doing so, explained in (Luxburg, 2006). Perhaps the most usual choice are k-nearest neighbour graphs. They connect vertex v to vertex u if vertex u is among the k-nearest neighbours of v . The resulting graph is directed which is not the desired result. Therefore, to make the graph undirected, it is possible to just connect two points with an undirected edge if one of the vertices

is among the k -nearest neighbours of the other. This type of graph is called k -nearest neighbour graph. Another way is to connect two points only if they are both among the k -nearest neighbours of the other. This type of graph is called mutual k -nearest neighbour graph. To have more information about how close neighbours are to each other, each edge, in both graph types, is weighted by the similarity of adjacent vertices. To calculate the similarity of two vectors many functions are available. Example of a such function is the Gaussian similarity function (17). Parameters σ is used to manipulate the width of the neighbourhood.

$$\text{similarity}(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} \quad (17)$$

Another way of constructing a similarity graph is to assume a fully-connected graph and weigh each edge using similarity metric between adjacent vertices.

The second step is reducing data dimensionality using the adjacency matrix of the graph. The adjacency matrix is a matrix of size $N \times N$, where N is the number of vertices (therefore, number of data samples in the dataset), and its element w_{ij} is the weight of the edge between two vertices (zero if there is no edge). Goal of dimensionality reduction is to map each data sample \mathbf{x}_i to \mathbf{y}_i where \mathbf{y}_i has k components, k being the number of clusters, in such a way that minimizes function (18). The reason why dimension k is chosen is discussed later.

$$\sum_{i,j}^N w_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \quad (18)$$

Why minimization of this function is chosen can be seen through two cases. First, if w_{ij} is zero or very small that means data samples \mathbf{x}_i and \mathbf{x}_j are dissimilar. In that case, squared difference between \mathbf{y}_i and \mathbf{y}_j is considered irrelevant since function minimization has no significant effect on that difference. This is not ideal, since in the ideal case difference between \mathbf{y}_i and \mathbf{y}_j should be increased. But here, it is hoped that since minimizing the function won't as a rule minimize the difference between \mathbf{y}_i and \mathbf{y}_j when w_{ij} is zero or very small, this difference would usually be larger than the difference in the other case. And in that other case, w_{ij} is large

which means data samples \mathbf{x}_i and \mathbf{x}_j are similar. Minimizing this function results in minimizing the squared difference between \mathbf{y}_i and \mathbf{y}_j , which is what is wanted. If two data samples are similar, goal of the algorithm is to keep them similar while reducing the dimensionality of the data.

It can be shown that minimization of function (18) is equivalent to minimizing expression (19) over Y (Luxburg, 2006)

$$Tr(Y^T LY) \quad (19)$$

where Y is a $N \times k$ matrix in which row i represents \mathbf{y}_i and L is a graph Laplacian⁶ of the graph adjacency matrix and Tr denotes trace of the matrix (Luxburg, 2006). Solution for this minimization problem is a known result from graph theory which states columns of Y should correspond to the first k smallest eigenvectors of L (eigenvectors are compared according to their eigenvalues) (Luxburg, 2006). This is how mapping from \mathbf{x}_i to \mathbf{y}_i is obtained.

Now it can be discussed why dimension k is chosen for \mathbf{y}_i . There is another way of looking at step two and three of spectral clustering algorithm. Specifically, for dividing data into only two clusters we look to cut the similarity graph into two components in a way that sum of weights of edges which are removed is minimal and so that two components have approximately the same number of vertices. This approach is called finding the ratio cut of the graph (Luxburg, 2006). This problem is NP-hard but when relaxed in a certain way it becomes exactly the same problem as minimizing (18) and (19) with $k = 1$. In this special case, the second smallest eigenvector is used as a indicator vector from which it is easy to determine which point belongs to which cluster (Luxburg, 2006). Taking the first smallest k or $k-1$ eigenvectors is a heuristic for trying to extend this approach of considering indicator vectors for k clusters.

Third step is the one where actual clustering happens. Vectors \mathbf{y}_i which are low dimensional representations of \mathbf{x}_i can be clustered using any clustering algorithm. K-means is often used.

6 https://en.wikipedia.org/wiki/Laplacian_matrix

3.11. Overview of approach

Goal of this thesis is to cluster signals which represent coverage graphs of reads to check if any group of signals we are unaware of will emerge and to also create a classifier for known classes using only unsupervised learning. Full outline of our approach follows:

1. Generate coverage graphs from the available data (Section 2.1.)
2. Create a balanced dataset fit for deep learning (Section 2.2.)
3. Train an autoencoder using balanced dataset
4. Encode data samples using trained autoencoder to achieve dimensionality reduction
5. Cluster encoded data

First two steps are covered in Section 2. Before the third step, minor preprocessing is applied to data. First, since signal lengths are varying each signal is sampled in order to achieve equal signal length among the dataset. Then, data is normalized in order to bring all values to be within the range [0, 1]. This is done by the formula (20) which was applied to every component of each vector.

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (20)$$

In the fourth step, a variational (Section 3.7.) or denoising autoencoder (Section 3.8.) was trained. After that, trained autoencoder, specifically the encoder part, is used to encode data to lower dimensional space. Then in step five, k-means (Section 3.9.) or spectral clustering algorithm (Section 3.10.) is applied to encoded data and clusters are observed.

4. Results and discussion

All used models which produced the results presented in this section are publicly available at <https://github.com/jantomlj/Experimenting-with-signal-clustering>.

4.1. K-means along variational autoencoder

4.1.1. Results

Results presented in this section are achieved using variational autoencoder and k-means clustering algorithm. First, hyperparameters are organized in Table 4.1.

Hyperparameters are set to the combination of values that resulted in the best or the most interesting results but since the number of different combinations of hyperparameters is so high, only a small portion of combinations was tested. Dataset size is 21 173 signals.

Table 4.1. Hyperparameters

Hyperparameter	Value
Signal length	250
Batch size for training variational autoencoder	25
Number of training epochs for variational autoencoder	20
Activation function of the autoencoder neural network	ReLU
Gradient descent optimization algorithm	Adam ⁷
Learning rate of variational autoencoder	0.00005
Number of encoder layers	4
Number of decoder layers	4
Encoder layer sizes	[250, 200, 150, 100]
Decoder layer sizes	[100, 150, 200, 250]
Latent representation vector dimension	10
Initialization of kmeans	k-means++
Number of iterations of kmeans algorithm	300
Number of times kmeans was restarted	20
Number of clusters	4, 5

Examples which show reconstruction quality using variational autoencoder on unseen data are shown in Figure 4.1. Almost all data was used for training the variational autoencoder but 4 sets of 60 data samples (sets representing chimeric, left repeat, right repeat and regular signals) which were manually labelled and used as a test set.

