

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND
COMPUTING

MASTER THESIS No. 1414

**Classification of 1D-Signal Types Using
Semi-Supervised Deep Learning**

Tomislav Šebrek

Zagreb, June 2017.

Zagreb, 3 March 2017

MASTER THESIS ASSIGNMENT No. 1414

Student: **Tomislav Šebrek (0036473583)**
Study: **Computing**
Profile: **Computer Science**

Title: **Classification of 1D-Signal Types Using Semi-Supervised Deep Learning**

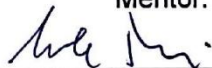
Description:

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 classification of these signal types based on many unlabelled and a small number of labelled learning data. Implement a network that is trained by minimizing both reconstruction and classification loss. Minimize the reconstruction loss by an autoencoder network and classification loss by a feed-forward deep network. These two networks should share frontend layers. Perform visualization of obtained representations using t-Distributed Stochastic Neighbor Embedding (t-SNE) and evaluate achieved results using manually labelled signals. Implement method using TensorFlow library. The code should be documented and hosted on a publicly available Github repository.

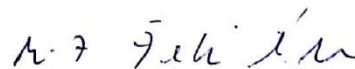
Issue date: 10 March 2017
Submission date: 29 June 2017

Mentor:



Associate Professor Mile Šikić, PhD

Committee Chair:



Full Professor Siniša Srblić, PhD

Committee Secretary:



Assistant Professor Tomislav Hrkać, PhD

Zagreb, 3. ožujka 2017.

DIPLOMSKI ZADATAK br. 1414

Pristupnik: **Tomislav Šebrek (0036473583)**
Studij: Računarstvo
Profil: Računarska znanost

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

Opis zadatka:

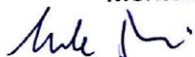
Tijekom de novo sastavljanja genoma, važan korak je pronalaženje preklapanja među očitanjima. Pogrešno detektirana preklapanja onemogućuju ispravno sastavljanje originalne sekvence. Za svako očitavanje moguće je izgraditi graf pokrivenosti svake baze. Takav graf daje nam 1D-signal koji se može dodatno 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 razviti metodu za klasifikaciju tipova signala temeljenu na velikom broju neoznačenih i malom broju označenih uzoraka za učenje. Implementirati neuronsku mrežu koja će učiti tako da minimizira funkciju gubitka i kod klasifikacije i kod rekonstrukcije. Gubitak kod rekonstrukcije minimizirati pomoću mreže tipa autoenkoder, dok gubitak kod klasifikacije minimizirati koristeći unaprijednu mrežu (engl. feed-forward). Mreže bi trebale dijeliti prednje slojeve. Dobivene rezultate vizualizirati pomoću metode t-SNE (using t-Distributed Stochastic Neighbor Embedding) i procijeniti kvalitetu usporedbom s ručno označenim uzorcima. Metodu implementirati 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

I would like to thank my family for all the support they gave me.

Also, many thanks to my mentor Mile Šikić for all the help over the past years.

Further on, thanks to Robert Vaser for the help with the assembly process and Jan Tomljanović for the help with the dataset generation.

Table of contents

| | |
|---|----|
| 1. Introduction | 1 |
| 2. Types of reads | 2 |
| 3. Theoretical basis of machine learning..... | 7 |
| 4. Deep learning..... | 11 |
| 4.1. Model | 11 |
| 4.2. Loss function | 16 |
| 4.3. Optimization | 17 |
| 5. Semi-supervised learning..... | 19 |
| 6. Theoretical basis of semi-supervised models | 22 |
| 6.1. Regular autoencoder..... | 22 |
| 6.2. Variational autoencoder | 24 |
| 6.2.1. Semi-supervised variational autoencoder..... | 30 |
| 6.3. Generative adversarial network..... | 36 |
| 6.3.1. Semi-supervised generative adversarial network | 39 |
| 7. Evaluation and results..... | 41 |
| 7.1. Discussion..... | 46 |
| 2.2. Visualization | 47 |
| 8. Conclusion | 52 |
| 9. Bibliography | 53 |

1. Introduction

One of the most important tasks in bioinformatics is the assembly process, which refers to combining fragments of DNA, obtained by the sequencing device. These fragments will be referred to as reads, and the goal of the assembly process is to fully reconstruct the DNA. The modern *de novo* assembly process is conducted over a pre-generated set of reads and is realized in three basic phases: overlap, layout, and consensus.

Due to the imprecision of sequencing devices, some overlaps are wrongly detected, and they can overcomplicate the following phases. On the other hand, as a result of the same drawback, some overlaps will be omitted and they will not be processed in the layout phase. However, overlap algorithms are somewhat robust to these imperfections. Unfortunately, there are other specific reads, that will be referred to as chimeric and repeat, with unique characteristics, which are reflected on their coverage graph, that additionally impede the next phases.

A read can definitely be classified as chimeric or repeat when taking into account the information about its mapping onto the reference genome. We will therefore manually label some reads sequenced from already assembled genomes, but since this labeling process is extremely slow and usually rather grueling, we will propose methods that are based on semi-supervised learning with as little labeled data as possible. As the input in a model, we will use a signal that represent the coverage graph of the corresponding read. When finished, we expect that models will be able to generalize on any read, regardless whether the reference genome has already been assembled or not.

In the next chapter, we will define specific types of reads. In the chapters 3 and 4, we will briefly introduce theoretical fundamentals of machine and deep learning. Basic assumptions and theoretical background of semi-supervised models are presented in chapter 5. Concrete semi-supervised models will be explained in chapter 6. Evaluation and results will be presented in chapter 7, and all of it will be summed up in chapter 8.

2. Types of reads

Before we introduce specific types of reads, we will define a very simplified overlap phase that is based on the a very unrealistic assumption, that a sequencer makes no mistakes while reading nucleotide pairs. In that case, the shared region between two reads will be indicated as an overlap if the shared region is a prefix of one read, and a suffix of another, and contains no mismatches.

Chimeric read. A read is referred to as chimeric if it contains detached parts of the reference genome that were concatenated into one read as a consequence of a mistake made by a sequencing device.

Performing an overlap phase with the presence of chimeric reads can cause junction of the disjointed parts of the reference genome, and therefore they need to be eliminated.

Repeat read. A read is referred to as repeat if it contains a repetitive region of the reference genome. A repeat region is a consecutive sequence of nucleotide pairs that is replicated at different positions of the reference genome. Depending on the position of the repeat region in the read, the read is referred to as a left, centered or right repeat.

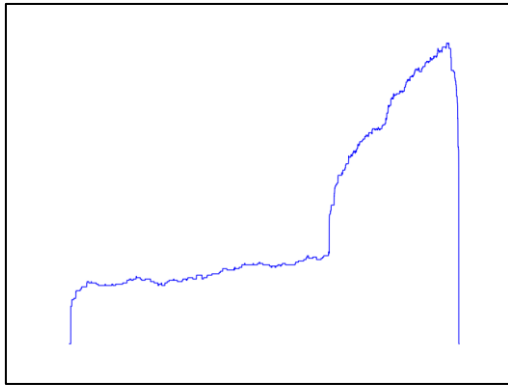
Centered repeats do not obstruct sequencing process, but left and right repeats can make it needlessly intricate and need to be removed as well.

Regular read. A read is referred to as regular if it is neither chimeric nor repeat.

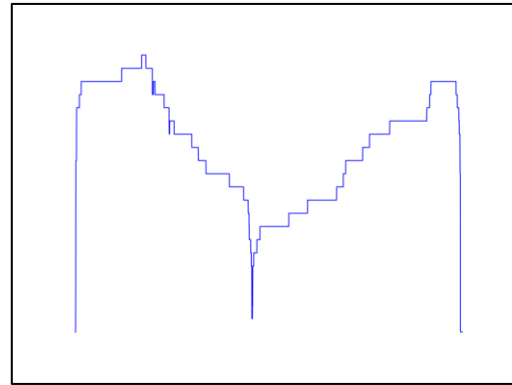
Coverage graph. A coverage graph is a graph which is created for each read by counting the number of times that each nucleotide pair in the read overlapped with some other read. Considering the circumstances in which coverage graphs are generated, we assume that the discriminative features of each read will be reflected in their coverage graph. Figures 2.2, 2.3 and 2.4 show possible sequencing scenarios which result in creating either chimeric, repeat or regular reads with the corresponding coverage graphs.

A chimeric coverage graph is characterized by a sudden drop followed by a sudden raise. This point represents the place where the two detached parts of genome have connected. On the other hand, a repeat coverage graph has greater

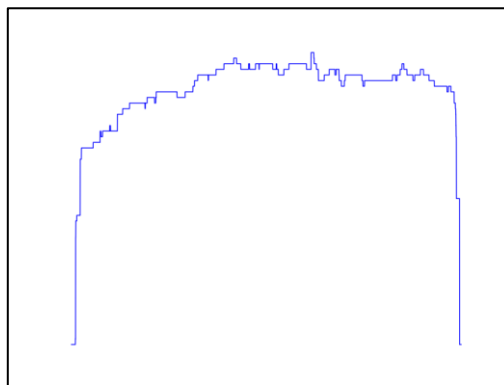
coverage on the one side, either left or right, than on the other. Areas with high coverage represent repeat regions of the reference genome. Finally, a coverage graph of a regular read does not contain any prominent aberrations. Examples of coverage graphs with the corresponding classes are shown in Figure 2.1.



a)



b)



c)

Figure 2.1. a) coverage graph of a repeat read. b) coverage graph of a chimeric read. c) coverage graph of a regular read.

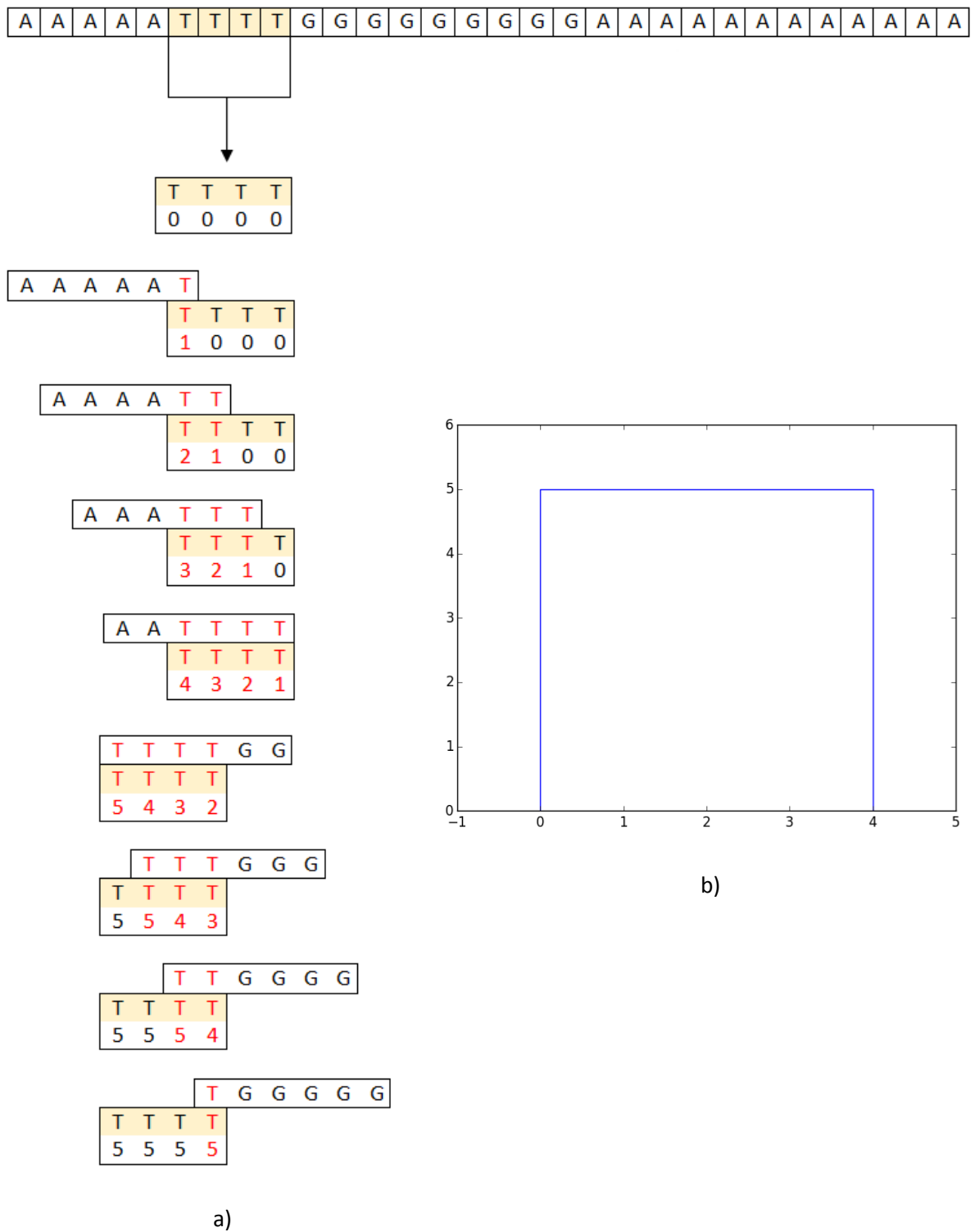
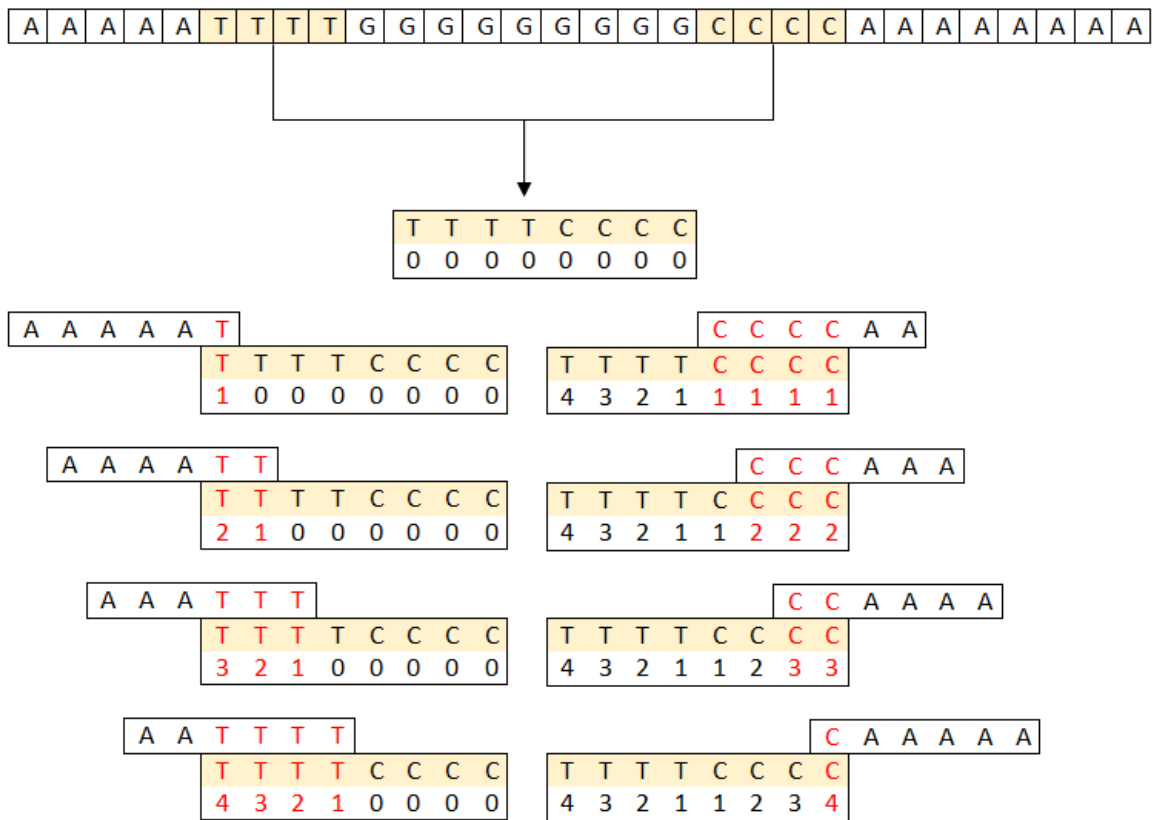
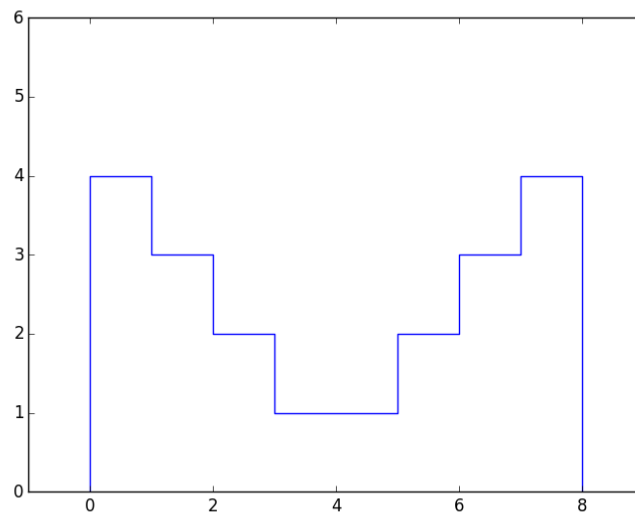


Figure 2.2. Generation process of a coverage graph on a regular read a) with the corresponding coverage graph b). The nucleotide pairs colored yellow show the observed regular read over which coverage graph is built. The figure shows all reads that overlap with the observed read with the update of the coverage graph marked in red.



a)



b)

Figure 2.3. Generation process of a coverage graph on a chimeric read a) with the corresponding coverage graph b). The nucleotide pairs colored yellow show the observed chimeric read that was formed by concatenating separate parts of the reference genome also colored yellow. The figure a) shows all the reads that overlap with the observed read. The coverage update is marked in red. We can notice a gradual decrease in the coverage graph at the point where the sequencer connected the separated parts of the original genome. As this part does not exist in the original genome, it correlates with the sudden drop in the coverage graph.

