Application of neural networks in recognition of complex solutions

Kožić, Sven Benjamin

UNIVERSITY OF ZAGREB
FACULTY OF SCIENCE
DEPARTMENT OF PHYSICS

Sven Benjamin Kožić

APPLICATION OF NEURAL NETWORKS IN RECOGNITION OF COMPLEX SOLUTIONS

Master Thesis

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Sven Benjamin Kožić

UPOTREBA NEURALNIH MREŽA U RASPOZNAVANJU KOMPLEKSNIH OTOPINA

Diplomski rad

Zagreb, 2020.

UNIVERSITY OF ZAGREB
FACULTY OF SCIENCE
DEPARTMENT OF PHYSICS

INTEGRATED UNDERGRADUATE AND GRADUATE UNIVERSITY
PROGRAMME IN PHYSICS AND COMPUTER SCIENCE EDUCATION

Sven Benjamin Kožić

Master Thesis

Application of neural networks in recognition of complex solutions

Advisor: Vinko Zlatić, dr. sc.

Co-Advisor: Krešimir Kumerički, prof. dr. sc.

Master Thesis grade: _____

Committee: 1. _____

2. _____

3. _____

Master Thesis defence date: _____

Zagreb, year

Hvala mentoru dr. sc. Vinku Zlatiću na mogućnosti rada na ovoj temi, strpljenju i savjetima, te komentoru prof. dr. sc. Krešimiru Kumeričkom na podršci.
Zahvaljujem na svemu svojoj obitelji i dragoj Eni.

Upotreba neuralnih mreža u raspoznavanju kompleksnih otopina

Sažetak

Upotreba neuralnih mreža i strojnog učenja općenito trenutačno je vrlo relevantno i zanimljivo područje, i u ovom radu testirat ćemo primjenjivost takve metode u svrhu raspoznavanja smjesa Na^{+}, K^{+} i Ca^{2+} različitih koncentracija koja može predstavljati osnovu u istraživanju nekih drugih naprednijih neuromorfnih tema.
Ključne riječi: Neuralne mreže, strojno učenje, PCA, OECT

Application of neural networks in recognition of complex solutions

Abstract

Application of neural networks and machine learning in general is currently a very relevant and interesting field, and in this thesis we will test its applicability in recognition of blends of different Na^+, K^+, and Ca^{2+} concentrations which could provide basis in investigating other more advanced neuromorphic topics.

Keywords: Neural networks, machine learning, PCA, OECT

Contents

1 Introduction

1.1 From Machine Learning to Neuromorphic Engineering

The idea of artificial intelligence dates back to 1950s, but it would be useful to point out that although large strides of progress have been made, frequently used terms such as AI and "learning" don't (yet) really refer to human level intelligence in general sense. It is important to make that distinction and provide the appropriate definitions.

Chollet [1] defines learning, in the context of machine learning, as a search process of finding better representations of input data, while Mitchell [2] defines computer program learning as improvement of doing a task through experience, in reference to some performance measures.
Multiple machine learning models have been made that use different ideas to manipulate input data, from logistic regression, kernel methods to decision trees. As part of decision tree methods, random forest and boosting algorithms have become popular second-best algorithms applicable for a wide range of problems [1].

Early learning algorithms have been based on simulating the behavior of a neuron in the brain, called artificial neural networks, that have slowly evolved to become the field of deep learning. The issue that emerged at first was that the computer infrastructure wasn't advanced enough. But as the chips got smaller and faster, software more user-friendly and with the rediscovery of backpropagation algorithm in 1980s, neural networks have again resurfaced and were rebranded as early as 2006 by the term "deep learning" that surpasses past neuroscientific perspective [10].
Deep learning, as a subfield of machine learning, concentrates on building multiple levels of useful representation, using multilayered neural networks. In simpler terms "Deep learning allows the computer to build complex concepts out of simpler concepts" [10]. Again, it has to be noted that even though it was initially inspired by neurological principles, models used in deep learning don't represent the the behavior that occurs in the brain [1]. Nevertheless, the field has been proven to solve monumental tasks.