⁷ https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam

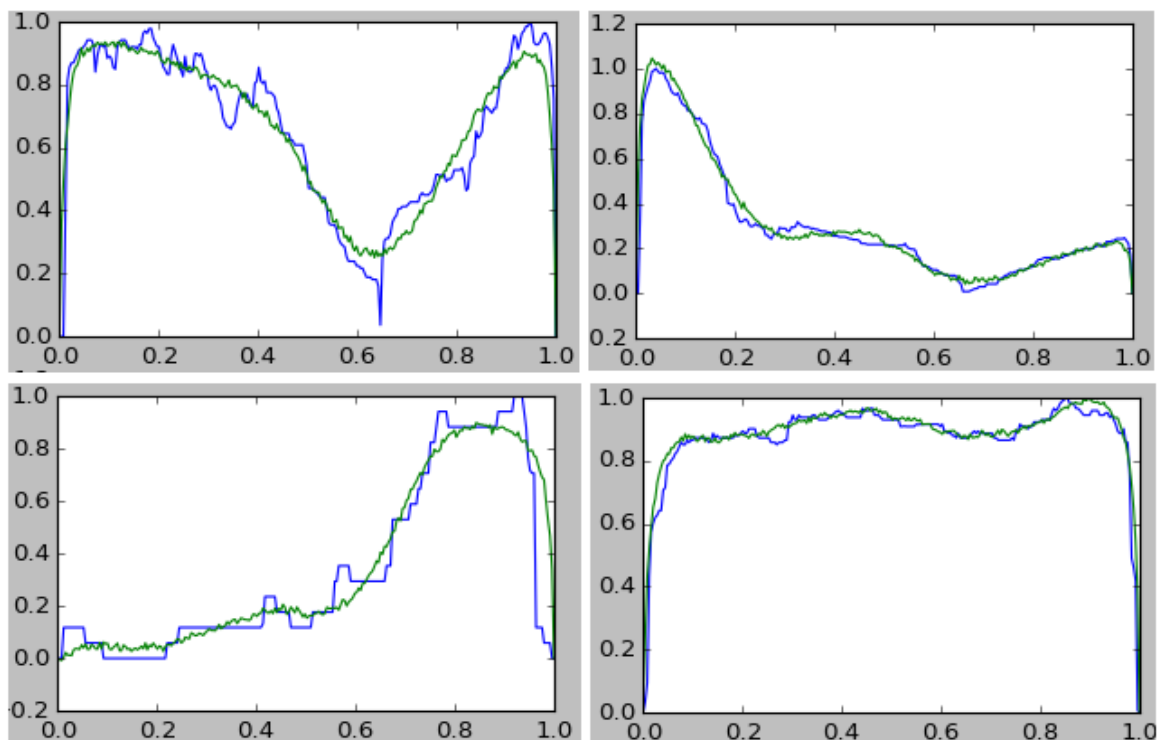


Figure 4.1. Reconstruction quality using variational autoencoder (blue line – signal, green line – reconstruction)

Same training and test set split was used for clustering as was for training the variational autoencoder. K-means model that was the result of clustering the data was applied to the test set to see how well do its clusters correspond to 4 classes we are aware exist. K-means model places an unseen data sample to the cluster with the mean closest to the data sample. We distinguish two cases, the one where data was divided into four clusters and second where data was divided into five clusters. For each case the process of training the variational autoencoder and clustering the data was run five times.

For the first case, confusion matrix obtained by classifying data samples from the test set is presented in Table 4.2.a). The most representative example of the matrix is chosen after running the algorithm five times. Columns represent chimeric, left repeat, right repeat and regular classes of signals in that order from left to right. Prior to running the algorithm we don't know which cluster will correspond to which class. Therefore matching the cluster to class is done after

seeing how data samples from the test set are distributed among clusters. Each cluster is named after the class which is most represented in it. This is done because that is the way classes would be decided if this k-means model was to be used as a classifier in the future and that is because this choice is the one which produces best results.

Table 4.2. Confusion matrices obtained by dividing the data into a) four clusters and b) five clusters

		a)				b)					
		True labels				True labels					
		ch	lr	rr	re						
Predicted labels	ch	7	0	6	2	Predicted labels	ch	41	1	9	0
	lr	17	55	1	0		lr	7	54	0	0
	rr	12	1	50	0		rr	2	0	48	0
	re	24	4	3	58		re	8	5	0	56
							2	0	3	4	

Representative example of the confusion matrix for the second case is shown in Table 4.2.b). Test samples are distributed among five clusters. Four clusters are named in the same manner as in Table 4.2.a). Remaining fifth cluster isn't named and what it represents is discussed later.

It is possible to calculate accuracy and F-score⁸ of the k-means model which is being used as a classifier for four known classes only. Macro-averaged F_1 score is used (Asch, 2013). Samples classified as members of the fifth, unlabelled cluster affect the recall, but not the precision of the classifier. Averages and standard deviations over five runs are presented in Table 4.3.

8 https://en.wikipedia.org/wiki/F1_score

Table 4.3. Scores

	Four clusters	Five clusters
Accuracy	0.74 ± 0.09	0.82 ± 0.05
Macro F₁ score	0.6 ± 0.1	0.82 ± 0.06

Data visualisation algorithm t-SNE was used to create two dimensional visualisation of the data. The t-SNE algorithm is a dimensionality reduction algorithm which is primarily used to map data to two or three dimensional space (Maaten et al, 2008). Following figures were generated as follows. Encoded (using the encoder of the trained variational autoencoder) part of training or test set was provided to t-SNE algorithm which mapped each data point to two dimensional space. Points were then labelled according to the cluster k-means algorithm placed them into.

In the first case, shown in Figure 4.2., k-means divided data into 4 clusters. The graph on the left hand side is obtained by providing an encoded sample from the training data to the t-SNE algorithm. The graph on the right hand side is obtained by using 240 encoded examples of the test set as input to the t-SNE algorithm.