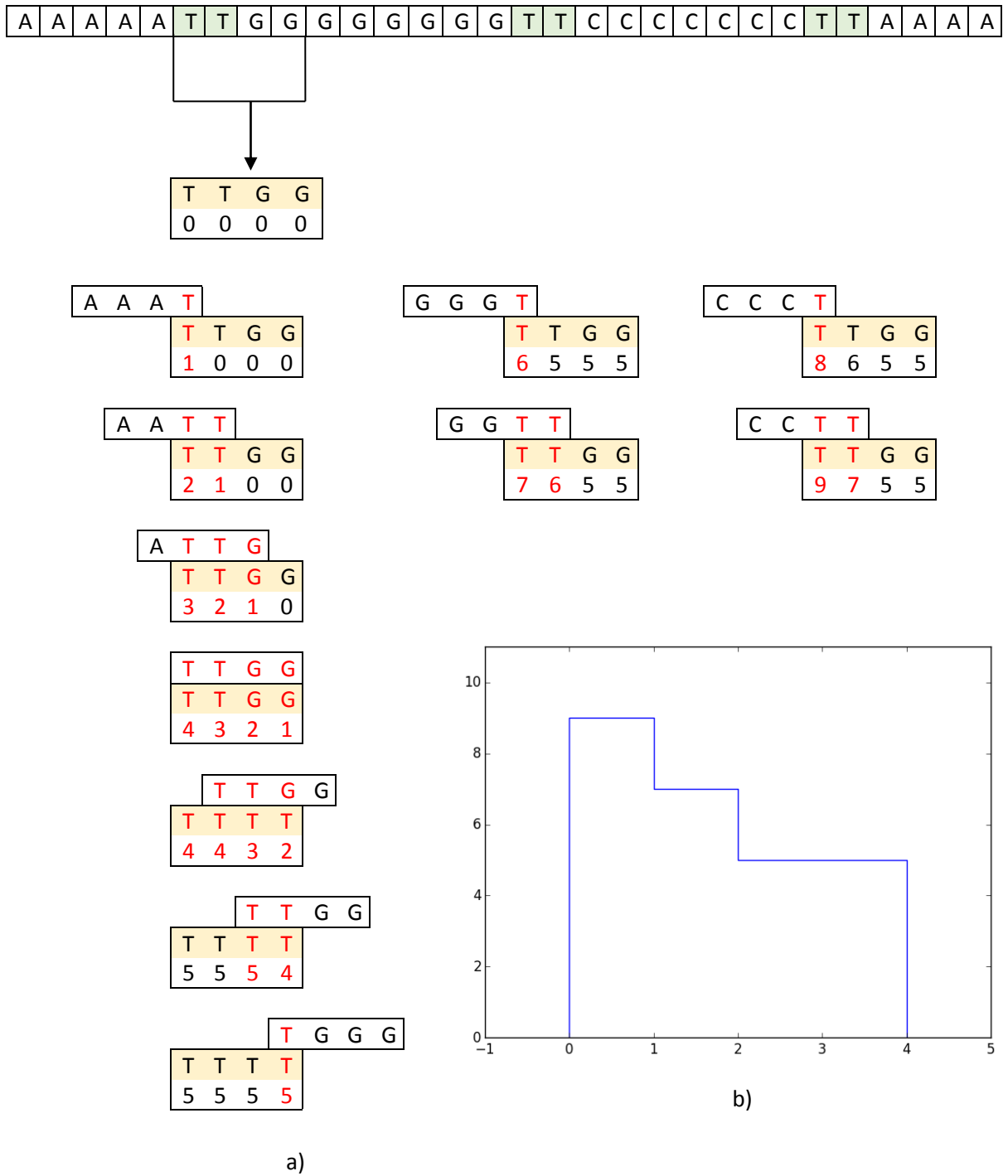


Figure 2.4. Generation process of a coverage graph on a repeat read a) with the corresponding coverage graph b). The nucleotide pairs colored yellow show the observed left repeat read over which the coverage graph will be built, and the nucleotide pairs colored green show the repeat regions in the reference genome. The figure a) shows all the reads that overlap with the observed read. The coverage update is marked in red. Since that read over which coverage graph was built contains repeat regions, all reads near these regions are also taken into consideration while building the coverage graph. Because of that, one side of the coverage graph has higher coverage than the other.

3. Theoretical basis of machine learning

"Machine learning is a way of programming computers to optimize a performance criterion using example data or past experience" [1]. We want to emphasize that this is not the only definition of machine learning. On the contrary, there are plenty of other different definitions, but the common denominator of all of them is the ability of the an algorithm to learn from data.

Machine learning algorithms can be divided into many categories, and we will list only a few of them. The first division distinguishes algorithms based on the type of their output values and categorizes them either as regression or classification. Generally speaking, an algorithm is referred to as regression if it is used for predicting continuous values. If it is used for predicting a class label from a predefined set of admissible labels, it is referred to as classification.

Furthermore, classification algorithms can be categorized as generative or discriminative, depending on whether the algorithm models the joint probability of input and its label or not. Generative models learn the distribution of individual classes by modeling the joint distribution of the example and its corresponding label, and based on that probability perform classification tasks. On the other hand, discriminative models perform classification tasks based on the learned boundaries between classes, without modeling the joint probability at all.

Finally, considering the type of the provided input information, we can distinguish two categories of machine learning tasks: supervised and unsupervised learning. Supervised learning uses input elements as well as the corresponding class labels, while unsupervised learning uses only the input elements. Semi-supervised learning lies somewhere in between supervised and unsupervised learning and will be explained in detail in chapter 5.

Before we outline the theoretical foundations and concepts of machine learning, we need to agree on the notation that will be used in further definitions.

- X** a set of training examples (input elements)
- $\mathbf{x}^{(i)}$** the i -th example (input element) which corresponds to the i -th row in the matrix X

$y^{(i)}$ the i -th expected output value associated with $x^{(i)}$ for supervised learning

As we will be focusing on semi-supervised machine learning algorithms used for classification tasks, it should be noted that the following definitions will be limited to this restriction.

Any machine learning algorithm has three basic elements: [2].

1. Model

Model (H) is defined as a set of hypotheses (h), where each hypothesis relates each input to one of the permissible outputs. In that respect, we can think of a model as a parameterized function, where each ordered set of parameters corresponds to a different hypothesis. Formula 3.1. shows the relation between the model and the hypotheses.

$$H = \{h(x | \theta)\}_{\theta} \quad (3.1)$$

2. Loss function

Loss function (L) calculates the error of the current hypothesis by comparing the expected output with the calculated output $h(x | \theta)$.

Error function (E) is defined as the expected value of the loss function, and it is shown in formula 3.2.

$$E(h | X) = \frac{1}{N} \sum_1^N L(x^{(i)}) \quad (3.2)$$

3. Optimization

Optimization is a process of finding the hypothesis that minimizes the error function on the set that contains the input elements from X. This process is also known as the learning process, and it is shown in formula 3.3.

$$h^* = \operatorname{argmin}_h E(h | X) \quad (3.3)$$

Working with machine learning entails making lot of numerous assumptions, and choosing a model is only one of them. Usually, the complexity of a model can be

defined with a set of parameters, and choosing a model correlates with choosing the set of parameters that describe it. These specific parameters are called hyperparameters. We want to emphasize that since these parameters are not the part of a model itself, they cannot be learned by the optimization step, which finds the optimal hypothesis in the defined model. In general, we want to choose a model which complexity matches the complexity of the function that has actually generated the data. Choosing a model can lead to one of the following two scenarios.

- **Underfitting** is a case where a model is not as complex as the function that has generated the data, and it does not have the capacity to interpret it. This case is shown in Figure 3.1.a.
- **Overfitting** is a case where a model is much more complex than the original function, so it can easily adapt to the data and learn noise in it. This case is shown in Figure 3.1.b.

The question, then, arises: how do we choose an optimal model? The idea is to divide examples into two sets: a train and test set. The model is trained on the train set, while the generalization performance is measured on the test set. This process is known as a cross-validation. A very simple model will produce an equal amount of mistakes on both sets, while a very complex model will show good performance on the train set, and bad performance on the test set. In order to find the optimal model, we need to find balance between errors on the train and test set. This effect is shown in Figure 3.2.

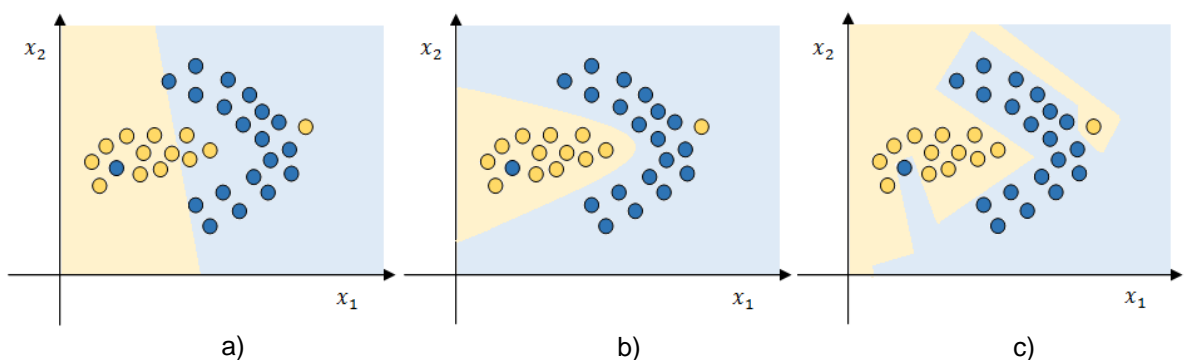


Figure 3.1. a) underfitted model. b) optimal model. c) overfitted model. All three figures show the decision boundary depending on the complexity of the model. If the model is too simple, as shown in a), it has no capacity to model real data dependencies. On the other hand, if the model is too complex it will easily adapt to the noise in the data c). The optimal model is somewhere in between.

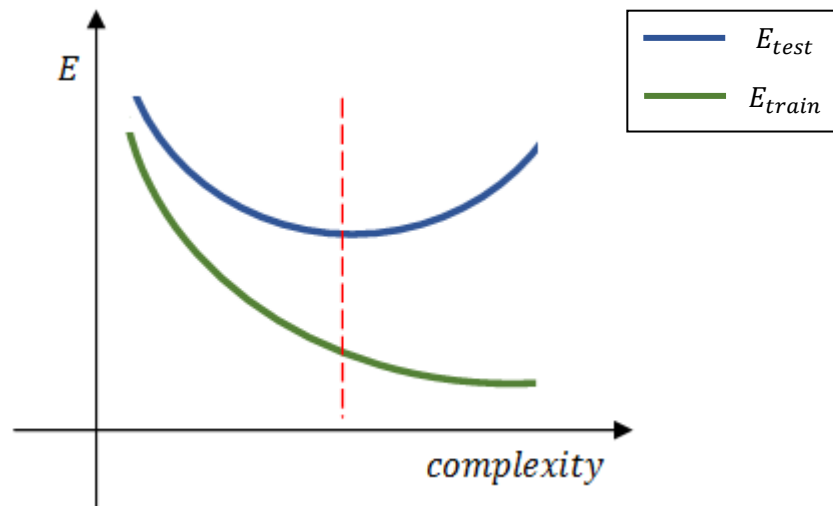


Figure 3.2. Dependencies of the error function on the train and test set with respect to the complexity of the model. We can notice that the error on the train set decreases with the increase of the model's complexity, while the error on the test function starts to increase after the point representing the optimal complexity. This is the expected behavior because more complicated models can learn decision boundaries with arbitrary high complexity. In that case, the generalization ability vanishes, which results in meager performance on the unseen examples in the test set, and an excellent one on the training data. The green dashed line represents optimal models, and it is a place where the underfitting and overfitting areas meet.

4. Deep learning

Deep learning is a growing trend in machine learning, and in essence, deep learning algorithms do not differ from any other machine learning algorithm since they can be interpreted through the basic elements: the model, loss function, and optimization. However, over the last couple of years, their popularity has increased significantly, due to the fact that recent experiments achieved state of the art results in areas of computer vision and speech recognition, which made them a hot topic in the area of machine learning.

As already said, we will describe deep learning algorithms suited for classification tasks owing to the fact that they are the main focus of the thesis.

4.1. Model

The basic model of any deep learning algorithm is a feed-forward deep neural network which consists of layers, as its basic computational units. The layers are determined with the certain set of parameters which is used for computing the output of a layer. In the rest of the thesis, the output of the layer will be referred to as a feature map. The feed-forward neural network has a composite structure that can be interpreted as a computational graph, and it is important to note that the output of a layer is only affected by the last extracted feature map as well as that computational graph of a feed-forward neural network does not contain any cycles.

There are many other types of neural networks such as residual and recurrent networks that extend the functionality of simple feed-forward networks with residual connections and cycles in the computational graph, but since they are not a part of this thesis we will not pay attention to them.

The basic model of the deep neural network is shown in Figure 4.1.

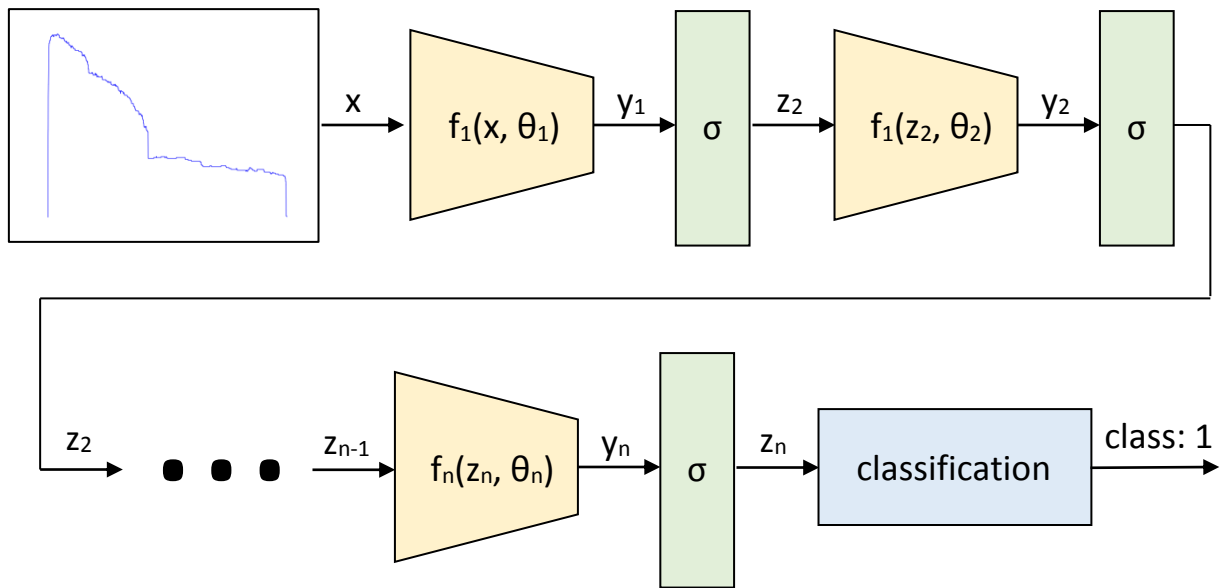


Figure 4.1. Basic feed-forward neural network architecture. The yellow parts represent layers, the green parts represent non-linearity among features maps, and the blue part represents the classification part of the neural network. We can notice that only the first layer is in touch with the input, while others operate on the latest extracted feature map.

There are many types of layers, but we are going to introduce only ones that were used in this thesis.

Fully connected layer is a basic layer, whose output is calculated according to the following matrix multiplication formula, where x is an input vector, W is a weight matrix and b is a bias vector:

$$y = x \cdot W + b$$

The fully connected layer has the capacity to model an arbitrary linear relationship in the input vector, but due to the fact that each feature in the output vector is calculated over the entire input vector, this approach demands a lot of trainable parameters and it is prone to overfitting [3]. Because of that, it is common practice to place the fully connected layer at the end of the pipeline so it can model relationships between extracted features and the corresponding classes. The graphical representation of a fully connected layer is represented in Figure 4.2.a.

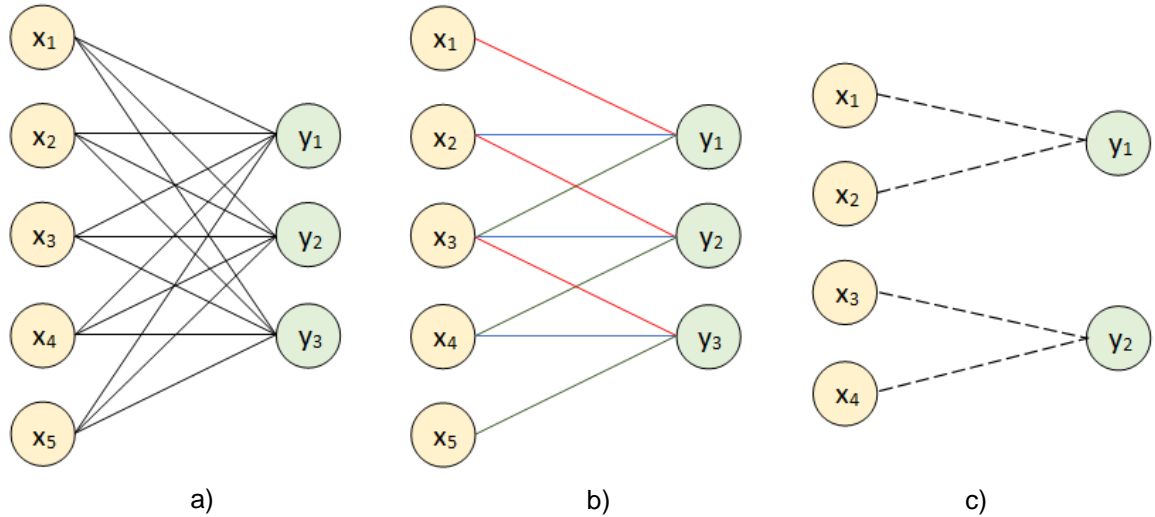


Figure 4.2. a) fully connected layer. b) convolution layer. c) pooling layer. Different colors in the convolution layer represent shared parameters, while the dashed lines indicate that the layer contains no learnable parameters. It can be noticed that the convolution layer demands only $k \cdot |y|$ where k is the kernel size, while the fully connected layer needs $|x| \cdot |y|$ if the bias variable is ignored.

Convolution layer uses convolution rather than a matrix product to compute the output.