Convolutional neural networks have been successful in the task of image recognition while recurrent neural networks are used in speech recognition [9]. Some more examples include using deep learning techniques for autonomous driving [12] and even using neural networks to measure pressure inside of protons [13]. As shown, the development of machine learning and of deep learning as powerful tools show great promise in general application and solving of numerous problems.

In this master thesis we will test the applicability of machine learning, specifically artificial neural networks, in determining relative concentrations of different electrolyte compositions. This investigative effort can be seen as one of many stepping stones to further research of intelligent sensors that can "sense" different chemical or organic compositions and which could be used in various fields (such as in medicine) and environments.

An interdisciplinary concept of neuromorphic engineering has recently emerged that is inspired by biological principles [3] and its goal is to mimic neuro-biological architectures present in the nervous system and it seems that, for many problems, biological solutions appear more effective compared to traditional electronics. [4].

Researchers have used neuromorphic sensors and artificial intelligence in robotics to combine perception with motoric ability [5], have tested a scalable tactile glove that can learn tactile signatures of the human grasp [6], and have also invented prothesis enhanced with a multilayered electronic dermis that can help amputees perceive touch and pain to a level of determining curvature and sharpness of objects [7]. These are only a few of many possible uses of neuromorphic engineering.

The advancement in architectures capable of reproducing the organizing principles that guide biological neural systems could lead to significant impact in fields of bioinformatics, neuroscience and even brain-computer interfaces [8].

This convergence of electronics and biology is encapsulated by the field of bioelectronics, which is based on organic (electrochemical) materials that have better characteristics compared to inorganic materials. Some examples of organic transistors in neuromorphic and sensing applications include:

(a) Organic field effect transistor (OFET).

(b) Organic electrochemical transistor (OECT).

(c) Internal ion-gated organic electrochemical transistor (IGT).

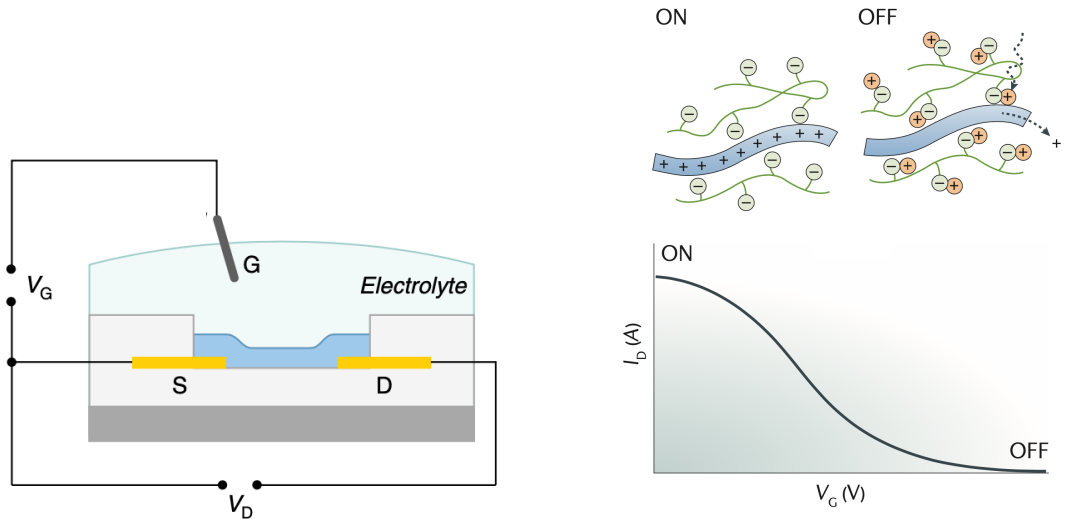
This kind of devices enable us to transform biological (ion- or chemical-based) signals into electronic signals [8].

1.2 Overview of Transistors

A Field Effect Transistor is a semiconductor device in the electrical circuit which uses the electric field to control the current. It is a *unipolar* device, meaning it uses only one type of charge carrier unlike the Bipolar Junction transistor, another common type of transistor.
In solid state physics we distinguish between two types of carriers, electrons and holes (which represent the absence of electrons in the valence band). Concentrations of electrons and holes are marked as n and p. An *intrinsic* semiconductor, usually silicon or germanium, has the same electron and hole concentrations.
Introducing impurities in the material we form *extrinsic* semiconductors, of which we have two types: Donors have excess electron carriers, refered to as n-type, while acceptors have a surplus of holes that can accept electrons, and we call them p-type impurities [11].