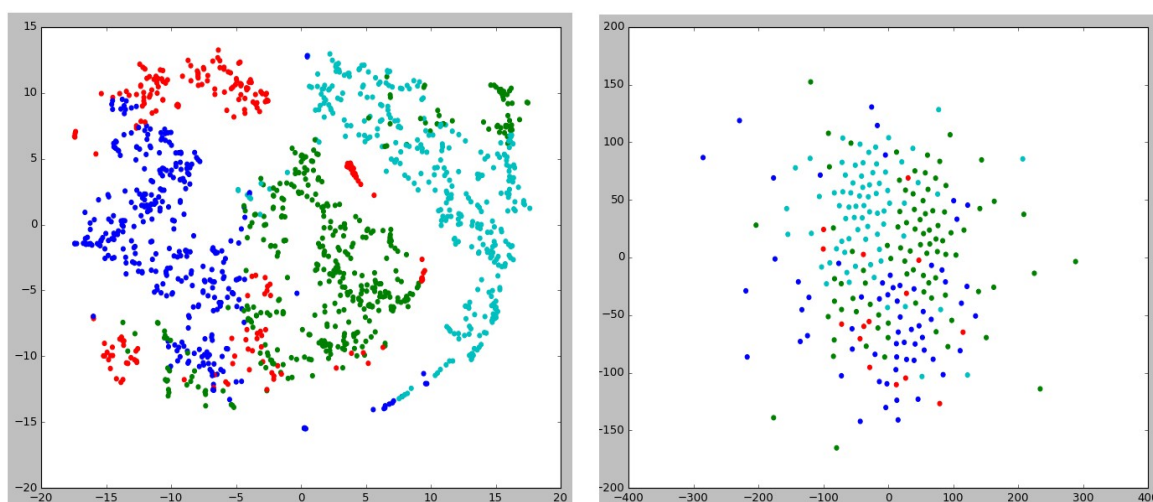


Figure 4.2. t-SNE of four clusters (blue – right repeat, green – regular, red – chimeric, cyan – left repeat)

Same procedure was done in the second case where five clusters are assumed. Results are shown in Figure 4.3.

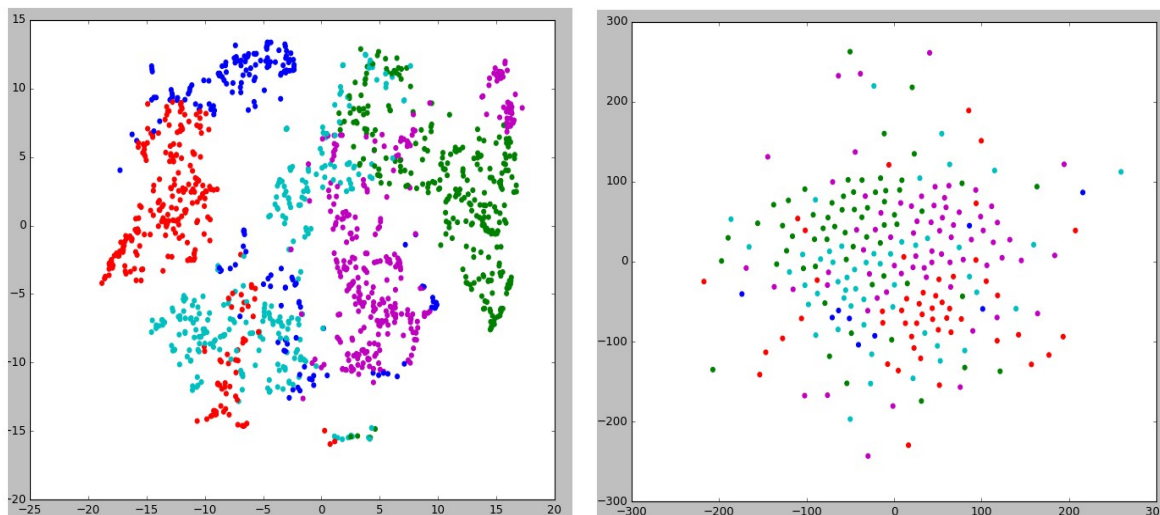


Figure 4.3. t-SNE of five clusters (blue – unknown, green – left repeat, red – right repeat, cyan - chimeric, magenta - regular)

4.1.2. Discussion

In Figure 4.1. it can be seen that the general shape of the signal is present in the reconstruction, but details are not perfectly preserved. This is expected in some amount due to information loss as a result of compression. Left repeat, right repeat and regular signals seem to be reconstructed better than chimeric signals. Chimeric signals are often characterized by a very steep drop and rise of the signal, but that steepness disappears in the reconstruction. As can be seen in confusion matrices in Table 4.2., that results in worse results when classifying chimeric signals.

Confusion matrices show that cluster which represents regular signals is usually of very high quality, meaning in this case, that it contains most of the regular signals from the data and almost nothing else. This can be useful if this was used as a classifier which extracts only regular signals, for they are most useful and reliable when reconstructing genomes.

When comparing confusion matrices along with accuracy and F-score between clustering into four and five clusters it can be observed that k-means model acts as a better classifier in the latter case. In the case of four clusters performance is significantly worse than in the case of five clusters. It can be argued that that is the case because there is a fifth type of signal in the data so the clustering algorithm works worse when it is made to deal with only four clusters. Furthermore, confusion matrix with five clusters could also be viewed from that perspective, since there is a very small number of samples put into that fifth cluster and we know that the test set is made up of only obvious cases of the four known clusters. To investigate, signals from the training set which were put into that fifth cluster were thoroughly examined. But it turned out to be very difficult to characterize this cluster. There are many very different signals in it and a new special form of signal isn't observed. Still it can be seen that signal curve usually has larger values on the right hand side, but it is difficult to detect if those signals are right repeat signals. Some definitely are, but most are ambiguous as can often be the case for these signals when trying to classify them using the naked eye. There is also a smaller but significant number of chimeric signals which have a steep drop and rise on the far left hand side and then the signal gradually rises on the right side. So this cluster can perhaps be characterized as containing mainly questionable signals which have larger values on the right hand side. That is probably the reason why scores are better if five clusters are used, since the signals in the fifth cluster are difficult to classify among four known classes. Two examples are shown in Figure 4.4.

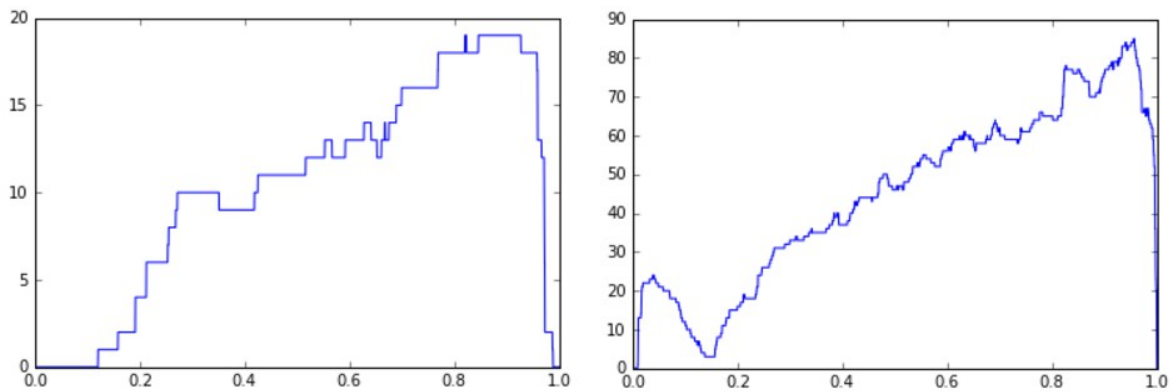


Figure 4.4. Signals from the fifth cluster

t-SNE in Figure 4.2. which is done with four clusters shows a very good separation of three classes on training data and on test data as well. Those are left repeat, right repeat and regular signals. But chimeric signals are very scattered. Most interesting about the Figure 4.3., which shows t-SNE done after clustering into five clusters, are blue dots of the fifth cluster. They seem to be a combination of right repeat, chimeric and regular signals (in case of the four clusters assumption) which corresponds well to what was observed in the previous paragraph.