Discrete 1D convolution is defined with the following formula, where x and w are arbitrary discrete signals:

$$(x * w)(t) = \sum_{n=-\infty}^{\infty} x(n) \cdot w(t - n)$$

In order to make the computation simpler, we used cross-correlation instead and assumed that values of functions x and w for the argument which does not belong in the domain are equal to zero. w is a discrete signal defined for only K domain elements and is usually referred to as the convolution kernel, and K is referred to as kernel length. The adjusted formula is shown below.

$$(x * w)(t) = \sum_{n=-K/2}^{K/2} x(t + n) \cdot w(n)$$

By using a convolution layer instead of a fully connected layer, we achieved a significant parameter reduction. Each feature in the feature map is now connected only to the adjacent elements in the input vector, where adjacency is controlled by

kernel length, and not to all elements, which was the case with the fully connected layer. Extracting features by using convolution on a 1D signal is shown in Figure 4.3.

Furthermore, we want to emphasize that convolution is equivariant to displacement in the input parts [3]. This means that if we move some part of the input, we will obtain the same features in the feature map, but in different positions.

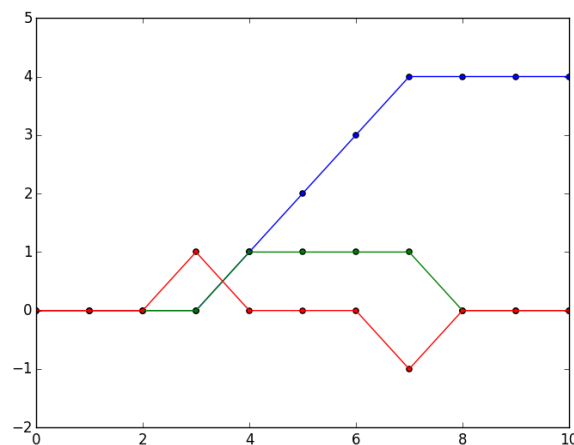


Figure 4.3. The figure shows the original 1D signal painted in blue color, and the results of convolution operations with the kernel $k=[-1, 1]$. The green signal is the result of applying convolution on the original signal, and the red one is the result of convolving the previously calculated convolution with the same kernel. We can infer that this kernel can be used to detect changes in the signal direction.

Also, we want to stress out that over the same input representations, multiple convolutions with multiple kernels can be calculated. With this approach, the model is given the ability to maintain different kernels over the same input representations and in one pass extracts different features.

Pooling is a sampling process, whose goal is to down-sample an input representation based on a pooling function, which maps topologically adjacent features in the input to one feature in the output, causing dimensionality reduction. The most commonly used pooling layer is max pooling. There are two main reasons for using the pooling layer:

- it reduces the spatial size and thereby reduces the number of parameters that need to be trained and the number of computations over them

- feature maps are invariant to small displacements in the input, which is useful in cases where classification information is more important to detect a specific feature, rather than binding the feature to the specific location [4].

It is common practice to periodically insert pooling layers between successive convolution layers as shown in Figure 4.4.

Transpose layer is a general abstract layer which can invert the result of a layer and generate input vector taking feature map as input.

Batch-normalization layer standardizes each feature in the feature map by applying the following formula where μ and σ are the mean and standard deviation of features in the current batch and γ and β are learnable parameters that control the influence of batch-normalization operation:

$$y = \gamma \cdot \frac{x - \mu}{\sigma} + \beta$$

Using batch-normalization layer can speed up the training process [4].

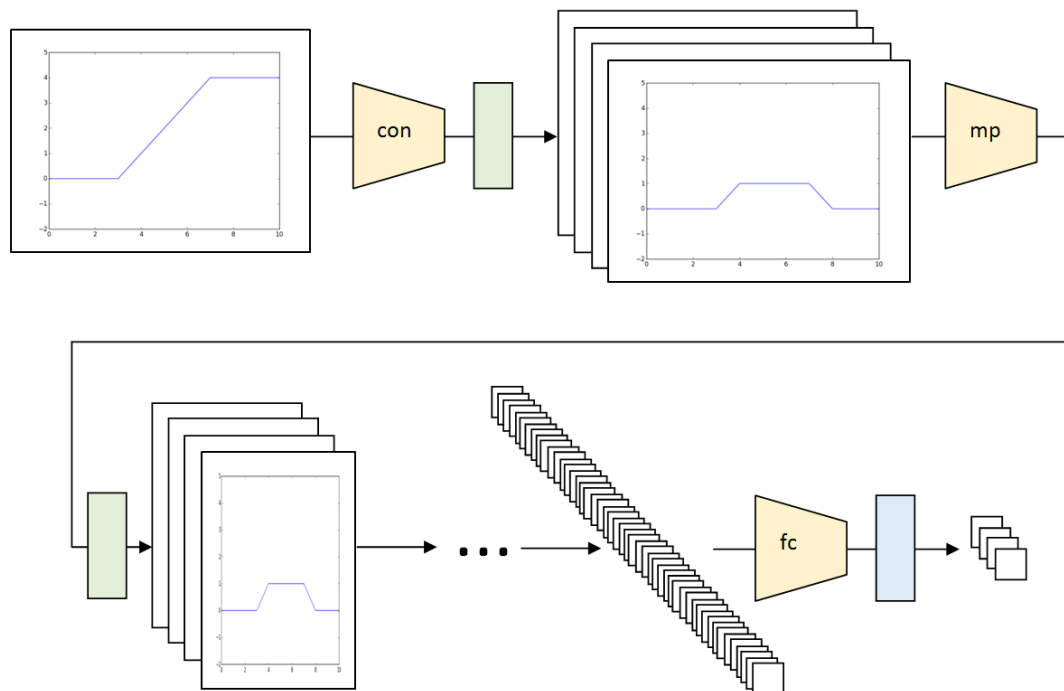


Figure 4.4. General deep neural network architecture that uses a convolution layer. Convolution layers are used in the early stages of the computations to extract useful features. Pooling layers are positioned in-between them in order to reduce the number of computations. Fully connected layers are positioned at the end of the pipeline in order to find relations between class labels and all extracted features.

4.2. Loss function

The loss function is based on the following two assumptions:

- the last output contains $N_{classes}$ elements, where $N_{classes}$ is the number of classes
- each element in the output vector corresponds to one of the class labels, and the greater the value, the more likely it is that the input element belongs to the corresponding class

Therefore, we need a way of translating outputs of the final layer into a multinomial probability distribution over class labels, and using the softmax function is one way in which we can achieve this.

The softmax function is a generalization of a sigmoid function that translates an N-dimensional vector of arbitrary real values to an N-dimensional vector of real values in the range [0, 1] that sum up to 1. It is formally defined by the following formula.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{i=1}^N e^{z_i}}$$

We are building our classifier based on an assumption that each example belongs to exactly one class. Because of that, we can represent class labels as a vector of L elements where only one element is equal to 1, and the corresponding index represents the class label, while all other elements are equal to 0. This label representation is known as one hot encoding.

Furthermore, the loss function can be defined as negative log-likelihood, where y is the output of the softmax function, and y' is the class label:

$$Loss(y, y') = -\log p(y = y' | \theta)$$

Having a label encoded as one hot vector, we can derive the final form of the loss function:

$$Loss(y, y') = -\log\left(\sum_{k=1}^{N_{label}} y'_k \cdot y_k\right)$$

Figure 4.5 shows the calculation of the loss function when the label and output vectors are provided.

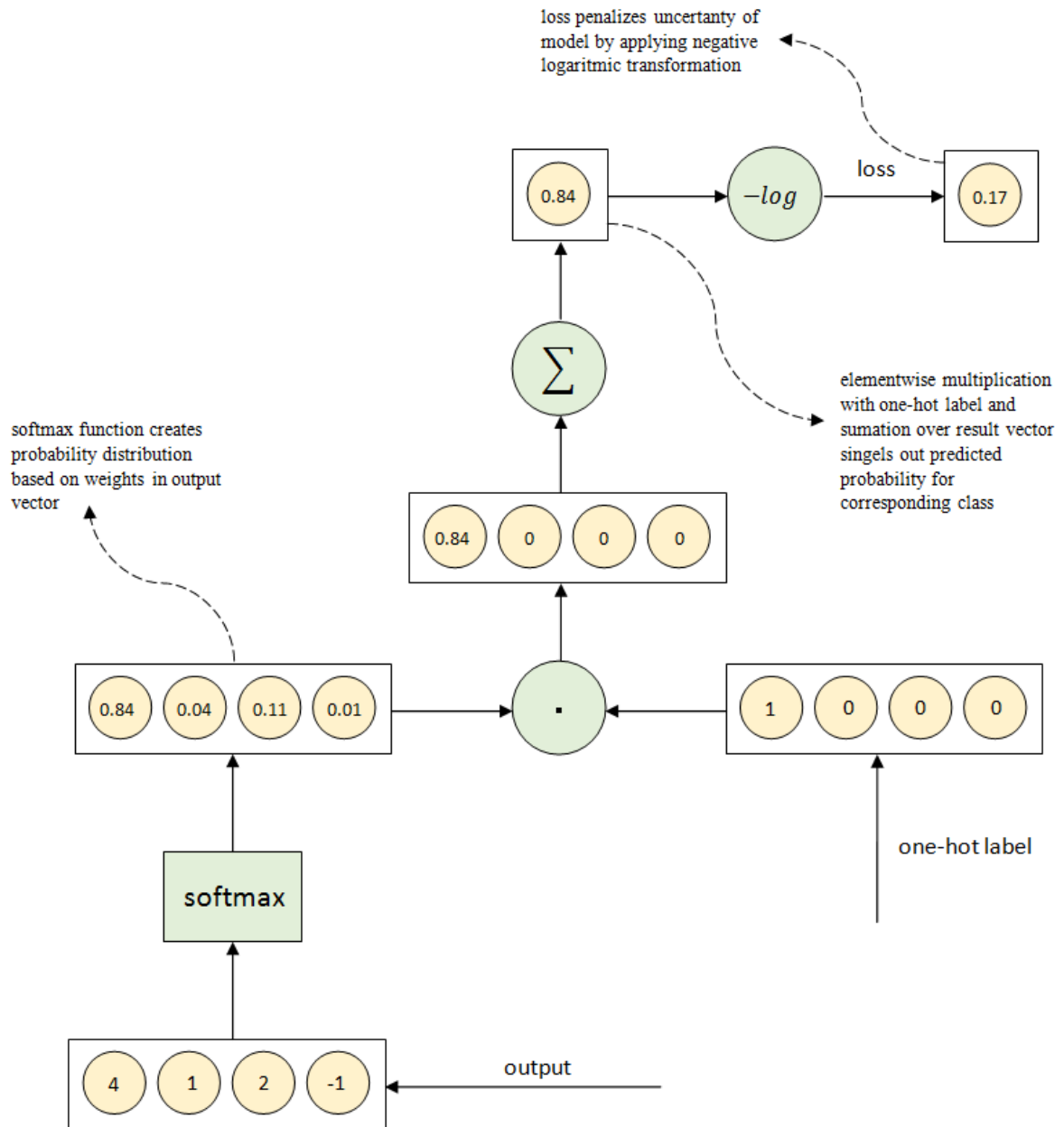


Figure 4.5. Calculation of the loss function having $[4, 1, 2, -1]$ as the output vector and class label 0. When encoded to the one-hot representation, the label is represented as $[1, 0, 0, 0]$.

4.3. Optimization

Due to the composite structure and non-linear operations applied among feature maps decision boundary is usually very complex, and the corresponding function does not have a solution in a closed form, so we will have to use some variant of

iterative optimization in order to perform the optimization step. Gradient descent is a first-order iterative optimization algorithm that finds local optima by taking small steps proportional to the negative gradient of the loss function at the current point. Since this algorithm uses derivatives of the loss function with respect to an arbitrary parameter of the model, we need to determine $\frac{\partial L}{\partial \theta_i}$. As has already been said, owing to the high complexity of the function that the deep neural net models, it is difficult to find exact formulas of this derivatives. Therefore, we will use backpropagation algorithm that performs in two steps:

- forward phase calculates the output of each elementary function whilst storing derivatives of the output with respect to the input of the function, that are known as local gradients
- backward phase spreads gradient of loss with respect to the input by using the chain rule $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$ and stored local gradients calculated in the forward pass

Gradient descent has many problems in cases where equal displacements in perpendicular directions of the arguments of the loss function imply unequal values of the loss function. This behavior is very common near the local optimum. In that case, gradient descent oscillates mostly in one dimension and temporizes the convergence. Therefore, instead of the vanilla gradient descent, it is common practice to use advanced learning algorithms, like adaptive moment estimation - ADAM, also based on the calculated gradients that cope with the mentioned problem more efficiently.

5. Semi-supervised learning

Semi-supervised learning is somewhere in between supervised and unsupervised learning. Usually, most of the elements in the dataset are unlabeled, and only a few of them are provided with a label. More formally, the dataset $X = \{x_i\}_{i=1..N}$ can be divided into two parts:

- $X_l = \{x_i\}_{i=1..N_{labeled}}$ for which labels $Y_l = \{y_i\}_{i=1..N_{labeled}}$ are provided
- $X_u = \{x_i\}_{i=N_{labeled}+1..N}$ for which labels are not provided

In order for semi-supervised learning to work at all, certain assumptions need to hold. The main one among them can be formulated as follows: if two points x_1, x_2 in a high-density region are close, the same should be true for the corresponding outputs y_1 and y_2 [5]. This assumption is quite intuitive but it is important to note that without it generalization would not be possible because a model is trained on a finite train set, while being expected to predict on a potentially infinite test set.

The next point that needs to be elaborated is the reason why we use semi-supervised learning at all. The answer lies in the fact that unlabeled data is usually very easy to obtain, but whereas the annotation process takes a lot of time and is usually tedious. Therefore, the goal of semi-supervised learning is to take advantage of the unlabeled data and achieve better results using both labeled and unlabeled data rather than using each of them separately.

Before we move on, we still have to answer the question of how can the unlabeled data help a classifier to obtain better results. It all comes down to the fact that using only labeled data, a model is forced to be fit to the usually deficient train set. Although unlabeled data contains no labels, they can interpret the overall data distribution in some way, and classifier can use this information to make a better prediction. Based on the way in which an unsupervised algorithm tries to contextualize the data distribution, we differentiate between various types of semi-supervised models. We will define two of them: generative models and manifold learning models [5, 6].

As already said in chapter 3, using generative models means tacitly modeling conditional density $p(x|y)$. Therefore, any additional information about $p(x)$ through unlabeled examples is very useful and can be used to find parameters of conditional distribution such that the training set generated from the conditional distributions is as likely as possible. One of the major strengths of generative models as a semi-supervised technique is the ability to incorporate domain-specific knowledge in the model. A drawback of this approach is the necessity to work with latent variables and expectations. Furthermore, if we create a model based on the wrong assumptions we could obtain worse results. The conceptualization of this model is shown in Figure 5.1.

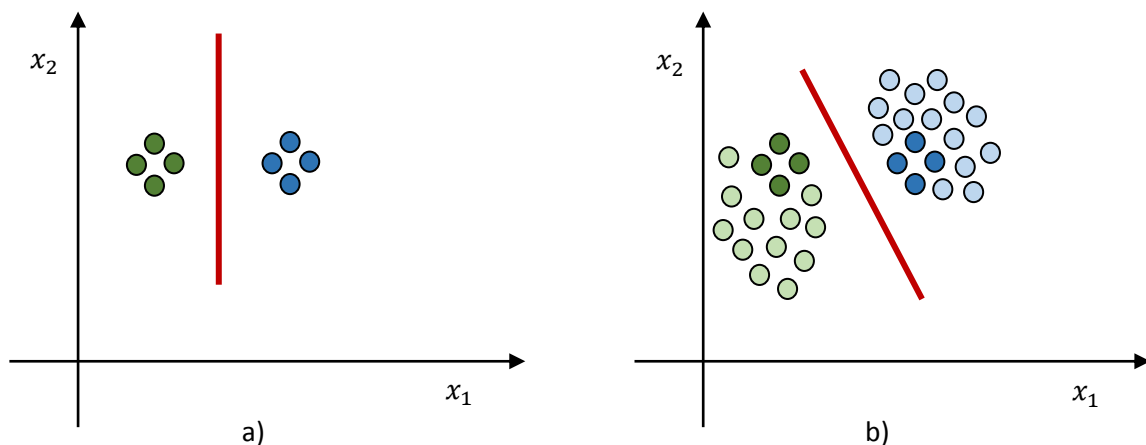


Figure 5.1. a) decision boundary based only on the labeled data. b) decision boundary based on both labeled and unlabeled data. Green circles represent inputs of one class while blue ones represent another one. Darker circles represent labeled data, the lighter ones represent unlabeled data and the red line represents the decision boundary that was calculated on all data. Since we have not opted for a formal model to classify data, the decision boundary was determined based on the assumption that it should lie in the low-density region. In this case, we can think of lighter circles as elements with latent labels that determine the class, so the overall process can be seen as classification with additional information on marginal density.

On the other hand, a manifold learning model is based on the following assumption: the high-dimensional data roughly lies on a low dimensional manifold [5]. Loosely defined, a manifold is a connected set of points that can be approximated by considering only a small number of dimensions embedded in a higher dimensional space [6]. The models will try to determine the low-dimensional representation of the input, and based on that lower dimensional interpretation

build a classifier that will be trained only on labeled data. However, since latent space has fewer dimensions, the classifier requires less labeled data to achieve the same performance as when no unlabeled data was used at all. The idea of manifold learning models is shown in Figure 5.2.

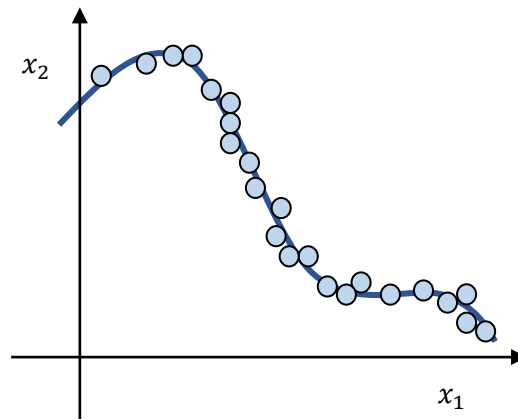


Figure 5.2. The figure shows samples from a 2D distribution as light blue circles. We can notice that sampled data are concentrated near the 1D manifold represented by the dark blue curve. During the training process, the model should easily infer the underlying manifold of the 2D data representation.