We can construct a Junction Field Effect Transistor (JFET) or a Metal Oxide Semiconductor (MOSFET). Terminals source S and the drain D represent the ohmic contact points from which the main carriers enter and exit the transistor. They are connected to two semiconductor bars that are both n-type or p-type, and the channel trough which the majority carriers travel that connects the source and the drain is named accordingly.
The gate G terminal controls the current through the channel and consists of impurity regions of the type contrary to the channel type. MOSFET differs from JFET because the gate is insulated from the main current with a thin coating layer of silicon dioxide.
Other than the JFET and MOSFET, there is a variety of FET types depending on the the device design and the properties of the carrier. The alternative architecture worth mentioning is the thin-film transistor TFT. The structure of TFT is well suited for the use of organic semiconductors [16].

1.3 Organic Electro-Chemical Transistor



(a) Typical OECT structure

(b) Depletion type transfer curve

Figure 1.1: **a** On the schematic of an OECT cross-section, gate (G), source (S) and drain (D) electrodes are labelled, and the organic semiconductor channel is marked with blue color. The device is immersed in the electrolyte solution. **b** Transfer curve $(I_D - V_G)$ for depletion-mode operation of an OECT is shown. At zero gate voltage the polymer channel conducts hole current and as the applied voltage increases, holes are replaced with cations [20].

Improvement in shaping and moulding of semiconducting material such as silicon has made MOSFETs present everywhere in our environment, from cellphones to computers. Rapid miniaturisation and scaling of MOSFETs has been observed in the form of Moore's law. The term organic electronics specifies the development of organic semiconductors that parallels the progress made in inorganic semiconductors [17]. Organic electronic materials replace the conventional inorganic conductors and semiconductors with organic molecules or polymers. Organic semiconductors represent low-cost, lightweight and flexible alternative to typical semiconductors [19].
Although organic semiconductors were known as early as the late 1940, industrial interest in OTFTs and their application didn't start until the more recent decades when improved mobility[1] of organic semiconductors was comparable to that of amorphous silicon [16].

[1]Specifically hole mobility with values exceeding 1.0 cm^2/Vs, as electron mobility is much lower than that of typical semiconductors [19].

Organic materials have a wide range of interesting properties such as configuring molecules to fit specific requirements (tunability) through synthetic chemistry and a potential for low cost production through large area printing enabled by ease of applying polymer coatings on a variety of substrates [17].

Organic electrochemical transistors (OECTs) belong to the subset of OTFTs [18], in which the thin film that is made of organic active layer is approximately two-dimensional. First OECT has been developed in 1984. by White et al. demostrating the modulation of polypyrrole film conductivity in application of gate voltage through electrolyte [15]. OECTs are similar to OFETs and they share properties, form factors and materials [20]. The difference comes from the fact that while OFET replaces the semiconductor channel with an organic semiconductor film that is separated from the gate electrode by the gate dielectric [22], an OECT has the organic film in contact with an electrolyte[2] which enables ions in electrolyte to penetrate into the semiconductor active layer, changing the conductivity and making the electrolyte integral to device behavior.

Gate is immersed in the electrolyte and source and drain electrodes provide the contact the with the channel. The on/off switch is produced as ions are injected from the electrolyte into the organic film controlled by the gate voltage V_G. The conductivity of the channel is then changed based on the doping state of the organic semiconductor and the drain current I_D is induced by the drain voltage V_D.

A p-type semiconductor material that is commonly used in the OECT channel is the poly(3,4-ethylenedioxythiophene) doped with poly(styrenesulfonate) abbreviated as PEDOT:PSS. The structure of PEDOT:PSS is shown on figure 1.2. PEDOT:PSS maintains good electrochemical stability in aqueous electrolytes, exhibits high electrical conductivity, and can be easily printed on a variety of substrates and manufactured to have transconductance in order of millisiemens and response time in microseconds [22] [20].