4.2. K-means along denoising autoencoder

4.2.1. Results

All comments and procedures from Section 4.1.1. apply here with the distinction that denoising autoencoder is used instead of variational. Hyperparameters are presented in Table 4.4.

Table 4.4. Hyperparameters

Hyperparameter	Value
Signal length	250
Batch size for training denoising autoencoder	20
Number of training epochs for denoising autoencoder	25
Activation function of the autoencoder neural network	ReLU
Gradient descent optimization algorithm	Adam
Learning rate of denoising autoencoder	0.00005
Noise type	Masked noise
Corruption level	0.2
Number of encoder layers	4
Number of decoder layers	4
Encoder layer sizes	[250, 220, 200, 150]
Decoder layer sizes	[150, 200, 220, 250]
Latent representation vector dimension	12
Initialization of kmeans	k-means++
Number of iterations of kmeans algorithm	300
Number of times kmeans was restarted	20
Number of clusters	4, 5, 6

Examples which show reconstruction quality using variational autoencoder on unseen data are shown in Figure 4.5.

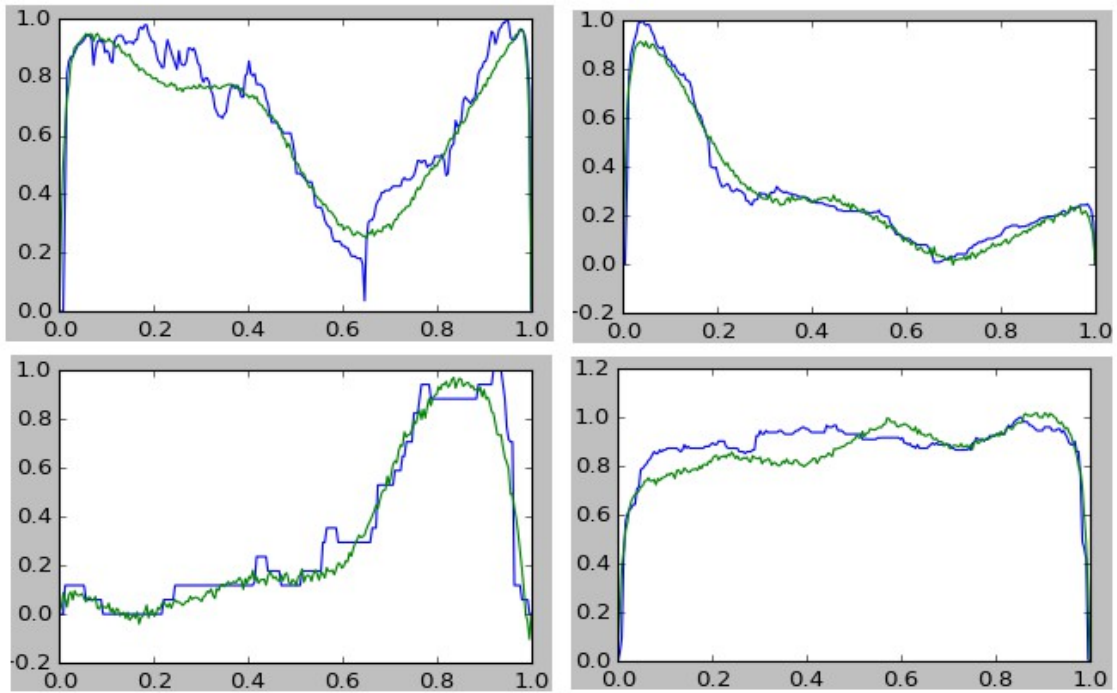


Figure 4.5. Reconstruction quality using denoising autoencoder (blue line – signal, green line – reconstruction)

Representative examples of confusion matrices after five runs are shown in Table 4.5.

Table 4.5. Confusion matrices obtained by dividing the data into a) four clusters, b) five clusters and c) six clusters

a)

		True labels			
		ch	lr	rr	re
Predicted labels	ch	49	7	5	3
	lr	1	51	1	0
	rr	8	1	54	0
	re	2	1	0	57

b)

		True labels			
		ch	lr	rr	re
Predicted labels		49	7	5	2
		2	50	0	0
		2	1	47	0
		2	2	0	57
		5	0	8	1

c)

		True labels			
		ch	lr	rr	re
Predicted labels		43	0	6	0
		0	40	1	0
		2	0	44	0
		1	0	0	59
		8	19	0	1
		6	1	9	0

Accuracy and F-score averaged over five runs are presented in Table 4.6.

Table 4.6. Scores

	Four clusters	Five clusters	Six clusters
Accuracy	0.85 ± 0.04	0.81 ± 0.07	0.7 ± 0.1
Macro F₁ score	0.83 ± 0.05	0.8 ± 0.1	0.7 ± 0.1

t-SNE graphs are shown in Figure 4.6. (four clusters), Figure 4.7. (five clusters) and Figure 4.8. (six clusters).

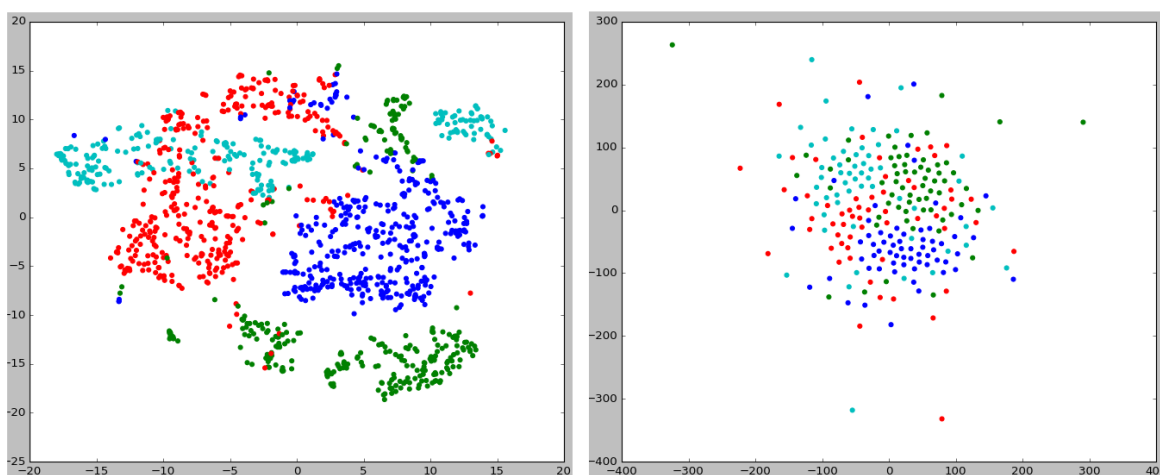


Figure 4.6. t-SNE of four clusters (blue – right repeat, green – regular, red – chimeric, cyan – left repeat)