6. Theoretical basis of semi-supervised models.

We will describe three models of semi-supervised learning:

- semi-supervised learning based on a regular autoencoder
- semi-supervised learning based on a variational autoencoder
- semi-supervised learning based on a generative adversarial network

6.1. Regular autoencoder

An autoencoder is a neural network trained with a goal to reconstruct the input in the output of the neural network. The basic structure of the autoencoder has two main parts: the encoder and decoder. The encoder maps raw input to its latent space representation by extracting the features z , using the deterministic encoding function E . On the other hand, the decoder tries to reconstruct the original input from the latent representation z using the deterministic decoding function D . An autoencoder is trained by minimizing the predefined loss function, which usually correlates with the difference between input and its reconstruction, so the encoder and decoder needs to work together, so that the encoder extracts useful features from raw input, and the decoder uses them in order to provide the most plausible reconstruction. The basic autoencoder structure is shown in Figure 6.1.

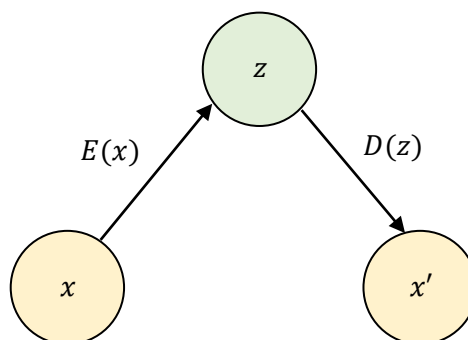


Figure 6.1. A general model of an autoencoder. $E(x)$ is a deterministic function that maps the input representation to its latent representation. This function will be referred to as the encoder. On the other hand, the deterministic function $D(z)$ translates examples from latent space to input space, and it will be referred to as the decoder.

There are several types of autoencoders, and they differ among themselves in the loss function they are trying to minimize. We will use a regular autoencoder whose encoder and decoder are implemented as deep neural networks, and whose loss function is a mean square error between the input and its reconstruction. In order to extract useful features, the dimensionality of the latent vector needs to be smaller than the dimensionality of the input vector. Otherwise, the neural network could easily learn to associate identity function with $E(x)$ and $D(z)$, extracting no useful features from the input. Forcing the autoencoder to extract useful features by reducing the dimensionality of the latent space is known as autoencoder regularization [6].

A semi-supervised model based on a regular autoencoder is shown in Figure 6.2. It is a simple enhancement of a regular autoencoder that uses the deterministic function C , usually implemented as a shallow model, that performs classification based on the features that have been extracted by the encoder. Therefore, the final objective function comprises two parts: classification and reconstruction loss. Classification loss is a standard cross-entropy-log loss introduced in section 4.2, while reconstruction loss is defined as the mean square error between the input and its reconstruction.

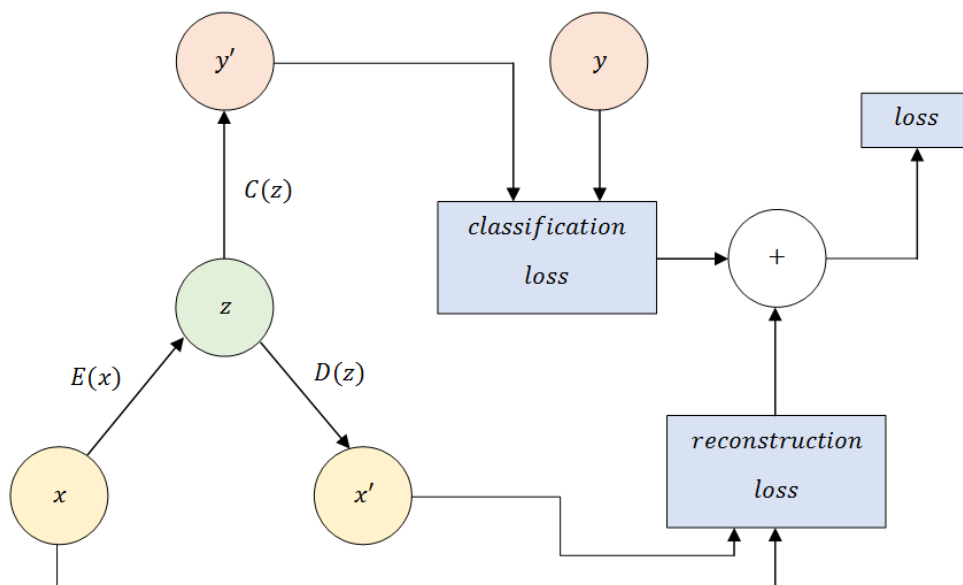


Figure 6.2. A general model of a semi-supervised model based on a regular autoencoder. $E(x)$ and $D(x)$ are deep neural networks while $C(x)$ is a shallow model, usually a fully connected layer. The common part in computational graphs of both losses represents $E(x)$. While minimizing total loss we are finding the optimal parameters of the encoder by balancing between correct classification and a correct reconstruction. We hope that these tasks will lead optimization in the same direction.

6.2. Variational autoencoder

A variational autoencoder is a generative model that combines ideas of deep learning with statistical methods. In essence, it is a probabilistic graphical model and since its structure resembles that of an autoencoder, it is studied together with other types of autoencoders. Therefore, it can be used as a feature extractor along with other functionalities of generative models.

It is assumed that for each element x from the dataset, there is a latent vector z , which cause generation of the example x by applying some transformation over the latent vector. More formally, z is a random variable sampled from the independent prior distribution $p(z)$, and x is then sampled from the conditional distribution $p_{\theta}(x | z)$ [7]. This assumption about how the data was generated can be interpreted as a probabilistic graphical model shown in Figure 6.3. Therefore, $p(x, z) = p(z) \cdot p_{\theta}(x | z)$ can be factorized in prior and conditional distribution, and we can derive $p(x)$ by marginalizing over latent variables.

We can notice that conditional distribution is parameterized with θ , and it is usually represented as a deep neural network.

Since $p(z)$ is independent of any other random variable, we can assume its distribution, and we chose the normal distribution with the mean 0, and the standard deviation 1. We justify this choice by the following two facts:

- this distribution has a simple and well-defined formula that will make further computations as simple as possible
- any distribution can be generated by taking a normally distributed variable and mapping it through an arbitrary complex function [7]

The second reason is proof that probability density function of conditional distribution $p_{\theta}(x | z)$ can be modeled by a deterministic function $f(z; \theta)$. However, in order to make the calculation and sampling process possible, we will assume that the conditional distribution can be approximated with some well known theoretical distribution, and we will propose a multivariate normal distribution with a diagonal covariance matrix. In mathematical notation, conditional distribution can be seen as:

$$p_{\theta}(x | z) = \mathcal{N}(f_{\mu}(z; \theta), f_{\sigma}(z; \theta))$$

Regardless of the chosen distribution, the idea is that the parameters of that distribution are determined by the deterministic functions, usually represented as deep neural networks, and if this function is complex enough and theoretical distribution is appropriate, the chosen distribution can efficiently mimic the real conditional distribution [7].

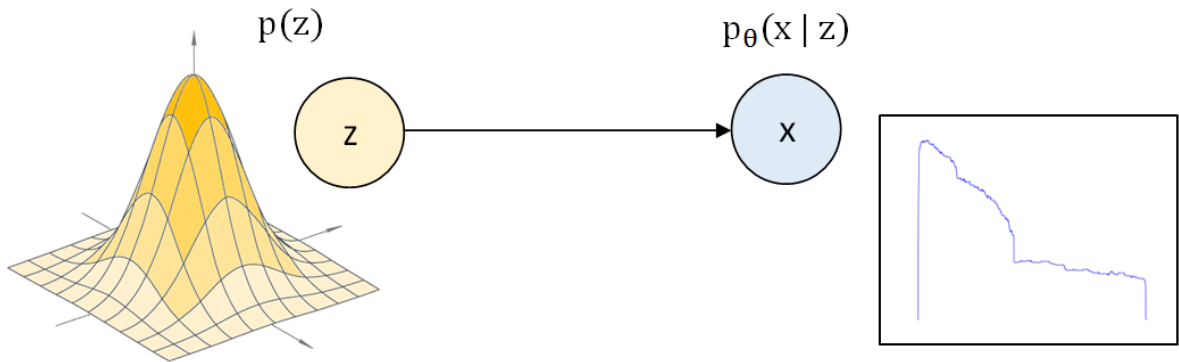


Figure 6.3. The probabilistic graphical model that models generation process from latent vectors that are sampled from normal distribution over conditional distribution parameterized with θ .

We will try to fit the defined model to the data. More precisely, given a set of N observations $\{x_1, x_2, \dots, x_N\}$ we wish to learn parameters θ of conditional distribution in that way that given observations are the most likely to be generated by the described process. In mathematical notation, we are trying to maximize the probability of each element in training set according to the following formula:

$$p(x) = \int p(z) \cdot p_{\theta}(x | z) \cdot dz$$

The next problem that we are faced with is the fact that we have a list of observations, and not the latent representations. Furthermore, and we cannot sample features z based on a $p(z | x)$, since it is intractable due to the intractability of the denominator, as it is shown in the following formula.

$$p(z | x) = \frac{p(z) \cdot p(x | z)}{p(x)} = \frac{p(z) \cdot p(x | z)}{\int p(z) \cdot p(x | z) \cdot dz}$$

Since we cannot calculate it we will try to approximate it with $q_\varphi(z|x)$. If this approximation is good enough, the sampled latent features will be similar to the one that actually produces X . Again, we will need to make certain assumptions about the distribution q , and we will represent it as a multinomial normal distribution whose parameters are the determined by deterministic function $g(x; \varphi)$, usually a deep neural network. Therefore, the conditional distribution takes the following form:

$$q_\varphi(z|x) = \mathcal{N}(g_\mu(x; \varphi), g_\sigma(x; \varphi))$$

With the sufficient capacity of $g(x; \varphi)$, the approximated distribution can mimic the intractable distribution [7].

The final model can conceptually be divided into two parts. The deep neural network that models $p_\theta(x|z)$, and the one that models $q_\varphi(z|x)$. These parts will be referred to as the decoder and the encoder, respectively. The basic model of a variational autoencoder is shown in Figure 6.4.

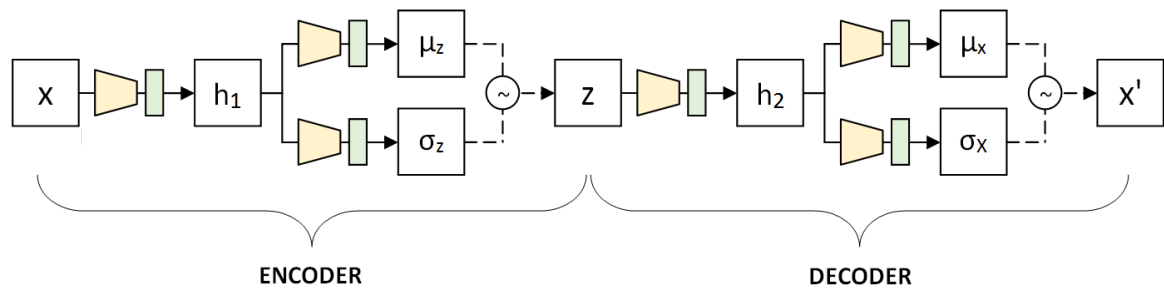


Figure 6.4. Overview of the variational autoencoder structure. The neural network is divided into two parts: the encoder and decoder. Solid lines represent a deterministic function, while dashed lines represent a sampling from the distribution modeled by the encoder or the decoder.

The optimal parameters θ and φ can be determined by maximizing the log likelihood of the observed examples. Further calculations are done using only one observation rather than all of them, because log-likelihood of all observations is modeled as the sum of individual likelihoods. The final form of the log-likelihood, shown in 6.1., can be derived by using simple mathematical operations [7, 8].

$$\begin{aligned}
L^{(i)} &= \log(p(x^{(i)})) \\
&= \sum_z q(z|x^{(i)}) \cdot \log(p(x^{(i)})) \\
&= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(z, x^{(i)})}{p(z|x^{(i)})}\right) \\
&= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(z, x^{(i)})}{q(z|x^{(i)})} \cdot \frac{q(z|x^{(i)})}{p(z|x^{(i)})}\right) \\
&= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(z, x^{(i)})}{q(z|x^{(i)})}\right) + \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{q(z|x^{(i)})}{p(z|x^{(i)})}\right) \\
L^{(i)} &= L_v^{(i)} + D_{KL}(q(z|x^{(i)}) || p(z|x^{(i)})) \tag{6.1}
\end{aligned}$$

The log-likelihood now comprises two parts: the lower variational bound (L_v) and KL-divergence (D_{KL}) between the real intractable distribution $p(z | x)$, and its approximation. The KL divergence is a measure of how one probability distribution differs from another. Assuming we use an encoder with an arbitrarily high capacity, $q(z | x)$ will match $p(z | x)$. In that case, the KL-divergence will become zero and by maximizing the lower variational bound we will indirectly maximize the log-likelihood of the observed examples. That way we made the intractable distribution $p(z | x)$ tractable, simply by replacing it with $q(z | x)$. L_v can be further decomposed by applying the following chain of operations.

$$\begin{aligned}
L_v^{(i)} &= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(z, x^{(i)})}{q(z|x^{(i)})}\right) \\
L_v^{(i)} &= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(x^{(i)}|z) \cdot p(z)}{q(z|x^{(i)})}\right) \\
L_v^{(i)} &= \sum_z q(z|x^{(i)}) \cdot \log\left(\frac{p(z)}{q(z|x^{(i)})}\right) + \sum_z q(z|x^{(i)}) \cdot \log(p(x^{(i)}|z)) \\
L_v^{(i)} &= -D_{KL}(q(z|x^{(i)}) || p(z)) + E_{z \sim q(z|x^{(i)})}(\log(p(x^{(i)}|z))) \tag{6.2}
\end{aligned}$$

It can be noticed from 6.2 that L_v consists of two parts: $-D_{KL}(q(z|x^{(i)}) || p(z))$, and $E_{z \sim q(z|x^{(i)})}(\log(p(x^{(i)}|z)))$ which represent the regularization factor and the quality of reconstruction, respectively [8]. The regularization factor is represented as a KL divergence, but this time measures how the distribution $q(z|x^{(i)})$ generated from the example $x^{(i)}$ differs from the prior distribution $p(z)$, which we assumed to be normal, and penalizes it according to the formula of the KL-divergence. On the other hand, the quality of reconstruction measures probability that $x^{(i)}$ is generated from $p(x^{(i)}|z)$, where z is sampled from a distribution $q(z|x^{(i)})$. In order to achieve the optimal performance, we need to consider both factors. The balance between them is shown in Figure 6.5.

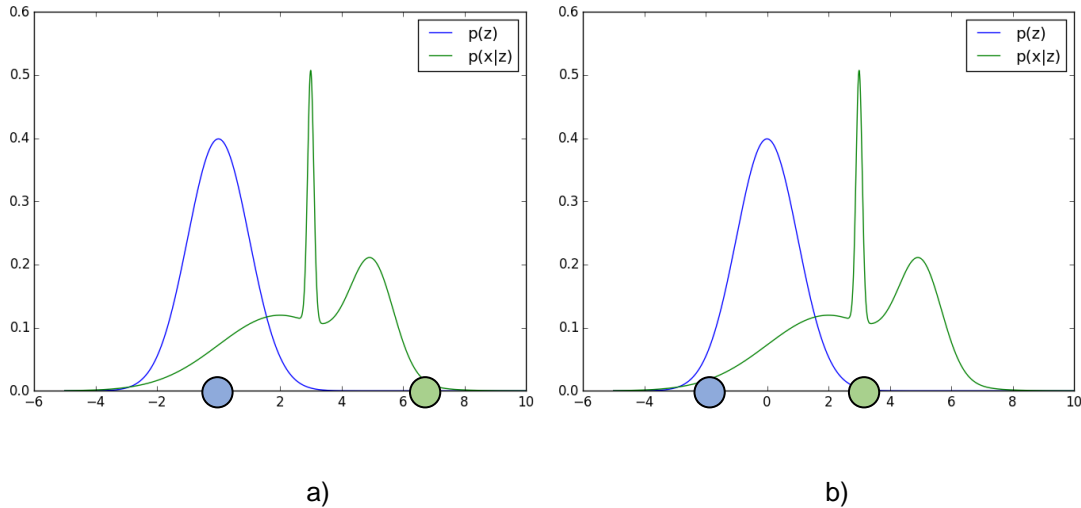


Figure 6.5. a) case where the quality of reconstruction is significantly lower than the regularization factor. b) case where the regularization factor is significantly lower than the quality of reconstruction. The blue distribution represents the prior $p(z)$, while the green distribution represents the conditional distribution $p(x^{(i)}|z)$ modeled by the decoder. The blue circle represents the latent vector that is sampled from $q(z|x^{(i)})$, while the green one represents $x^{(i)}$. In the first case, even though $q(z|x^{(i)})$ plausibly mimics the normal distribution, $x^{(i)}$ is not likely to be generated by this latent representation. On the other hand, even though $x^{(i)}$ is highly likely to be generated from the latent vector, the latent vector is not likely to be generated by the prior distribution.

Maximizing the lower variational bound demands an infeasible backpropagation step through a nondeterministic sampling layer at the end of the decoder. However, this sampling operation can be approximated with the following matrix multiplication, where μ and σ are parameters calculated by the encoder and ε is a random variable sampled from the normal distribution:

$$z = \mu + \sigma \cdot \varepsilon; \varepsilon \sim \mathcal{N}(0,1)$$

This fix is known as the reparameterization trick [8], and it is shown in Figure 6.6. The random variable z has the same distribution, but now it can be backpropagated through.

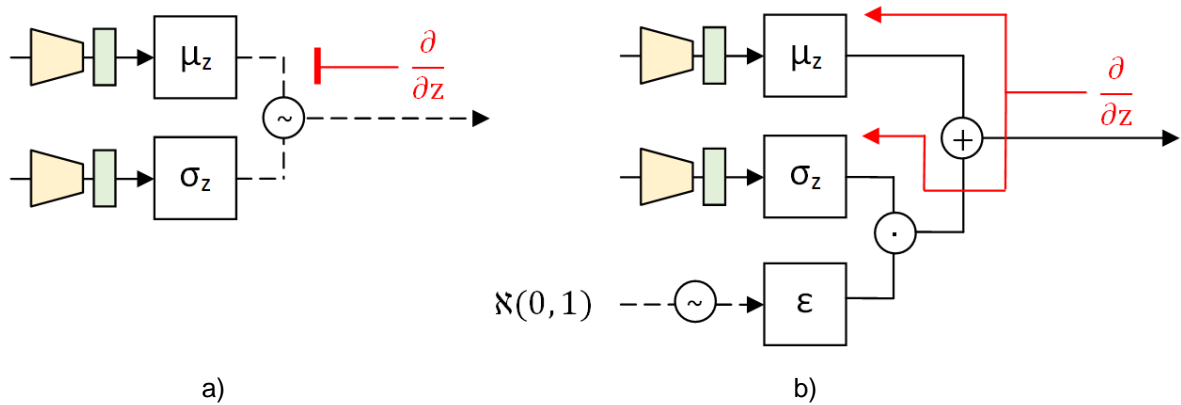


Figure 6.6. a) output of the encoder without the reparameterization trick. b) output of the encoder with the reparameterization trick. Dashed lines represent a sampling process, while solid lines represent a deterministic function. Due to the reparameterization trick, the gradient can pass into the encoder since only sampling process that remains is generating ε .