[2]An extreme case of OFET, the Electrolyte-gated OFET (EGOFET) is also immersed in the electrolyte and looks similar to an OECT, but there is no electron transfer from the electrolyte and the organic semiconductor, only capacitive processes occur [21].

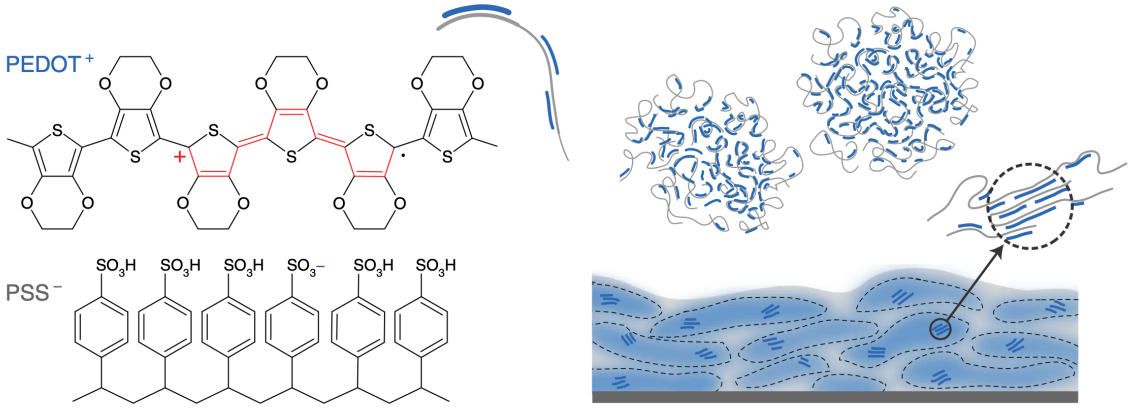


Figure 1.2: **Polyelectrolyte-doped, PEDOT:PSS**
On the left is the chemical structure of PEDOT:PSS and on the right, picture represents the synthesis and the resulting film with PEDOT:PSS- rich (blue) and PSS-rich (grey) phases. Figure taken from [23]

OECTs can operate in both accumulation and depletion mode but most research papers deal with depletion mode [18]. Because of PEDOT:PSS high conductivity, channel and the electrodes can be made out of the same material, offering simplification and versatility of fabrication process and making the transistor normally ON device [24]. The charge carriers in the drain current are holes, which in the depletion-mode OECT traverse the channel in the absence of the gate voltage (ON state). They are compensated as the ionized acceptors, that are the sulfonate anions of PSS, are injected into the channel under the application of the positive gate bias (OFF state). See figure 1.1.

Important electrical characteristic for transistors is the value of transconductance $g_m = \frac{\partial I_D}{\partial V_G}$ as it describes the conversion of modulation in gate voltage to modulation in drain current. The value of g_m determines the function of the device as an amplifier, as it is the direct measure of effective signal amplification of a single OECT and shows if the device can operate to transduce small biological signals, justifying its sensing capabilities [25]. Compared to transconductance of other electrolyte and ionic liquid-gated transistors and even solid state devices, OECT outperforms them and is only surpassed by III-V semiconductor bulk devices [26].

PEDOT:PSS structure is complex and ion injection and transport in organic semiconductors has yet to be sufficiently described and because of that simple models have

been used to determine transistor behaviour such as the much referenced Bernards model [18] that uses MOSFET physics as its basis [27] [20]. Bernards model successfully emulates OECT output and quantitatively predicts the value of transconductance g_m.
In the Bernards model, OECT transistor is described in the form of two circuits, electronic circuit representing hole transport through organic semiconductor channel and the ionic circuit of the ion flow through the electrolyte.

In case of the electronic circuit, Ohm's law is used, as the p-type organic semiconductor film transports holes from source to drain electrodes under the influence of local potential and is treated as a resistor in a MOSFET fashion [20] shown in equation:

$$J(x) = q\mu p(x)\frac{dV(x)}{dx} \tag{1.1}$$

where J represents the current as a function of position x in the channel, q is elementary charge, μ is the hole mobility treated as constant, p is the hole density and $\frac{dV(x)}{dx}$ is the electric field.