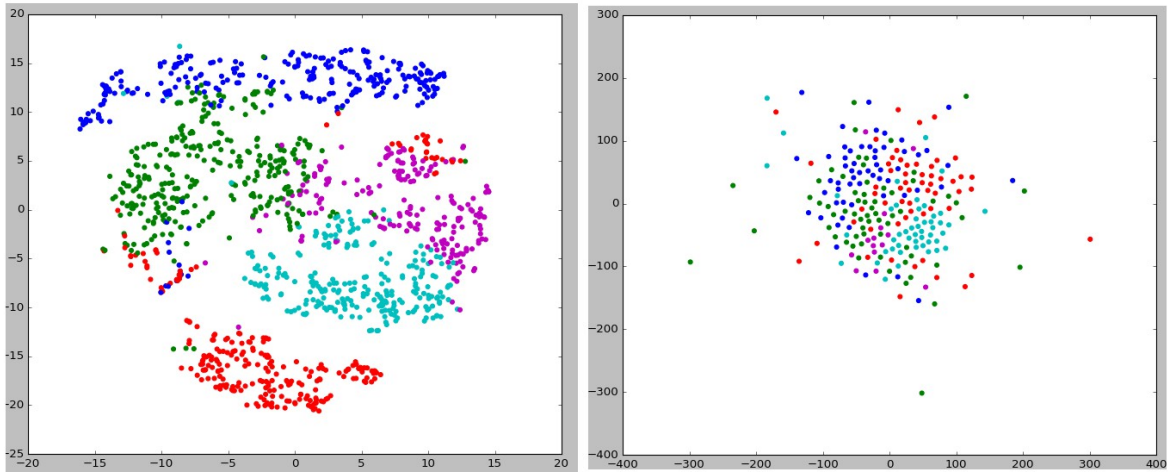


Figure 4.7. t-SNE of five clusters (blue – left repeat, green – chimeric, red – regular, cyan – right repeat, magenta - unknown)

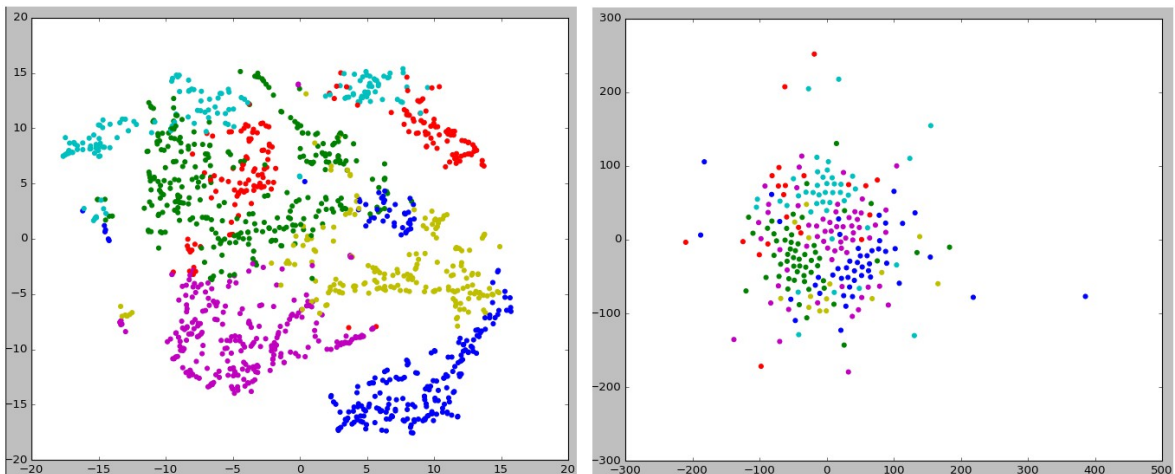


Figure 4.8. t-SNE of six clusters (blue – right repeat, green – chimeric, red – unknown, cyan – left repeat, magenta – regular, yellow - unknown)

4.2.2. Discussion

Signal reconstruction done by the denoising autoencoder presented in Figure 4.5. shows similar characteristics as was the case with variational autoencoder. General shape of the signal is present but lacks details. Same conclusions as in

4.1.2. can be made. It can be observed that reconstruction quality is slightly lower than in the case of variational autoencoder.

Furthermore, same conclusions as in 4.1.2. can be made about the high quality of the cluster which represents regular signals.

Looking at the relation between classifications scores and number of clusters it can be observed that scores drop as number of clusters is increased. It seems that four clusters are the most natural option here as they correspond well to four clusters we know about. After looking closely at the fifth and the sixth cluster it is observed that fifth cluster is very similar to the one found in Section 4.1. and sixth cluster can be considered a mirror image of the fifth cluster. The most likely classification of samples in the fifth cluster would be right repeat and the most likely classification of samples in the sixth cluster would be left repeat, but it is more difficult to conclusively decide which type of signals they are than it is for the samples in other clusters. Proposed names for the fifth and sixth cluster are soft right repeat and soft left repeat respectively. Typical examples from left repeat, soft left repeat, soft right repeat and right repeat clusters are shown in Figure 4.9.