After defining all components of the model, we can derive concrete formulas for both the regularization factor and the quality of reconstruction based on the chosen output distribution.

1D Gaussian distribution is defined by the following formula:

$$p(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Since we assumed that multivariate normal distribution modeled by the decoder is defined with a diagonal covariance matrix, the elements x_i and x_j are independent, so we can factorize it. In order to combine it with the formula for the quality of the reconstruction, we applied a log-transformation over it.

$$p(x|\mu, \sigma) = \prod_{i=1}^N p(x_i|\mu_i, \sigma_i)$$

$$\begin{aligned}
\log p(x|\mu, \sigma) &= \log \prod_{i=1}^N p(x_i|\mu_i, \sigma_i) \\
&= \log \frac{1}{\sqrt{2\pi}^N \cdot \prod_{i=1}^N \sigma_i} \cdot e^{-\sum_{i=1}^N \frac{(x_i - \mu_i)^2}{2\sigma_i^2}} \\
&= \log \frac{1}{\sqrt{2\pi}^N} - \log \prod_{i=1}^N \sigma_i - \sum_{i=1}^N \frac{(x_i - \mu_i)^2}{2\sigma_i^2}
\end{aligned}$$

Since none of the constants affect the lower bound maximization, they can be removed. Furthermore, if we assume that the mini-batch is large enough that only one sample of the latent vector z is enough to approximate expectation we can derive the final formula for the quality of reconstruction, where N is dimensionality of the input[8]:

$$E_{z \sim q(z|x)} [\log p(\vec{x}|\vec{\mu}, \vec{\sigma})] = E_{z \sim q(z|x)} \left[-\sum_{i=1}^N \log \sigma_i + \frac{(x_i - \mu_i)^2}{2\sigma_i^2} \right] = -\sum_{i=1}^N \log \sigma_i + \frac{(x_i - \mu_i)^2}{2\sigma_i^2}$$

Following the same idea and using the formulas for normal distribution and KL-divergence, we can derive the following formula for the regularization factor as well, where H is dimensionality of latent size. [8].

$$D_{KL}[q(z|x) || p(z)] = -\frac{1}{2} \sum_{i=1}^H 1 + \log \sigma_{z_i}^2 - \mu_{z_i}^2 - \sigma_{z_i}^2$$

Therefore, final objective function is given in 6.3.

$$Loss^{(i)} = \sum_{i=1}^N \log \sigma_i + \frac{(x_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2} \sum_{i=1}^H 1 + \log \sigma_{z_i}^2 - \mu_{z_i}^2 - \sigma_{z_i}^2 \quad (6.3)$$

6.2.1. Semi-supervised variational autoencoder

We will make several changes to the assumption about how the data is generated in order to support semi-supervised learning. The probabilistic graphical model that represents the generation process suitable for semi-supervised learning is shown in Figure 6.7. There are two independent distributions [9]:

- latent distribution: $p(z) = \mathcal{N}(z; 0, 1)$
- label distribution: $p(y) = \frac{1}{N_{classes}}$, where y is represented as a one-hot vector

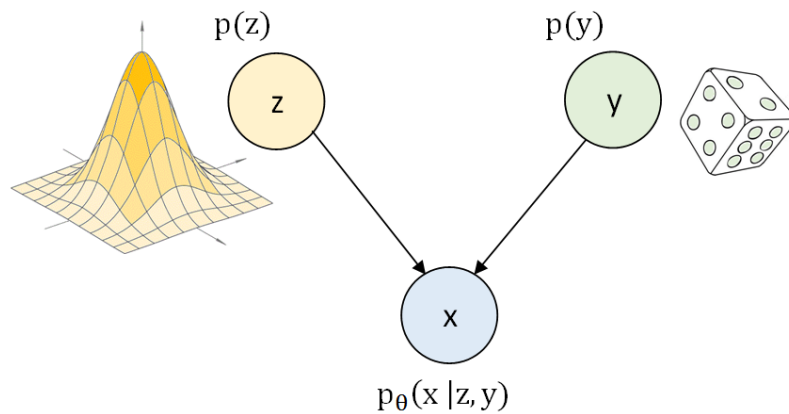


Figure 6.7. Probabilistic graphical model that describes the generation process in a semi-supervised autoencoder. The example x is generated based on a latent vector z , and a class label represented as a one-hot vector.

Owing to the fact that the new model apart from the example generation purpose should predict a class label for the given input, a classification module needs to be embedded.

Once again, based on the set of observations we will try to determine the optimal parameters such that the generated examples are as likely as possible. But, this time we face two possible scenarios:

- we observe both x and y , so this case is a simple extension of a variational autoencoder
- we observe only x , and treat y as a hidden variable

The second case is slightly more complicated because it demands interaction with the deterministic classifier. Same as before, we will use a variational approximation as the replacement for the intractable posterior. We will consider depending on whether the class label is observed or not. In the case where the label is observed we will use $q(z|x,y)$, and in the other case we will use $q(z,y|x) = q(y|x) \cdot q(z|x,y)$ [9]. It can be noticed that variational approximations rely on $q(z|y,x)$ that is identified with the encoder, and $q(y|x)$ that will be referred to as the classification part. With respect to this, a semi-supervised model is shown in Figure 6.8 [10].

In continuation, we will explain both scenarios depending on the observation of a class label.

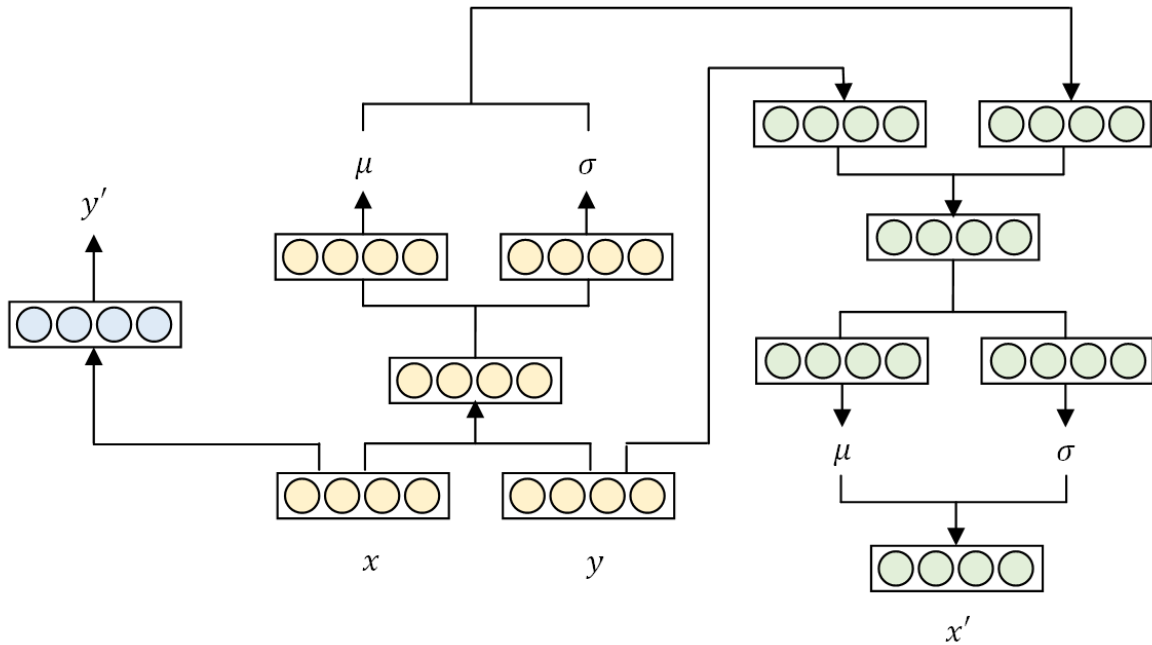


Figure 6.8. Model for a semi-supervised autoencoder. The blue part represents the classification part. The yellow part represents the encoder that based on the input, and the label generates the latent vector z . The green part represents the decoder that based on the generated latent representation and the class label generates the reconstructed example.

Case 1 - observed label

Similarly as was done with the autoencoders, we will try to maximize the log-likelihood $L = \log p(x, y)$ of x and corresponding label y .

Due to the intractability of $p(z|x, y)$ we will have to approximate it with the distribution $q(z|x, y)$, and following the same logic that was presented with variational autoencoders we can derive the following formula [10]:

$$L = \sum_z q(z|x, y) \cdot \log \frac{p(z, x, y)}{q(z|x, y)} - D_{KL}(q(z|x, y) || p(z|x, y))$$

The first part of L is lower variational bound (L_v) and we will try to maximize it, tacitly hoping that distribution $q(z|x, y)$ will match $p(z|x, y)$. Applying a similar transformation as we did for the variational autoencoders, L_v can be further decomposed [10].

$$\begin{aligned}
L_v &= E_{z \sim q(z|x,y)} \left[\log \frac{p(z, x, y)}{q(z|x, y)} \right] \\
&= E_{z \sim q(z|x,y)} [\log p(z) + \log p(y) + \log p_\theta(x|z, y) - \log q_\phi(z|x, y)] \\
&= E_{z \sim q(z|x,y)} [\log p_\theta(x|z, y) + \log p(y)] - E_{z \sim q(z|x,y)} \left[\log \frac{q_\phi(z|x, y)}{p(z)} \right] \\
&= E_{z \sim q(z|x,y)} [\log p_\theta(x|z, y)] + \log p(y) - D_{KL}(q_\phi(z|x, y) || p(z))
\end{aligned}$$

If we assume that expectation can be approximated precisely based only on one sampling, we can transform the upper formula to the final form of L_v , where the reconstruction cost and regularization factor have same formula as before.

$$L_v = \log p_\theta(x|z_{sample}, y) + \log p(y) - D_{KL}(q_\phi(z|x, y) || p(z)) = -Loss_{observed}(x, y) \quad (6.4)$$

Case 2 - unobserved label

On the other hand, if the class label is not observed, the formula will be extended [10].

$$L_v = E_{z, y \sim q(z, y|x)} \left[\log \frac{p(x, y, z)}{q(z, y|x)} \right]$$

Since we assumed inference model as $q(z, y|x) = q(y|x) \cdot q(z|x, y)$, we can rewrite the formula, to that the hidden variables are sampled according to this assumption [10].

$$\begin{aligned}
L_v &= E_{y \sim q(y|x)} \left[E_{z \sim q(z|x,y)} \log \frac{p(x, y, z)}{q(z, y|x)} \right] \\
&= E_{y \sim q(y|x)} \left[E_{z \sim q(z|x,y)} \log \frac{p(z) \cdot p(y) \cdot p(x|z, y)}{q(y|x) \cdot q(z|x, y)} \right] \\
&= E_{y \sim q(y|x)} [-Loss_{observed}(x, y) - E_{z \sim q(z|x,y)} \log(q(y|x))] \\
&= E_{y \sim q(y|x)} [-Loss_{observed}(x, y) - \log(q(y|x))] \\
&= \sum_y q(y|x) \cdot [-Loss_{observed}(x, y) - \log(q(y|x))] = -Loss_{unlabeled}(x) \quad (6.5)
\end{aligned}$$

We can formulate the objective function shown in 6.6 [9] by combining formulas 6.4 and 6.5:

$$Loss = \sum_{x,y \sim p_{labeled}} Loss_{observed}(x,y) + \sum_{x \sim p_{unlabeled}} Loss_{unlabeled}(x) \quad (6.6)$$

Finally, we can extend the loss function in a way that the deterministic classifier takes part in it in cases where a class label is observed by adding a cross-entropy part to the loss weighted by the hyperparameter α [9].

$$Loss^\alpha = Loss + \alpha \cdot E_{x,y \sim p_{labeled}} [-\log q(y|x)] \quad (6.7)$$

Finally, we introduce a model that is a hybrid between a pure variational autoencoder and its semi-supervised variant, known as M1+M2 model [9]. The variational autoencoder, referred to as M1, is pre-trained in order to extract features from the original input, and those features are presented to the semi-supervised variational autoencoder, referred to as M2, in order to perform classification. The structure of a stacked model is shown in Figure 6.9.

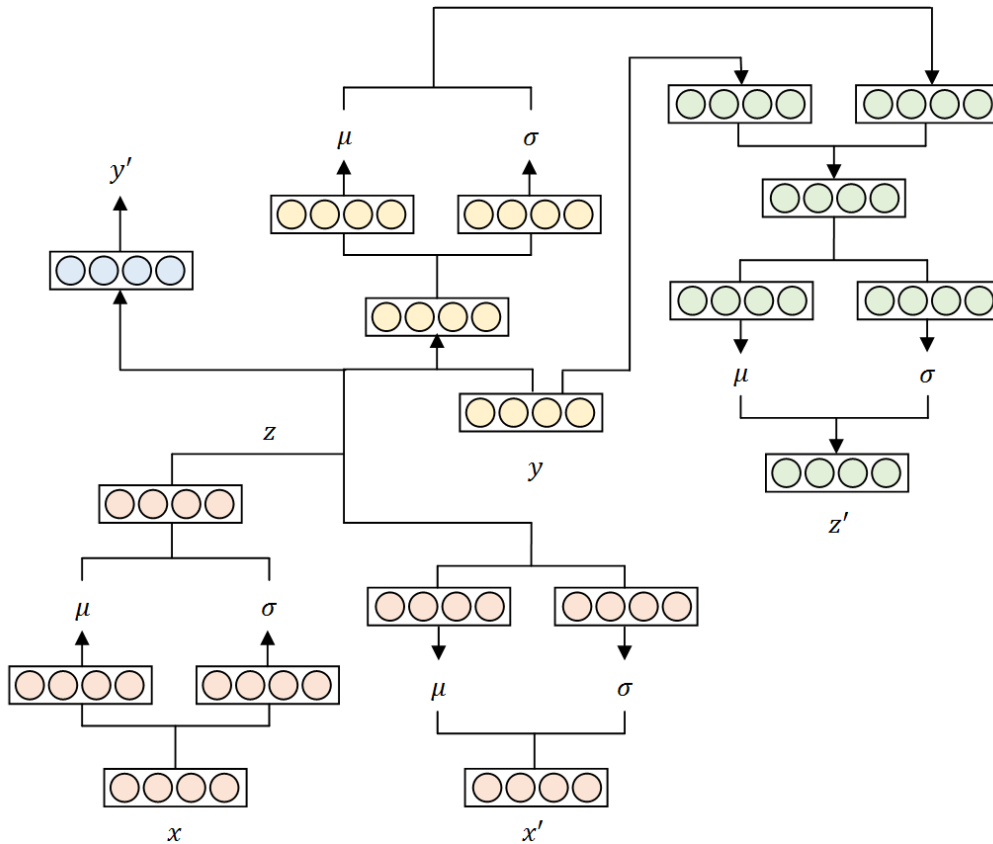


Figure 6.9. Structure of the M1+M2 model. The red part represents the variational autoencoder (M1), and the blue, yellow and green parts represent parts of the semi-supervised variational autoencoder (M2).

An algorithm of M1+M2 model is shown in Pseudocode 6.1.

Pseudocode 6.1. M1+M2 model

```

# M1 - algorithm

do
    x(1), ... x(n) = get_next_batch()
    loss = 0
    for i in {1, ... Nbatch}:
        μe, σe, z(i) = m1_encoder(x(i))
        μd, σd, x'(i) = m1_decoder(z(i))
        loss += Loss(i) # eq 6.3
    gradφ, gradθ = ∂loss/∂φ, ∂loss/∂θ
    φ = perform_ADAM_update(φ, gradφ) # encoder
    θ = perform_ADAM_update(θ, gradθ) # decoder
until convergence

# M2 - algorithm

do
    x(1), ... x(n) = get_next_batch()
    loss_observed = 0
    loss_unobserved = 0
    for i in {1, ... Nbatch}:
        if observed_label(x(i)):
            y(i) = get_label(x(i))
            μe, σe, z(i) = m2_encoder(x(i), y(i))
            μd, σd, x'(i) = m2_decoder(z(i), y(i))
            loss_observed += Loss(i) # eq 6.4
        else:
            loss' = 0
            for y(i) in {0, ... Nclasses}:
                μe, σe, z(i) = m2_encoder(x(i), y(i))
                μd, σd, x'(i) = m2_decoder(z(i), y(i))
                loss' += Loss(i) # eq 6.5
            loss_unobserved += loss'
    loss = loss_observed + loss_unobserved # eq 6.6 or 6.7
    gradφ, gradθ = ∂loss/∂φ, ∂loss/∂θ
    φ = perform_ADAM_update(φ, gradφ) # encoder
    θ = perform_ADAM_update(θ, gradθ) # decoder
until convergence

```

6.3. Generative adversarial networks

The basic idea of general adversarial networks (GAN) is to have two players, the generator and the discriminator, that are playing a game against one another [11].

- the generator generates data, and presents them to the discriminator
- the discriminator interprets data and tries to infer whether the data is real or fake.

The goal of the generator is to fool the discriminator by generating plausible input data, and the goal of the discriminator is not to be fooled by the generated examples. As both players are getting better and better during the training, the generator is forced to create data that is as realistic as possible, and the discriminator needs to get better at distinguishing real from fake examples, hopefully extracting some useful features, that can be used in further semi-supervised tasks.

The basic architecture of GAN is shown in Figure 6.10.

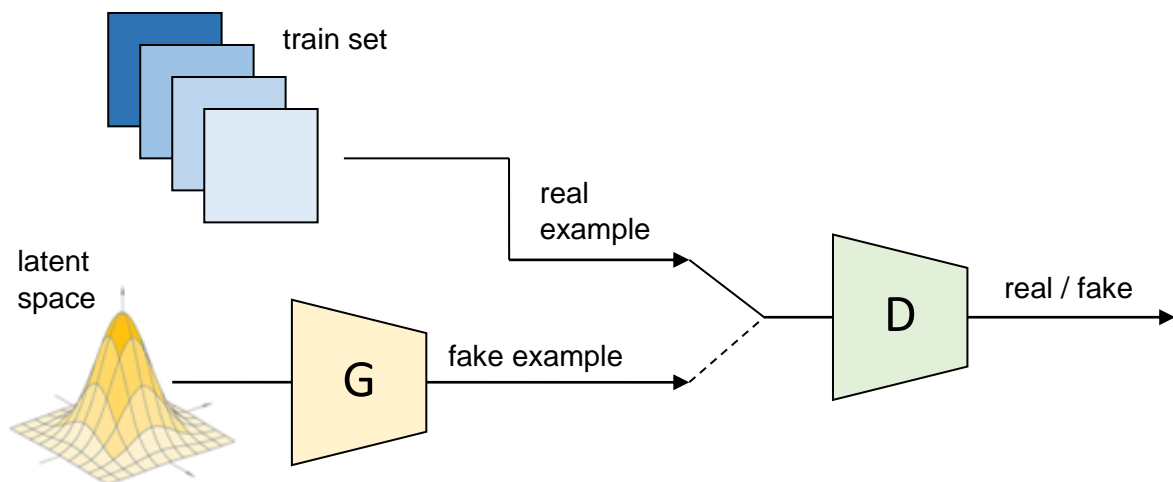


Figure 6.10. General structure of GAN. G and D are deep neural networks that represent the generator, and the discriminator respectively.

More formally, we assume that $p_{data}(x)$ is a distribution that generated the data, and the generator tries to learn the distribution $p_g(x)$. In order to achieve that we define a prior on the latent variable z , and a function $G(z; \theta)$ that represents a mapping from the latent space to the data space. In the context of probability, $G(z; \theta)$ can be seen as a conditional distribution. On the other hand, we define

$D(x; \varphi)$, whose output is a scalar that represents the probability that x came from the dataset, rather than being generated from distribution $p_g(x)$ [11]. We are training the discriminator to maximize the probability of distinguishing between both the real and generated dataset, and the generator is trained to do the opposite. This can be seen as a minmax game played with the value function $V(G, D)$, shown in formula 6.8 [11]:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [1 - \log D(G(z))] \quad (6.8)$$

A theoretical analysis indicated that if both the generator and discriminator are given enough capacity, the training process will end in a way that the generator recovers the true data distribution [11]. However, optimizing D in the inner loop is a computationally challenging task and considering that the model is trained on a finite training set, will probably lead to overfitting [11]. Instead, we alternate between optimizing the discriminator and generator cost functions.

Furthermore, a direct implementation of the value function defined with equation 6.8, will cause the gradient vanishing problem. In the early stages of a training process, discriminator easily distinguishes the real from fake examples and rejects fake examples with high probability, causing $1 - \log D(G(z)) \approx 0$, and providing no quality information for the update of a generator. In order to avoid this problem, generator can be trained to maximize $E_{z \sim p_z(z)} [\log D(G(z))]$, rather than minimizing $E_{z \sim p_z(z)} [1 - \log D(G(z))]$ [12].

If we adjust the formula, based on the equality $E_{z \sim p(z)} [G(z)] = E_{x \sim p_x(x)} [x]$, for a value function V in a way that it only depends on the random variable x , we can find the function of the optimal discriminator by deriving the value function with respect to the output of the discriminator.

$$V(D, G) = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [1 - \log D(G(z))]$$

$$V(D, G) = \int (p_{data}(x) [\log D(x)] + p_g(x) [1 - \log D(x)]) dx$$

Therefore, the optimal discriminator is given by the following formula [11]:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Now, the optimal generator can be calculated by combining formulas for the optimal discriminator and the value function of the min-max game [11].

$$\begin{aligned}
 V(D^*, G) &= E_{x \sim p_{data}(x)} \left[\frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + E_{x \sim p_g(x)} \left[\frac{p_g(x)}{p_{data}(x) + p_g(x)} \right] \\
 &= E_{x \sim p_{data}(x)} \left[\frac{\frac{p_{data}(x)}{2}}{\frac{p_{data}(x) + p_g(x)}{2}} \right] + E_{x \sim p_g(x)} \left[\frac{\frac{p_g(x)}{2}}{\frac{p_{data}(x) + p_g(x)}{2}} \right] \\
 &= -\log 4 + D_{KL}(p_{data} \parallel \frac{p_{data} + p_g}{2}) + D_{KL}(p_g \parallel \frac{p_{data} + p_g}{2})
 \end{aligned}$$

Since the KL divergence is non-negative and equal to zero only when the two distributions are equal, the global minimum of the value function with the optimal discriminator is achieved when $p_{data} = p_g$. The training process associated with the theoretical background of GANs is shown in Figure 6.11 [11].

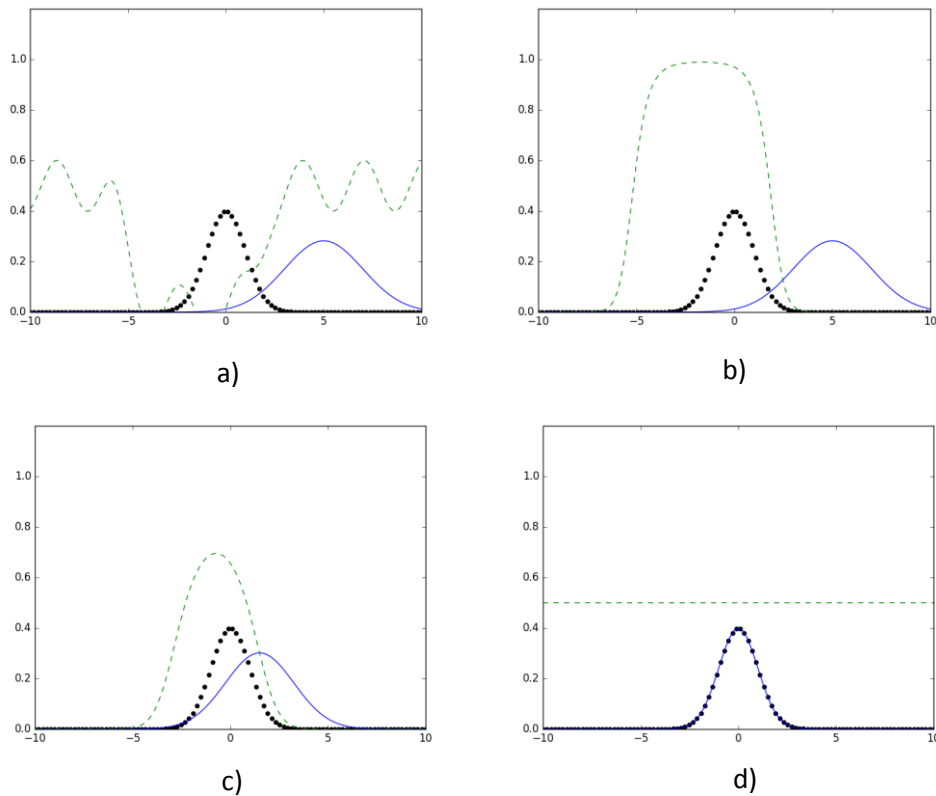


Figure 6.11. GAN learning process with 1D dimensional input. The black line represents the real data distribution, the blue line represents the generator's distribution while the green line represents outputs of the discriminator. a) the initialization phase where the discriminator is not trained yet. b) optimizing the discriminator resulting in $p_{data} / (p_{data} + p_g)$. c) change in the discriminator's output based on a new generator's distribution that is more similar to the real data distribution. d) the equilibrium state where $p_{data} = p_g$ and discriminator cannot differentiate real from fake data.

Generative adversarial networks are hard to train due to the fact that the optimal parameters are not found by minimizing a single cost function, but by finding the equilibrium of the game that is played between the two players [12]. Therefore, convergence is not guaranteed. Furthermore, alternating between optimizations of the generator and the discriminator could lead to other problems such as mode collapse [12].

6.3.1. Semi-supervised generative adversarial networks

In order to support semi-supervised learning, we need to make some changes to the adversarial model. We already said that the discriminator extracts useful features during the training process. In order to use them for classification, we will extend the discriminator in a way that it becomes responsible for the classification of real examples and differentiating the real from fake examples. More formally, the discriminator will produce $K+1$ outputs in a form $[p_{y_1}, p_{y_2}, \dots, p_{y_K}, p_{fake}]$, where K is the number of classes. p_{y_i} represents the probability that an example is real and belongs to the i -th class, and p_{fake} represents the probability that an example is fake [12, 13].

The output of a semi-supervised GAN can be easily transformed into the output of a regular GAN by summing probabilities associated with a real example: $p_{real} = \sum_{i=1}^{N_{classes}} p_{y_i}$.

Therefore, the minmax game played between the generator and discriminator stays more or less the same and is based on p_{fake} . This time the generator will try to fool the discriminator by producing samples that will cause $p_{fake} \approx 0$. On the other hand, the discriminator will try to minimize p_{fake} for examples that came from the data distribution. Minimizing p_{fake} implies distributing probability mass among p_i , and using the labeled examples for rearranging the class distribution in order to improve the classification results.

We proposed a method where semi-GAN is trained as a regular GAN in the first epochs only by using the unlabeled data in order to force the discriminator to learn to extract useful adversarial features. In the next epochs, we used cross-entropy

loss for the update of the discriminator. Since semi-GAN was pre-trained using unlabeled data only, we assume that the discriminator learns to bind previously learnt adversarial features with corresponding class using only labeled data.

An algorithm of semi-supervised GAN model is shown in Pseudocode 6.2.

Pseudocode 6.2. semi-supervised GAN

```

do
    if current_epoch < 10:
        x_real(1), ... x_real(n) = get_next_batch(unlabeled)
        x_fake(1), ... x_fake(n) = generate_from_generator()
        for step in {'discriminator', 'generator'}:
            loss_d, loss_g = 0
            for i in {1, ... Nbatch}:
                dis_real(i) = discriminator(x_real(i)).get_probability_real()
                dis_fake(i) = discriminator(x_fake(i)).get_probability_real()
                loss_d += -(log(dis_real(i)) + log(1-dis_fake(i)))
                loss_g += -log(dis_fake(i))
            gradφ, gradθ = ∂loss/∂φ, ∂loss/∂θ
            if step == 'discriminator':
                φ = perform_ADAM_update(φ, gradφ) # discriminator
            else:
                θ = perform_ADAM_update(θ, gradθ) # generator
        else:
            x_real(1), y(1), ... x_real(n), y(n) = get_next_batch(labeled)
            x_fake(1), ... x_fake(n) = generate_from_generator()
            for step in {'discriminator', 'generator'}:
                loss_d, loss_g = 0
                for i in {1, ... Nbatch}:
                    y'(i) = discriminator(x_real(i)).get_probability(y(i))
                    dis_fake(i) = discriminator(x_fake(i)).get_probability_real()
                    loss_d += cross_entropy_log_loss(y'(i), y(i))
                    loss_g += -log(dis_fake(i))
                gradφ, gradθ = ∂loss/∂φ, ∂loss/∂θ
                if step == 'discriminator':
                    φ = perform_ADAM_update(φ, gradφ) # discriminator
                else:
                    θ = perform_ADAM_update(θ, gradθ) # generator
    until convergence

```

7. Evaluation and results

We have not used raw coverage graphs as inputs for any model, but rather the signals that are the result of applying the following chain of operations:

- normalization
- down-sampling to the fixed length input

Squashing an output of a coverage graph to the interval $[0,1]$ retains the same form of the coverage graph, but enables the model to achieve the same performance as without normalization by virtue of having less complicated architecture.

Normalization is an optional transformation, but the down-sampling transformation is compulsory, because the fixed length input is one of the pre-requirements of the defined models. This hyperparameter needs to be carefully chosen because it directly affects the performance of the model to a great extent. The down-sampling effects with respect to the fixed length input are shown in Figure 7.1. In order to retain all important features of a coverage graph, but still, retain a model that is computationally efficient we decided to use an input of the length of 500, except for GANS where we used the fixed length of 100.

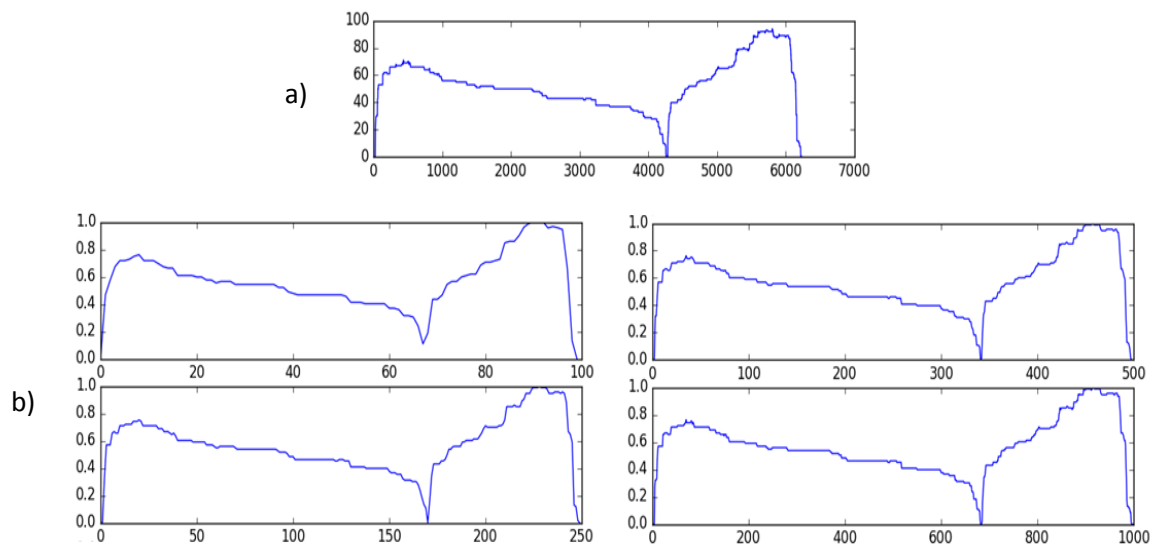


Figure 7.1. a) original coverage graph. b) transformations after down-sampling it to the length of 100, 250, 500 and 1000, respectively. It can be noticed that down-sampling to the length of 100 loses information about the height of a peak, while other signals keep all expressive parts of the coverage graph.

All models were trained on the dataset that was composed of overlaps generated from multiple reference genomes. We used graphmap [14] suited for overlaps generation by applying the following command:

```
graphmap -owler -r reads.fasta -d reads.fasta -o output.mhap
```

We have not used all overlaps because all the classes are not equally frequent. Since the repeat reads are not as recurrent as the regular ones, and the chimeric reads are somewhat obscure, running a model on the entire dataset would have an unavoidable bias towards the regular reads. In order to solve this problem we have designed a heuristic for detecting class label based on the following features:

- the median of the coverage graph
- the incline of certain parts of the coverage graph
- the mean value of the left and right side of the coverage graph, etc

We used this heuristic in order to somewhat balance the entire dataset. The dataset split is shown in Table 7.1.

Table 7.1. Data distribution over different sets. The balance of the unlabeled data is contingent upon the quality of the designed heuristics. The manually labeled data is divided into three sets: the train, validation, and test which are equally balanced.

| Set | | Number |
|----------------|------------|--------|
| unlabeled data | | 21600 |
| labeled data | train | 260 |
| | validation | 60 |
| | test | 240 |

During the training process we used:

- a mini-batch of size 32
- a data augmentation based on a random horizontal flip and random Gaussian noise

The training process was terminated after 50 epochs or when the performance on the validation set degraded to a great extent. Regardless of the condition that had terminated the training process, the performance of the model was evaluated on the test set, based on the model that had shown optimal performance on the validation set.

Training was performed on GEFORCE GTX TITAN BLACK graphic card and training time did not exceed 30 minutes for the most complicated model.

We trained 4 different models that were described in previous chapters: a simple feed-forward neural network (FF), a semi-supervised regular autoencoder (semi-AE), a M1+M2 model (M1+M2) and a semi-supervised generative adversarial network (semi-GAN). We have tried many different architectures and key details of winning architectures are shown in Table 7.2.

Table 7.2. Architectures that were used during the training of the described models with the chosen hyperparameters. conv K/N represents the convolution layer with kernel size K, and N filters, max-pool represents the max-pooling process, fc M represents the fully connected layer with M output features, bn represents the batch-normalization layer and conv⁻¹ and max-pool⁻¹ represent a transpose layer that invert the convolution or pooling operation.

| FF | SEMI-AE | M1+M2 | SEMI-GAN |
|--|--|---|---|
| conv 5/16 max-pool conv 3/32 max-pool conv 3/64 fc 256 fc 4 | encoder conv 5/16 max-pool conv 3/32 max-pool conv 3/64 fc 256 fc 10 decoder fc 8000 conv 3/64 ⁻¹ max-pool ⁻¹ conv 3/32 ⁻¹ max-pool ⁻¹ conv 5/16 ⁻¹ classificaton fc 4 | m1-encoder conv 5/16 max-pool conv 3/32 max-pool conv 3/64 fc 256 fc 10 m1-decoder fc 8000 conv 3/64 ⁻¹ max-pool ⁻¹ conv 3/32 ⁻¹ max-pool ⁻¹ conv 5/16 ⁻¹ fc 500 m2-encoder fc 64 fc 64 fc 3 m2-decoder fc 64 fc 64 fc 10 m2-classification fc 64 fc 64 fc 4 | generator fc 1600 bn conv 3/64 ⁻¹ bn max-pool ⁻¹ bn conv 3/32 ⁻¹ bn max-pool ⁻¹ bn conv 5/16 ⁻¹ fc 100 discriminator conv 5/16 bn max-pool bn conv 3/32 bn max-pool bn fc 256 bn fc 1024 fc 4 |
| input length: 500 | input length: 500 latent size: 10 | input length: 500 latent size M1: 10 latent size M2: 3 | input length: 100 latent size: 10 |
| non-linearity: ReLU optimizer: ADAM with learning rate 0.0001 | | | |

As the performance criterion, we used F-score[15], and the results are shown in Table 7.3.