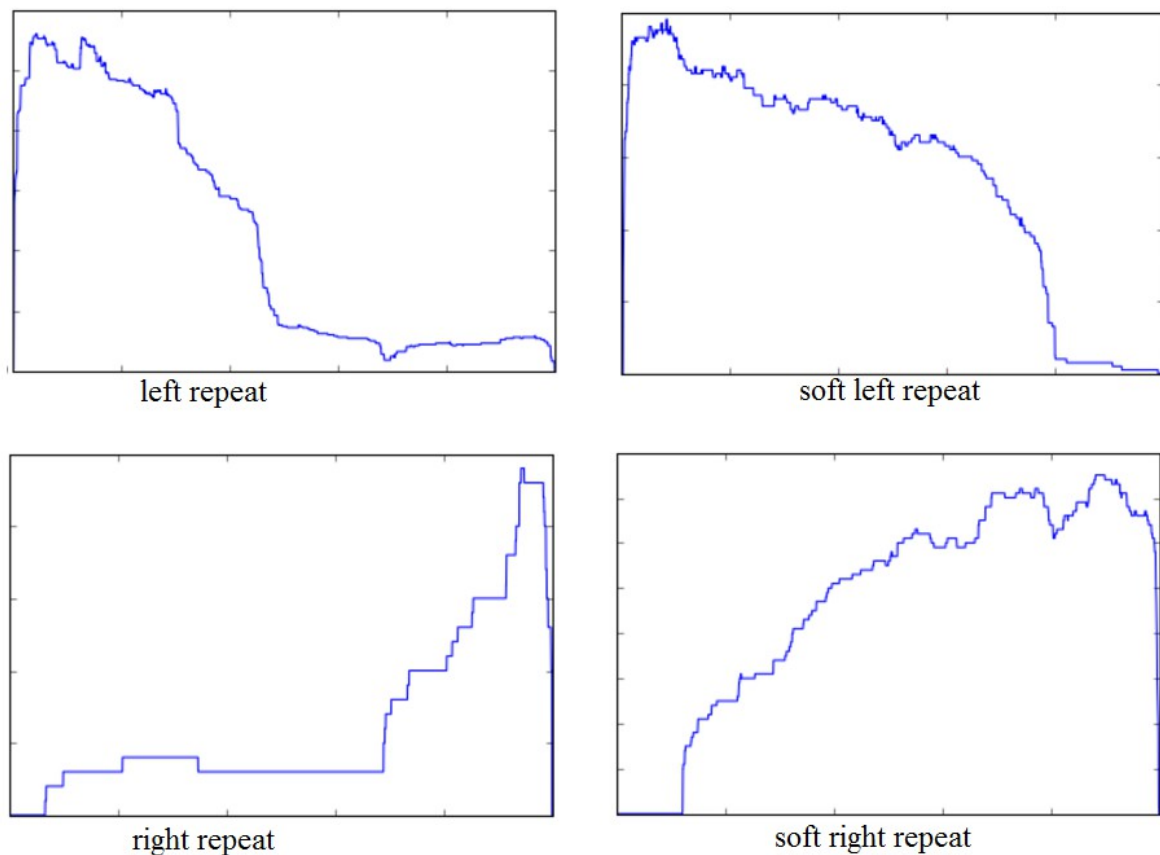


Figure 4.9. Variations of repeat clusters

t-SNE graphs reflect that dividing the data into six clusters appears messy and clusters look very scattered in that case. Cases of four and five clusters appear to be more coherent.

4.3. Spectral clustering along variational autoencoder

4.3.1. Results

All comments and procedures from Section 4.1.1. apply here with the distinction that spectral clustering is used instead of k-means. Specific differences in the details of approach are emphasized below. Hyperparameters are presented in Table 4.7.

Table 4.7. Hyperparameters

Hyperparameter	Value
Signal length	250
Batch size for training variational autoencoder	20
Number of training epochs for variational autoencoder	25
Activation function of the autoencoder neural network	ReLU
Gradient descent optimization algorithm	Adam
Learning rate of variational autoencoder	0.00005
Number of encoder layers	4
Number of decoder layers	4
Encoder layer sizes	[250, 220, 200, 150]
Decoder layer sizes	[150, 200, 220, 250]
Latent representation vector dimension	12
Graph construction algorithm	k-nearest neighbours
Number of neighbours k	10
Algorithm used for final clustering step	k-means
Number of times k-means was restarted	10
Number of clusters	3, 4

Examples which show reconstruction quality using variational autoencoder on unseen data are shown in Figure 4.10.

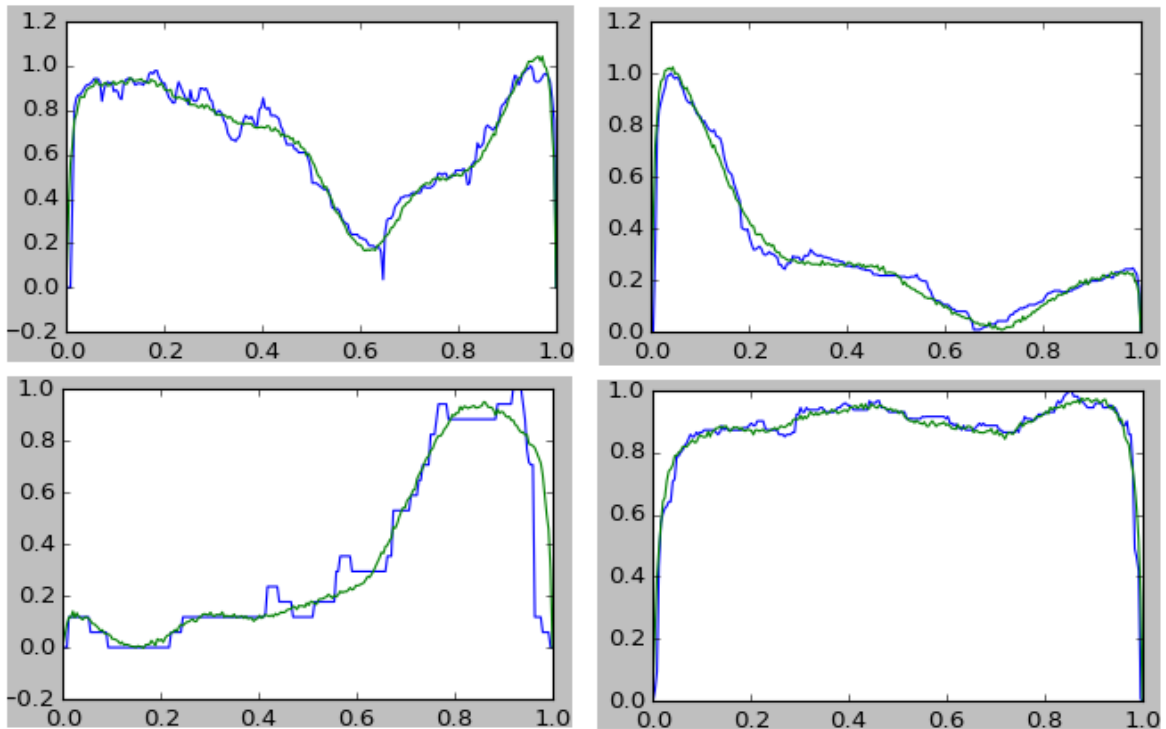


Figure 4.10. Reconstruction quality using variational autoencoder (blue line – signal, green line – reconstruction)

Representative examples of confusion matrices after five runs are shown in Table 4.8. Spectral clustering can not first cluster the training data and then place samples from the test set into best fitted clusters as k-means can, it can only form clusters and distribute all input data among them. For that reason test samples were merged with the training set and clustered together. Then labels of the samples from the test set were considered to form confusion matrices.

Table 4.8. Confusion matrices obtained by dividing data into a) three clusters and b) four clusters

		a)				b)			
		True labels				True labels			
		ch	lr	rr	re	ch	lr	rr	re
Predicted	lr	39	60	1	1				
	rr	17	0	58	0	lr	37	60	0
	re	4	0	1	59	rr	23	0	60
						re	0	0	0
									52

Accuracy and F-score averaged over five runs are presented in Table 4.9.