Table 7.3. Confusion matrices and the corresponding F-score of the optimal models with respect to the number of labeled examples that were used during the training process (N). Tags C, L, R and N represent the chimeric read, left-repeat, right-repeat and regular read, respectively. The confusion matrix was calculated based on the premise that the element corresponding with row x and column y represents the number of examples that actually belong to class x, and the classifier has classified them in class y.

| | FF | SEMI-AE | M1+M2 | SEMI-GAN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|---|----------|----------|----------|---|---|---|----|----|----|---|---|----|----|----|---|---|---|---|----|---|---|----|---|---|----|--|--|---|---|---|---|---|----|----|---|---|---|----|----|---|---|---|---|---|----|---|---|---|---|---|----|---|--|---|---|---|---|---|----|---|---|---|---|----|----|---|---|---|---|---|----|---|---|---|---|---|----|--|--|---|---|---|---|---|----|---|---|---|---|---|----|---|---|---|---|---|----|---|---|---|---|---|----|
| N=15 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>36</td> <td>24</td> <td>31</td> <td>7</td> </tr> <tr> <th>L</th> <td>16</td> <td>36</td> <td>2</td> <td>7</td> </tr> <tr> <th>R</th> <td>7</td> <td>0</td> <td>27</td> <td>5</td> </tr> <tr> <th>N</th> <td>1</td> <td>0</td> <td>0</td> <td>41</td> </tr> </tbody> </table> | | C | L | R | N | C | 36 | 24 | 31 | 7 | L | 16 | 36 | 2 | 7 | R | 7 | 0 | 27 | 5 | N | 1 | 0 | 0 | 41 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>40</td> <td>11</td> <td>7</td> <td>5</td> </tr> <tr> <th>L</th> <td>11</td> <td>48</td> <td>2</td> <td>3</td> </tr> <tr> <th>R</th> <td>5</td> <td>1</td> <td>51</td> <td>6</td> </tr> <tr> <th>N</th> <td>4</td> <td>0</td> <td>0</td> <td>46</td> </tr> </tbody> </table> | | C | L | R | N | C | 40 | 11 | 7 | 5 | L | 11 | 48 | 2 | 3 | R | 5 | 1 | 51 | 6 | N | 4 | 0 | 0 | 46 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>41</td> <td>4</td> <td>2</td> <td>4</td> </tr> <tr> <th>L</th> <td>13</td> <td>56</td> <td>0</td> <td>1</td> </tr> <tr> <th>R</th> <td>6</td> <td>0</td> <td>58</td> <td>0</td> </tr> <tr> <th>N</th> <td>0</td> <td>0</td> <td>0</td> <td>55</td> </tr> </tbody> </table> | | C | L | R | N | C | 41 | 4 | 2 | 4 | L | 13 | 56 | 0 | 1 | R | 6 | 0 | 58 | 0 | N | 0 | 0 | 0 | 55 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>50</td> <td>0</td> <td>4</td> <td>1</td> </tr> <tr> <th>L</th> <td>2</td> <td>60</td> <td>0</td> <td>0</td> </tr> <tr> <th>R</th> <td>7</td> <td>0</td> <td>56</td> <td>0</td> </tr> <tr> <th>N</th> <td>1</td> <td>0</td> <td>0</td> <td>59</td> </tr> </tbody> </table> | | C | L | R | N | C | 50 | 0 | 4 | 1 | L | 2 | 60 | 0 | 0 | R | 7 | 0 | 56 | 0 | N | 1 | 0 | 0 | 59 |
| | | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | 36 | 24 | 31 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 16 | 36 | 2 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 7 | 0 | 27 | 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 1 | 0 | 0 | 41 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 40 | 11 | 7 | 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 11 | 48 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 5 | 1 | 51 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 4 | 0 | 0 | 46 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 41 | 4 | 2 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 13 | 56 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 6 | 0 | 58 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 0 | 0 | 0 | 55 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 50 | 0 | 4 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 2 | 60 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 7 | 0 | 56 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 1 | 0 | 0 | 59 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | F=0.5833 | F=0.7708 | F=0.8750 | F=0.9375 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N=30 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>40</td> <td>19</td> <td>9</td> <td>1</td> </tr> <tr> <th>L</th> <td>10</td> <td>41</td> <td>10</td> <td>9</td> </tr> <tr> <th>R</th> <td>7</td> <td>0</td> <td>41</td> <td>4</td> </tr> <tr> <th>N</th> <td>3</td> <td>0</td> <td>0</td> <td>46</td> </tr> </tbody> </table> | | C | L | R | N | C | 40 | 19 | 9 | 1 | L | 10 | 41 | 10 | 9 | R | 7 | 0 | 41 | 4 | N | 3 | 0 | 0 | 46 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>44</td> <td>4</td> <td>3</td> <td>1</td> </tr> <tr> <th>L</th> <td>8</td> <td>50</td> <td>0</td> <td>4</td> </tr> <tr> <th>R</th> <td>7</td> <td>6</td> <td>55</td> <td>3</td> </tr> <tr> <th>N</th> <td>1</td> <td>0</td> <td>2</td> <td>52</td> </tr> </tbody> </table> | | C | L | R | N | C | 44 | 4 | 3 | 1 | L | 8 | 50 | 0 | 4 | R | 7 | 6 | 55 | 3 | N | 1 | 0 | 2 | 52 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>46</td> <td>1</td> <td>3</td> <td>3</td> </tr> <tr> <th>L</th> <td>6</td> <td>59</td> <td>0</td> <td>0</td> </tr> <tr> <th>R</th> <td>6</td> <td>0</td> <td>57</td> <td>1</td> </tr> <tr> <th>N</th> <td>2</td> <td>0</td> <td>0</td> <td>56</td> </tr> </tbody> </table> | | C | L | R | N | C | 46 | 1 | 3 | 3 | L | 6 | 59 | 0 | 0 | R | 6 | 0 | 57 | 1 | N | 2 | 0 | 0 | 56 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>52</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <th>L</th> <td>3</td> <td>58</td> <td>0</td> <td>0</td> </tr> <tr> <th>R</th> <td>5</td> <td>0</td> <td>59</td> <td>0</td> </tr> <tr> <th>N</th> <td>0</td> <td>0</td> <td>0</td> <td>60</td> </tr> </tbody> </table> | | C | L | R | N | C | 52 | 2 | 1 | 0 | L | 3 | 58 | 0 | 0 | R | 5 | 0 | 59 | 0 | N | 0 | 0 | 0 | 60 |
| | | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | 40 | 19 | 9 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 10 | 41 | 10 | 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 7 | 0 | 41 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 3 | 0 | 0 | 46 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 44 | 4 | 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 8 | 50 | 0 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 7 | 6 | 55 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 1 | 0 | 2 | 52 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 46 | 1 | 3 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 6 | 59 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 6 | 0 | 57 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 2 | 0 | 0 | 56 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 52 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 3 | 58 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 5 | 0 | 59 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 0 | 0 | 0 | 60 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | F=0.7000 | F=0.8375 | F=0.9083 | F=0.9699 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N=70 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>40</td> <td>7</td> <td>5</td> <td>6</td> </tr> <tr> <th>L</th> <td>11</td> <td>50</td> <td>4</td> <td>8</td> </tr> <tr> <th>R</th> <td>8</td> <td>3</td> <td>51</td> <td>4</td> </tr> <tr> <th>N</th> <td>10</td> <td>0</td> <td>0</td> <td>42</td> </tr> </tbody> </table> | | C | L | R | N | C | 40 | 7 | 5 | 6 | L | 11 | 50 | 4 | 8 | R | 8 | 3 | 51 | 4 | N | 10 | 0 | 0 | 42 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>47</td> <td>4</td> <td>2</td> <td>2</td> </tr> <tr> <th>L</th> <td>7</td> <td>56</td> <td>0</td> <td>2</td> </tr> <tr> <th>R</th> <td>5</td> <td>0</td> <td>58</td> <td>1</td> </tr> <tr> <th>N</th> <td>1</td> <td>0</td> <td>0</td> <td>55</td> </tr> </tbody> </table> | | C | L | R | N | C | 47 | 4 | 2 | 2 | L | 7 | 56 | 0 | 2 | R | 5 | 0 | 58 | 1 | N | 1 | 0 | 0 | 55 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>52</td> <td>1</td> <td>3</td> <td>2</td> </tr> <tr> <th>L</th> <td>3</td> <td>58</td> <td>0</td> <td>0</td> </tr> <tr> <th>R</th> <td>3</td> <td>0</td> <td>59</td> <td>1</td> </tr> <tr> <th>N</th> <td>2</td> <td>1</td> <td>0</td> <td>59</td> </tr> </tbody> </table> | | C | L | R | N | C | 52 | 1 | 3 | 2 | L | 3 | 58 | 0 | 0 | R | 3 | 0 | 59 | 1 | N | 2 | 1 | 0 | 59 | <table border="1"> <thead> <tr> <th></th> <th>C</th> <th>L</th> <th>R</th> <th>N</th> </tr> </thead> <tbody> <tr> <th>C</th> <td>54</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <th>L</th> <td>3</td> <td>60</td> <td>0</td> <td>0</td> </tr> <tr> <th>R</th> <td>3</td> <td>0</td> <td>59</td> <td>0</td> </tr> <tr> <th>N</th> <td>0</td> <td>0</td> <td>0</td> <td>60</td> </tr> </tbody> </table> | | C | L | R | N | C | 54 | 0 | 1 | 0 | L | 3 | 60 | 0 | 0 | R | 3 | 0 | 59 | 0 | N | 0 | 0 | 0 | 60 |
| | | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | 40 | 7 | 5 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 11 | 50 | 4 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 8 | 3 | 51 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 10 | 0 | 0 | 42 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 47 | 4 | 2 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 7 | 56 | 0 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 5 | 0 | 58 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 1 | 0 | 0 | 55 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 52 | 1 | 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 3 | 58 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 3 | 0 | 59 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 2 | 1 | 0 | 59 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | C | L | R | N | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | 54 | 0 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| L | 3 | 60 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R | 3 | 0 | 59 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | 0 | 0 | 0 | 60 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | F=0.7408 | F=0.9000 | F=0.9341 | F=0.9708 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Due to the fact that all the models have difficulties distinguishing the chimeric class, we will compare results of the best two models based on their precision-recall curves [15]. Figure 7.2 shows the precision-recall curve for the chimeric class only, as well as the mean precision-recall curve over all 4 classes.

Finally, we have extended the assembly process by omitting the reads that were classified as chimeric and overlaps between the reads that were classified as left and right repeats. We used the semi-GAN as the classifier, and conducted an experiment over the following reference genomes: NCTC74, NCTC86, NCTC129, NCTC204. With this extension, we have assembled all listed genomes and compared this results with the results of the assembly process where chimeric and

repeat reads were not omitted in relation to the number of contigs and NG50 statistic. Contig is a set of overlapping reads that unambiguously determine a region of the reference genome, and the number of it be seen as a measure of complexity of the assembly process. NG50 statistic represents the length of the largest contig in a way that along with the larger contigs covers 50% of the original genome. NG50 can be seen as a measure of quality of the assembly process. This comparison is shown in Table 7.4.

Table 7.4. Comparison between the regular and the extended process. NOC represents the number of contigs, and NG50 represents NG50 statistics.

| | Extended process | Regular process |
|---------|------------------------|-----------------------|
| NCTC74 | NOC=16 NG50=573717 | NOC=21 NG50=545944 |
| NCTC86 | NOC=53 NG50=187128 | NOC=126 NG50=74836 |
| NCTC129 | NOC=12 NG50=1008551 | NOC=36 NG50=269581 |
| NCTC204 | NOC=29 NG50=475910 | NOC=39 NG50=319779 |

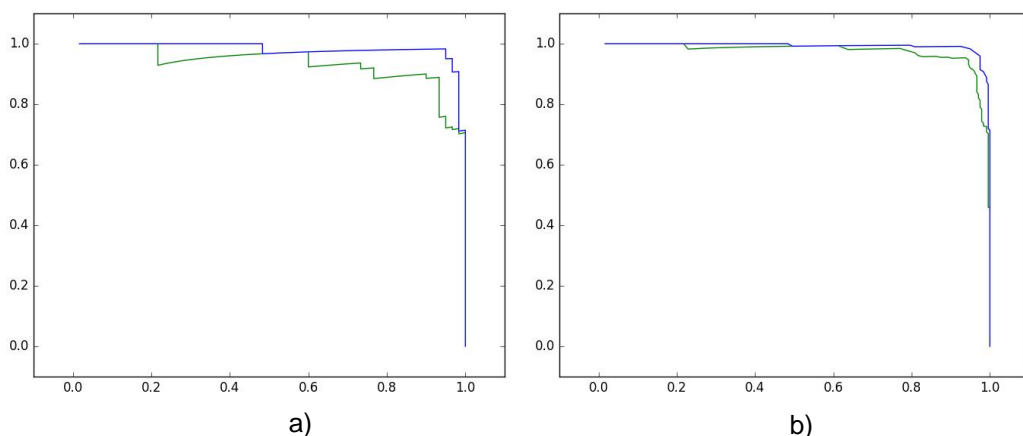


Figure 7.2. a) precision-recall curve for the chimeric class only. b) mean precision-recall curve for all 4 classes. The blue graph represents the curve for semi-GAN, while the green one represents the curve for M1+M2. X-axis represents recall, while Y-axis represent precision.

The source code with all implementation detail is hosted on the GitHub repository: <https://github.com/tomislavsebrek/diplomski>.

7.1. Discussion

Based on the results presented in Table 7.3., we can notice that all semi-supervised models achieved better results when compared with the model that had only used deficient labeled set only. Therefore, we can draw a conclusion that unlabeled data in some way helps classifiers to make better predictions.

We can also notice that M1+M2 and semi-GAN outperform semi-AE regardless of the number of labeled examples that was used during the training, and based on the results in Table 7.3., we may infer that the semi-GAN shows slightly better performance when compared with the M1+M2 model. However, this difference diminishes as we increase the number of labeled examples, and due to the fact that both models have problems with detecting chimeric reads, we compared them based on their precision-recall curves.

Figure 7.2. shows that based on the area under the curve, we can infer that the semi-GAN outperforms the M1+M2 model, regardless of the chosen threshold for detecting chimeric reads.

Also, since omitting chimeric reads by classifying them as regular is a worse scenario than classifying a regular read as chimeric, we can decrease the threshold for detecting chimeric reads and position the classifier at the point of the precision-recall curve where the recall is greater than precision. That way more reads will be detected as chimeric, but only a small number of chimeric reads will be omitted.

Considering all presented results, it can be inferred that the semi-GAN is the best classifier for this problem, despite the fact that we used coverage graphs down-sampled to the length of 100 as its input. We tried different experiments, but they failed to deliver better results. Because of that and having in mind that the model which uses down-sampled signal of a shorter length will be computationally efficient at the test time, we proclaimed the semi-GAN as the optimal classifier.

Also, we want to emphasize that detecting reads as chimeric or repeat has not been studied before, so we cannot compare our results with other approaches. Furthermore, there is no guarantee that the created test set is challenging enough. Therefore, the results achieved by extending the assembly process are important

proof that we built a good classifier which can reduce the number of contigs and thereby facilitate the assembly process.

7.2. Visualization

Another guarantee that the training process went in the right direction is the visualization of the latent variables. Even though we drastically decreased the number of dimensions, the latent space is still too dimensional for visualization. Due to this fact, we used t-SNE, a machine learning algorithm for dimensionality reduction, for embedding the high-dimensional latent space into a space of two dimensions [15]. This visualization is performed over the test set and the results are shown in Figure 7.3.

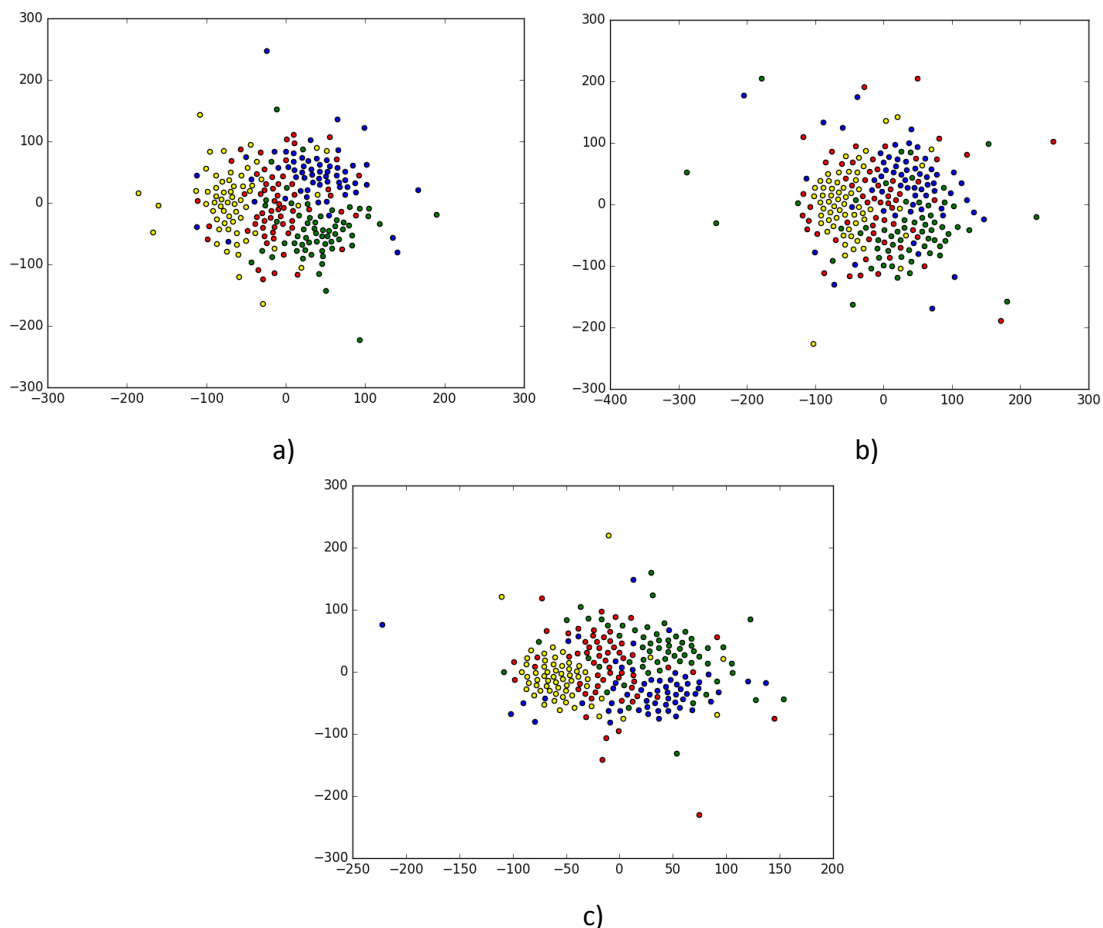


Figure 7.3. Visualization of the latent vectors of size 10 using t-SNE extracted by the a) semi-VA, b) M1+M2, c) semi-GAN. The red, green, blue and yellow circles represent examples from the dataset that were labeled as chimeric reads, left-repeats, right-repeats and regular reads, respectively.