Table 4.9. Scores

	Three clusters	Four clusters
Accuracy	0.73 ± 0.04	0.73 ± 0.05
Macro F_1 score	0.61 ± 0.08	0.62 ± 0.08

t-SNE graphs are shown in Figure 4.11. (three clusters) and Figure 4.12. (four clusters).

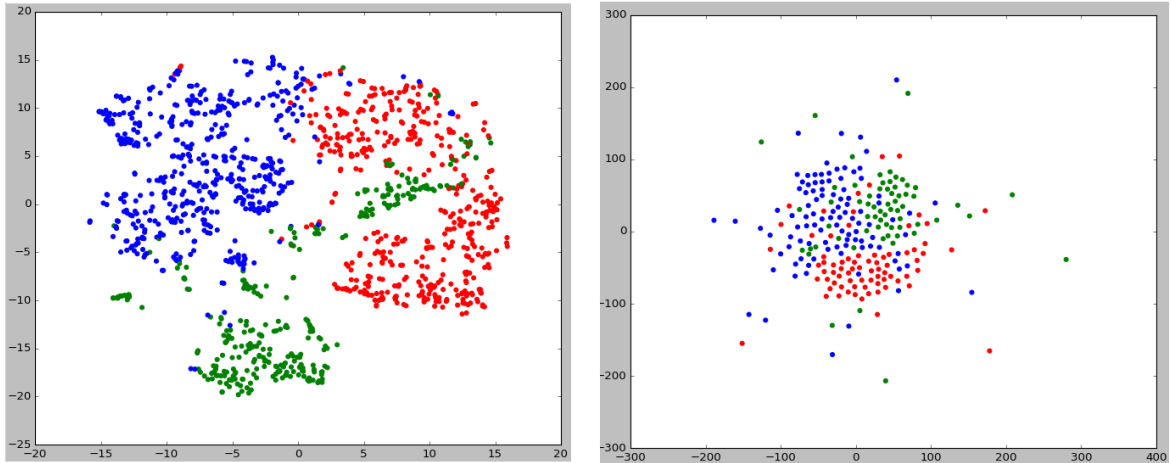


Figure 4.11. t-SNE of three clusters (blue – left repeat, green – regular, red – right repeat)

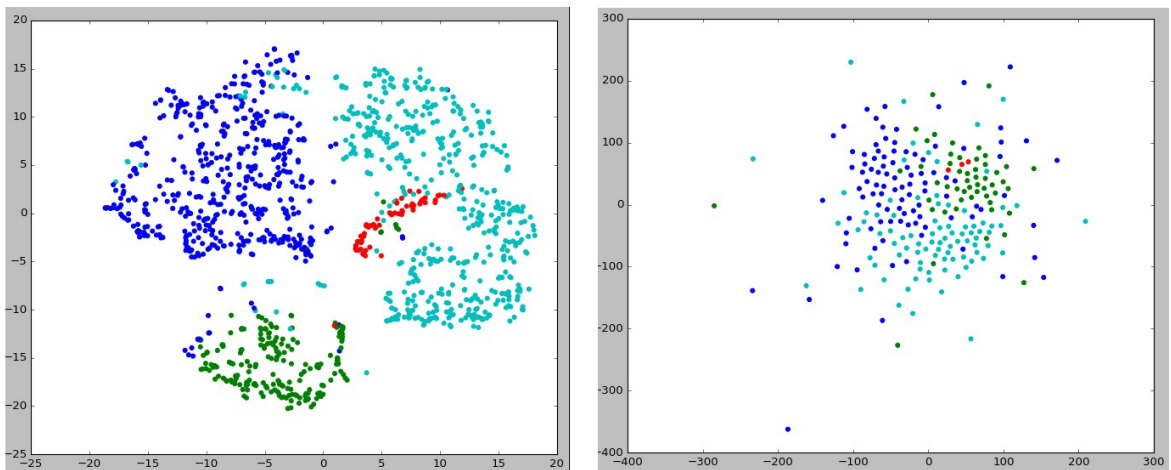


Figure 4.12. t-SNE of four clusters (blue – left repeat, green – regular, red – unknown, cyan – right repeat)

4.3.2. Discussion

Reconstruction quality is similar to the one in 4.1., but since different parameters for variational autoencoder are used it is not the same. Same conclusions about reconstruction quality as in 4.1.2. can be made.

By first running spectral clustering algorithm using four clusters it was obvious that the algorithm only saw three clusters. Fourth cluster was very small and specific so very few samples from the test set were placed into it. The next step was running the algorithm using only three clusters to see how well separated they are and what they represent. The best achieved accuracy and F-score are significantly lower than in Section 4.1. and Section 4.2. That is the result of found clusters not matching our idea of the four known clusters.

After manually looking at the three clusters, no connections were found between the shape of the signal and the cluster the signal belongs to. Clusters can almost seem random. But t-SNE graph shows very clear separation between the three clusters. This leads to the conclusion that there is some solid criteria by which they are distributed into clusters, it is just unknown to us. Results from the confusion matrices suggest that three clusters are left repeat, right repeat and regular, with chimeric signals being divided between the left repeat and the right repeat clusters. But manual inspection of clusters does not confirm this. For instance, cluster which should represent regular signals contains a lot of chimeric, left repeat and right repeat signals from the training set, unlike the confusion matrix would lead to believe. The only possible conclusion here is that applying spectral clustering dimension reduction algorithm to already compressed data produces unexpected results. Patterns found in the data are not connected to the shape of the signal, at least not in a way that we can easily observe.

Spectral clustering algorithm was likewise run along denoising autoencoder, but results were very similar, meaning three clusters were found but connection between signal shapes and clusters was not found. Therefore, same conclusions apply.

4.4. Comparison

Specific clusters found using different algorithms were discussed separately. Here, the aim is to compare classification scores between different clustering models used as classifiers. Table 4.10. shows all gathered results.

Table 4.10. Classification results

		3 clusters	4 clusters	5 clusters	6 clusters
Va. AE.	K-means +				
	accuracy	-	0.74 ± 0.09	0.82 ± 0.05	-
	Macro F₁ score	-	0.6 ± 0.1	0.82 ± 0.06	-
De. AE.	K-means +				
	accuracy	-	0.85 ± 0.04	0.81 ± 0.07	0.7 ± 0.1
	Macro F₁ score	-	0.83 ± 0.05	0.8 ± 0.1	0.7 ± 0.1
Va. AE.	Spectral +				
	accuracy	0.73 ± 0.04	0.73 ± 0.05	-	-
	Macro F₁ score	0.61 ± 0.08	0.62 ± 0.08	-	-

Note that number of clusters affects the potential process of classification greatly, assuming the goal is to classify each signal into one of the four known classes. If only three clusters are used there are only three classes as options for the model, which means no data will be classified into the fourth class. On the other hand, if five or six clusters are used, of which we consider four to represent our four known classes of signals, data classified into fifth and sixth cluster will be, in fact, left unclassified.