Considering the visualizations shown in Figure 7.3, we can infer that the class groups associated with semi-GAN and semi-AE are more detached and coherent than the groups associated with the M1+M2 model. One of the reasons is the fact that semi-GAN and semi-AE both use extracted features as input in the classification part of the model, while the M1+M2 model's classification part is separated from the feature extraction, and is performed on the raw input.

Further on, in order to represent the idea of a two-dimensional manifold, we trained a variational autoencoder with the same parameters, but with the latent size 2. We then sampled a two-dimensional latent space and generated coverage graphs from it. We visualize the generated coverage graphs so that the graphs generated from latent vectors that are close in Euclidian space, remain topologically close in a manifold as well. This visualization is shown in Figure 7.4, and proves that high dimensional data can be described with a small number of features and the smooth transitions among coverage graphs confirm that the model can interpret data through the two-dimensional manifold.

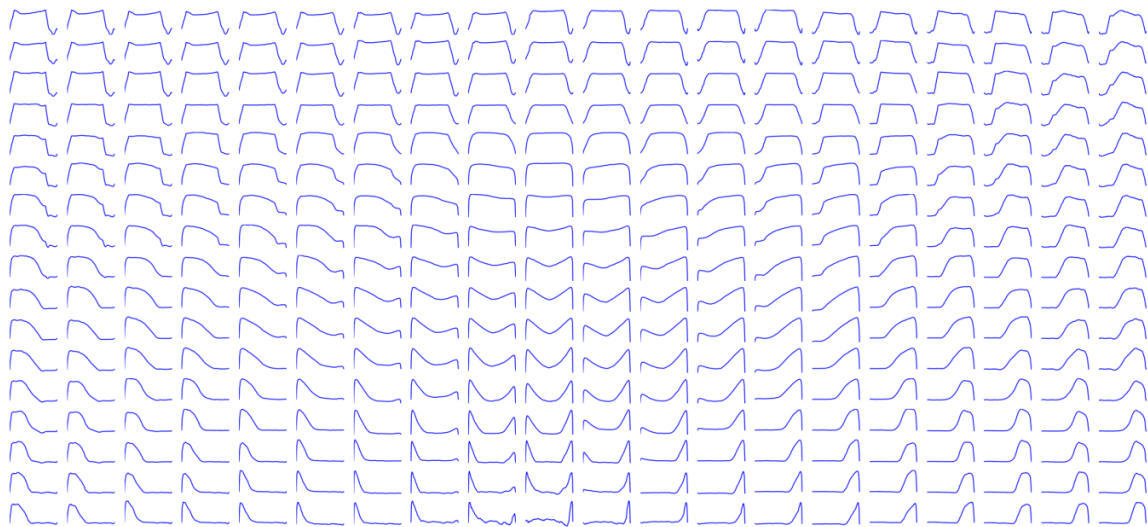


Figure 7.4. Visualization of the generated coverage graphs from 2-dimensional latent space. The axes represent equally spaced latent variable values between -3 and 3. It can be noticed that the coverage graphs placed in the center correspond with the chimeric reads, while the coverage graph placed on the edge corresponds to the left or right repeat.

In Figure 7.5 we compare the reconstruction of the input signals that are based on the regular and variational autoencoder having in mind that reconstruction, as well as feature extraction, is not the main purpose for which a variational autoencoder is used. It can be noticed that both models reconstruct the left repeat with adequate precision, but they both fail to cover the peak in the coverage graph of the chimeric read. However, chimeric reads can be recognized and classified correctly by a concave valley located around the peak in the coverage graph, even though it does not look very plausible.

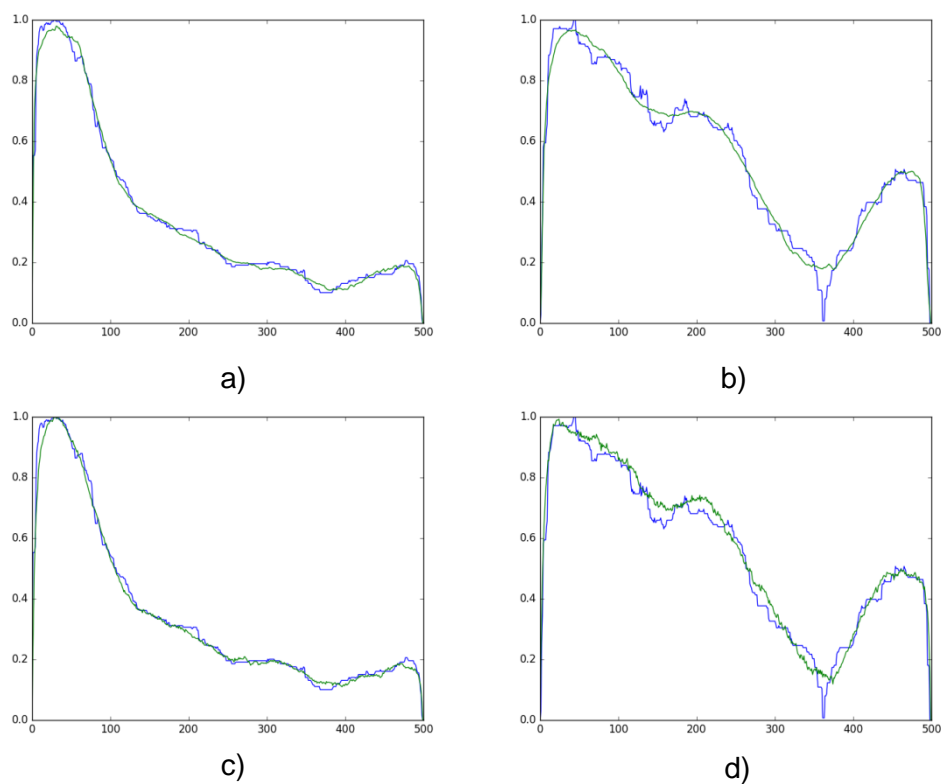


Figure 7.5. a) and b) original signal (blue) and its reconstruction (green) using a regular autoencoder. c) and d) original signal (blue) and its reconstruction (green) using a variational autoencoder.

Furthermore, we compare the examples generated from the variational autoencoder and generative adversarial network. This comparison is shown in Figure 7.6. It can be noticed that GAN can generate coverage graphs with more credible details while VAE tends to generate smooth coverage graphs, since it does not have a capacity to model sudden changes in the coverage graph.

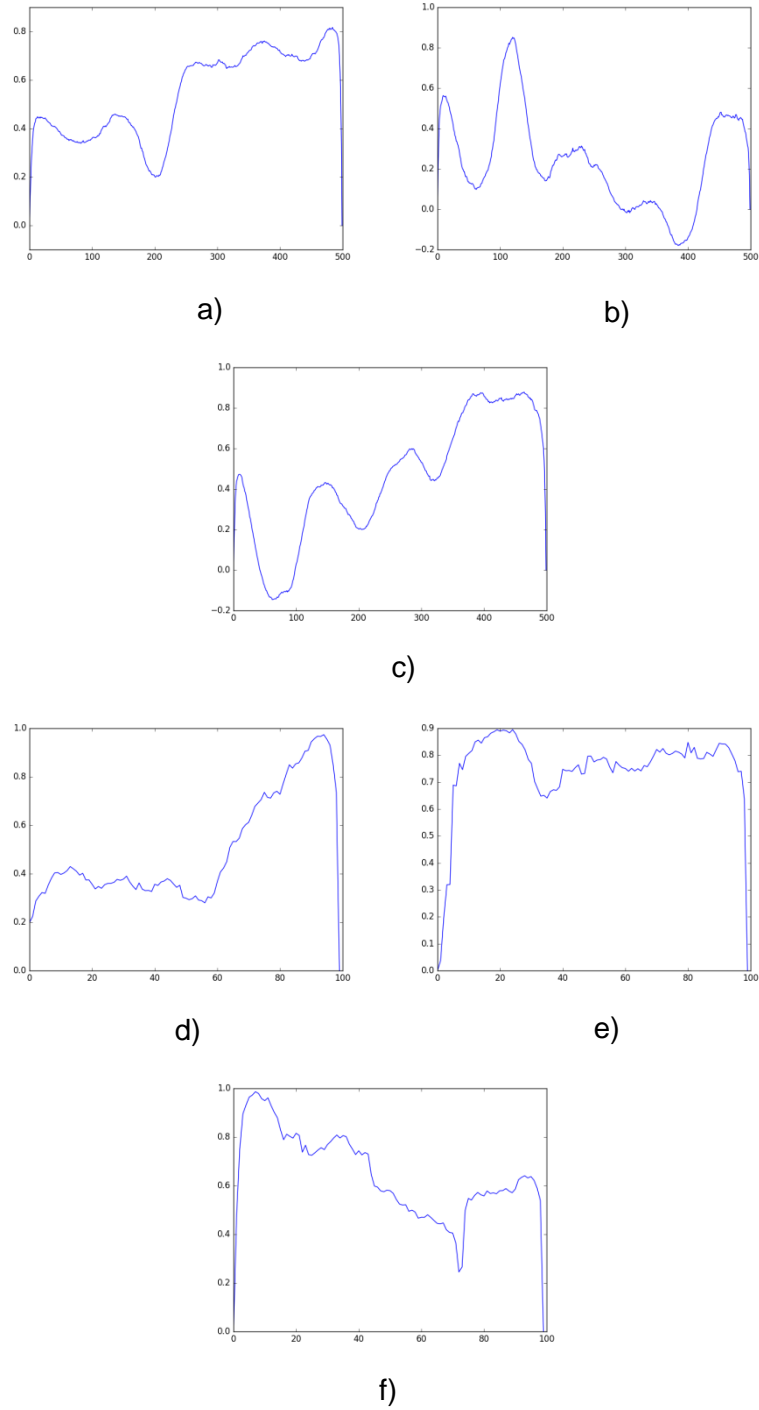


Figure 7.6. a), b) and c) coverage graphs generated by VAE. d), e) and f) coverage graphs generated by GAN. It can be noticed that the coverage graphs generated by VAE do not look as plausible as the ones generated by GAN.

Finally, we will show the generated samples from GAN with respect to the epoch in the training process in Figure 7.7. It can be noticed that as the training process approaches its end, the generator generates more plausible examples.

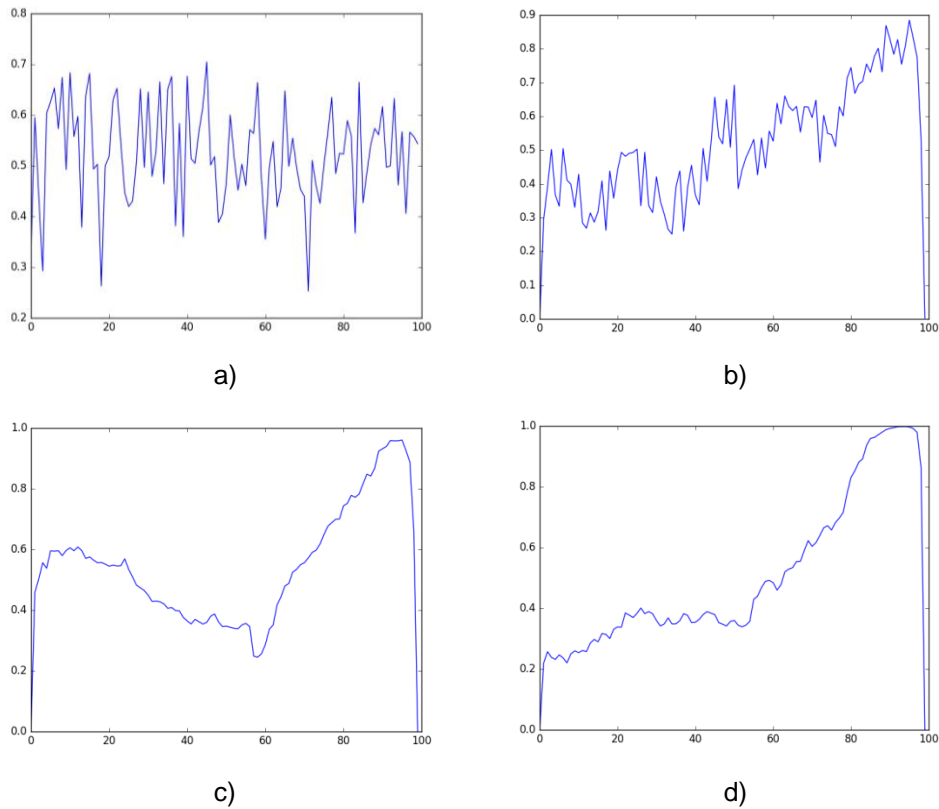


Figure 7.7. Coverage graphs presented by the generator in different phases of the training process. a) generated graph at the beginning of the training process looks more like random noise than a coverage graph. b) generated graph after the first epoch of the training algorithm starts to look like the right repeat due to the tendency of growth towards the right part. c) generated graph after 10 epoch of the training algorithm is smoother and contains conspicuous characteristics of the coverage graph associated with the right repeat, but its maximum is not reached in 1 as it is in all the real dataset examples, due to normalization. The discriminator can easily distinguish the real from fake examples based on that feature. d) generated graph at the end of the training process has the same maximum as any normalized graph so the discriminator needs to differentiate coverage graphs based on form rather than trying to trick the generator by checking if the coverage graph is normalized or not.

8. Conclusion

The main aim of this thesis was creating a method for classifying the coverage graph of a read as chimeric, repeat or regular. We implemented three models based on semi-supervised deep learning: a regular autoencoder, an M1+M2 model, and a semi-supervised generative adversarial network.

We have created a database and manually labeled some reads in order to evaluate all the models. Based on the results of the evaluation, we inferred that all semi-supervised models have achieved better results than a model with the similar capacity that used only the deficient set of labeled examples.

Among all the proposed methods, the model that is based on a semi-supervised generative adversarial network showed the best performance on the dataset that we created for the purposes of this thesis. Moreover, this method outperformed other models regardless of the number of labeled examples that were used during the training process.

We embedded this detection based on the semi-supervised generative adversarial network in the assembly process, and successfully reduced the complexity of the overall process. Based on this result and the results of the cross-validation, we can draw a conclusion that a semi-supervised generative adversarial network is well suited for coverage graph detection.

Nevertheless, we can still improve this work and the easiest way to do this is to collect more labeled data. On the other hand, a more challenging improvement would entail choosing a labeled set which would better reflect the data distribution.

9. Bibliography

- [1] Alpaydin E. Introduction to Machine Learning: Introduction. London: The MIT Press, 2004.
- [2] Alpaydin E. Introduction to Machine Learning: Supervised Learning. London: The MIT Press, 2004.
- [3] Goodfellow et al. Deep Learning: Deep Feedforward networks. London: The MIT Press, 2016.
- [4] Ioffe, Schegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Cornell University Library, 2015. Available: <https://arxiv.org/abs/1502.03167>
- [5] Chapelle et al. Semi-Supervised Learning: Introduction to Semi-Supervised learning. London: The MIT Press, 2006.
- [6] Goodfellow et al. Deep Learning: Autoencoders. London: The MIT Press, 2016.
- [7] Doersch C. Tutorial on Variational Autoencoders, Cornell University Library, 2016. Available: <https://arxiv.org/abs/1606.05908>
- [8] Durr, Introduction to variational autoencoders. 2016. Available: <https://home.zhaw.ch/~dueo/bbs/files/vae.pdf>
- [9] Kingma et al. Semi-supervised learning with deep generative models, Cornell University Library, 2014. Available: <https://arxiv.org/abs/1406.5298>
- [10] Kingma, Semi-supervised learning with deep generative models. Available: <http://musyoku.github.io/2016/07/02/semi-supervised-learning-with-deep-generative-models/>
- [11] Goodfellow et al. Generative adversarial networks, Cornell University Library, 2014. Available: <https://arxiv.org/abs/1406.2661>
- [12] Goodfellow I. NIPS 2016. Tutorial: Generative adversarial networks, Cornell University Library, 2016. Available: <https://arxiv.org/abs/1701.00160>

- [13] Odena A. Semi-Supervised Learning with Generative Adversarial Networks, Cornell University Library, 2016. Available: <https://arxiv.org/pdf/1606.01583.pdf>
- [14] Sovic I, Graphmap - A highly sensitive and accurate mapper for long, error prone reads. Available: <https://github.com/isovic/graphmap>
- [15] Šnajder J. Vrednovanje modela, IPythonNotebook, 2015. Available: <http://nbviewer.jupyter.org/github/jsnajder/StrojnoUcenje/blob/master/notebooks/SU-2015-10-VrednovanjeModela.ipynb>
- [16] van der Maaten L.J.P., Hinton, G.E., Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research. 9: 2579-2605, 2008. Available:<http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf>

Classification of 1D-Signal Types Using Semi-Supervised Deep Learning

Abstract

In this thesis, we proposed methods for detecting the type of the coverage graph based on semi-supervised deep learning models: autoencoders and generative adversarial networks. We evaluated the performance of each model based on the dataset that contained reads from multiple reference genomes. We have manually labeled some of the data and compared the results of all models with respect to the number of labeled examples that were used during the training process. We have embedded this detection in the assembly process and achieved good results. The source code is available at <https://github.com/tomislavsebrek/diplomski>.

Keywords: deep learning, autoencoder, generative adversarial network, semi-supervised learning, coverage graph, chimeric read, repeat read

Identifikacija tipova 1D-signalâ pomoću polu-nadziranog dubokog učenja

Sažetak

U okviru ovog rada, predložili smo metode za detekciju tipova grafova pokrivenosti na temelju polu-nadziranih modela dubokog učenja: autoenkodera i generativnih suparničkih mreža. Testirali smo performanse svih modela koristeći bazu koja sadržava očitavanja s više referentnih genoma. Ručno smo označili dio podataka i usporedili rezultate s obzirom na količinu označenih primjera koji su bili korišteni prilikom treniranja. Ovu detekciju smo ugradili u proces sastavljanja i dobili dobre rezultate. Izvorni kod je dostupan na <https://github.com/tomislavsebrek/diplomski>.

Ključne riječi: duboko učenje, autoenkoder, generativni suparnički model, polu-nadzirano učenje, graf pokrivenosti, kimerno očitavanje, ponavljajuće očitavanje