Looking at the Table 4.10. we can single out two best approaches. Using the combination of the denoising autoencoder along k-means algorithm with four clusters produces scores of about 85% accuracy and 0.83 F-score. Since exactly four clusters are used, all data can automatically be classified. Using the combination of variational autoencoder along k-means algorithm with five clusters

produces similar scores. But here, data classified into the fifth cluster would stay unclassified and some other method or manual classification would need to be applied to classify the full dataset.

4.5. Application

Combination of the denoising autoencoder along k-means algorithm was used to help generating an overlap graph for four bacteria from NCTC bacteria collection⁹. Again, graphmap tool was used to generate overlaps from which coverage graphs were calculated. Our trained model classified all coverage graphs of these bacteria as regular, left repeat, right repeat or chimeric. Chimeric signals and overlaps between left and right repeat signals were filtered out before generating the overlap graph. The results obtained by using this technique are compared to the results obtained by using all coverage graphs in the Table 4.11.

Table 4.11. Comparison of the qualities of overlap graphs

bacteria		NCTC74	NCTC86	NCTC129	NCTC204
Using all coverage graphs	number of contigs ¹⁰	21	126	36	39
	NG50 ¹¹	546k	75k	270k	320k
Using coverage graphs selected by our model	number of contigs	20	54	12	38
	NG50	553k	216k	1132k	271k

Since the number of contigs can be understood as a measure of the complexity of the overlap graph, it can be observed that our model, in some cases, reduces that complexity, which can make the following steps of the genome assembly process simpler.

9 <https://www.phe-culturecollections.org.uk/collections/nctc.aspx>

10 <https://en.wikipedia.org/wiki/Contig>

11 https://en.wikipedia.org/wiki/N50,_L50,_and_related_statistics#NG50

Conclusion

This thesis deals with signals, i.e. coverage graphs, generated using overlaps of reads. Shape of the coverage graph of a read reflects how that read was created and thereby which type of a read it is. It is assumed we are aware of four different types of reads and, by extension, coverage graphs. Thesis explores how do clusters of signals found by different approaches correspond to our knowledge of types of reads and how well can signals be classified using only unsupervised learning.

The approach used was to first compress signals into a lower dimensional space and then apply a clustering algorithm to those compressed signals. Two types of autoencoders, variational autoencoder and denoising autoencoder, were used as feature extractors which compressed signals. Clustering was performed by two standard unsupervised learning algorithms: k-means and spectral clustering.

Three combinations of algorithms specified above were used and in two cases found clusters correspond relatively well to the four assumed clusters. But one or two potential additional clusters can be observed. In the third case clusters found were completely unrelated to the four assumed clusters and a simple criteria for dividing signals into those clusters was not found. Quality of classification achieved shows that decent results can be obtained but it is questionable whether these results are good enough to be used in genome reconstruction algorithms.

Further work can be focused on researching whether additional clusters found could signalize a specific type of read yet unspecified. In the above mentioned third case, semantics of found clusters can be studied further. In order to achieve better classification results, supervised or semi-supervised learning should be an improvement over the conducted unsupervised approach.

Bibliography

- Šikić M., Domazet-Lošo M., "*Bioinformatika*", 2013,
https://www.fer.unizg.hr/_download/repository/bioinformatika_skripta_v1.2.pdf
- Metzker M.L., "*Emerging technologies in DNA sequencing*", *Genome Res.* 15: 1767–1776, 2005
- Shendure J., Hanlee J., "Next generation DNA sequencing", *Nat. Biotechnol.* 26: 1135–1145, 2008
- Sović I., Šikić M., Wilm A., Fenlon S.N., Chen S., Nagarajan N., "*Fast and sensitive mapping of nanopore sequencing reads with GraphMap*", *Nature Communications* 7, 2015
- Dalyac A., Shanahan M., Kelly J., "*Tackling Class Imbalance with Deep Convolutional Neural Networks*", 2014
- Russell S.J., Norvig P., "*Artificial Intelligence: A Modern Approach*", 2009
- Dürr O., "Introduction to variational autoencoders", 2016,
<https://home.zhaw.ch/~dueo/bbs/files/vae.pdf>
- Kingma D.P., Welling M., "*Auto-Encoding Variational Bayes*", 2014
- Vincent P., Larochelle H., Lajoie I., Bengio Y., Manzagol P.A., "*Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*", *The Journal of Machine Learning Research* Vol. 11: 3371-3408, 2010
- Celebi M. E., Kingravi H. A., Vela P. A., "*A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm*", *Expert Systems with Applications* 40: 200-210, 2012

Arthur D., Vassilvitskii S., “k-means: The advantages of careful seeding”, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007

Luxburg U., “*A Tutorial on Spectral Clustering*”, Max Planck Institute for Biological Cybernetics, 2006

Van der Maaten L.J.P., Hinton G.E., “*Visualising High-Dimensional Data Using t-SNE*”, Journal of Machine Learning Research 9: 2579-2605, 2008

Van Asch V., “*Macro- and micro-averaged evaluation measures*”, <http://www.clips.uantwerpen.be/~vincent/pdf/microaverage.pdf> , 2013

Identification of 1D-Signal Types Using Unsupervised Deep Learning

Abstract

During de novo genome assembly process, certain types of sequenced reads can cause problems during genome reconstruction. Goal of this thesis is to learn more about possible types of reads and classification of those reads using unsupervised learning. Coverage graphs of reads are generated using read overlaps and those coverage graphs are further analysed. Autoencoder is used to compress the signal, i.e. the coverage graph, and clustering algorithm is then applied to the compressed data. Variational and denoising autoencoders along with k-means and spectral clustering algorithms are used. Visualisation of found clusters is performed along with semantic analysis. Signal classification quality using unsupervised learning is estimated.

Keywords: bioinformatics, unsupervised learning, deep learning, autoencoders

Identifikacija tipova 1D-signalna pomoću nenadziranog dubokog učenja

Sažetak

Tijekom de novo procesa sastavljanja genoma određene tipovi očitavanja mogu uzrokovati probleme prilikom rekonstrukcije genoma. Cilj ovog rada je naučiti više o mogućim tipovima očitavanja te kako klasificirati očitavanja koristeći nenadzirano učenje. Koristeći preklapanja među očitanjima, za svako očitavanje generiran je graf pokrivenosti i oni su dalje analizirani. Autoenkoder je korišten s ciljem sažimanja signala, tj. grafa pokrivenosti, i zatim je nad sažetim prikazom podataka obavljeno grupiranje. Korišteni su varijacijski i *denoising* autoencoder te algoritmi grupiranja *k-means* i spektralno grupiranje. Nađene grupe signala su vizualizirane te je provedena semantička analiza grupa. Procjenjena je kvaliteta klasifikacije signala korištenjem nenadziranog učenja.

Ključne riječi: bioinformatika, nenadzirano učenje, duboko učenje, autoenkoderi