Ionic circuit is formed by combining into series, a resistor R_s that describes the electrolyte conductivity i.e. flow of ions and a capacitor C_d describing the polarization of the electrolyte at the contact with organic film and with the gate electrode and as the storage of ions in the channel.
The model doesn't delve into the specific of ion injection but rather describes the process electrostatically with no electrochemical reactions between the electrolyte and the organic film.

Steady state occurs as the capacitor is charged and the gate current goes to zero [20]. Transient response is exhibited in the form of a charging capacitor:

$$Q(t) = Q_{ss}\left[1 - e^{-t/\tau_i}\right], Q_{ss} = C_d\Delta V \tag{1.2}$$

$\tau_i = C_d R_s$ is the ionic transit time and the Q_{ss} notes the total charge passing through the ionic circuit from the voltage applied across electrolyte.

Amount of charge in the organic film is expressed as:

$$Q(x) = c_d W dx(V_g - V_x) \tag{1.3}$$

Where W is the width of the conducting polymer film, V_g is the gate voltage and $V(x)$ notes the voltage dependency on the position in the channel. It is more useful to write capacitance per unit area $c_d = C_d/A$ in this context.

Using these equations the current function in the channel at the steady state can be formulated as:

$$J(x) = q\mu p_0 \left[1 - \frac{V_g - V(x)}{V_p}\right] \frac{dV(x)}{dx} \tag{1.4}$$

p_0 is the initial hole density in the organic semiconductor before the application of a gate voltage and $V_p \equiv qp_0h/c_d$ is the pinch-off voltage, h denoting channel height. The transconductance in the saturation regime is then given as:

$$g_m = \frac{Wd}{L}\mu C^*(V_{threshold} - V_G) \tag{1.5}$$

where width W, length L and thickness d describe the channel geometry, μ is the charge-carrier mobility, and C^* is the capacitance per unit volume of the channel.

The equation 3.3 links to FET behaviour in the case of limiting the channel thickness [25]. High transconductance of OECTs comes from the volumetric nature of their response. Important characteristic of OECTs that differentiates them from other OTFTs is that doping occurs over the entire volume which makes the channel volume affect the transistor performance in terms of amplification. That in turn allows low-gate voltages to make large modulations in the drain current [20].

Inal at al. highlight the μC^* product of electronic mobility and volumetric charge capacity as an important value that captures the mixed ionic-electronic properties of OECT channel material [25]. On the other hand, high transconductance limits the response time. The thickness of the channel is proportional to the reaction time of an OECT. Khodagholy at al. have shown that frequency response of the OECT is dominated by ion transport [26].

Compared to organic field-effect transistors (OFETs), OECTs have higher transconductance up to a certain frequency and that suggests that they could replace OFETs when it comes to application requirement in range of 10 kilohertz depending on the type of electrolyte and specifics of the design such as choices regarding the organic channel [20]. This requirement is adequate for quasi-static biosensor application, such as extracellular signals in order of 100mV that can be discerned at the frequency of action potentials of 1kHz, enabling the interface with active cells and tissues [26].

Khodagholy at al. [26] have also shown that OECTs have resistance to aggressive mechanical deformation by crumpling and uncrumpling the devices peeled-of a glass substrates and testing to confirm no change in transconductance or time response. OECTs can therefore be used in malleable materials such as textiles, as an example, for detection of analytes in sweat [20]. OECTs can have a wide range of applications from recording brain activity to measuring concentrations of glucose, dopamine, penicillin and other complex biochemicals and for use in neuromorphic engineering [15].

2 Cation Measurements with Organic Electrochemical Transistors

2.1 Cation Discrimination by Dual Frequency Sensing

Pecqueur et al. [29] have previously confirmed the operability of the OECTs on a broad range of cations. The research was based on the previous knowledge of frequency-dependant behaviour of PEDOT:PSS/electrolyte interfaces. Under steady state regime ions of the specific configuration accumulate in the bulk of PEDOT:PSS and modulate the channel conductance, while ionic currents dominate drain current response in the transient regime [28]. These two different regimes are ion concentration dependent and influence transistor's impedance at different frequencies such as channel dedoping on low frequencies and the electrolyte gate capacitive coupling at high frequencies.

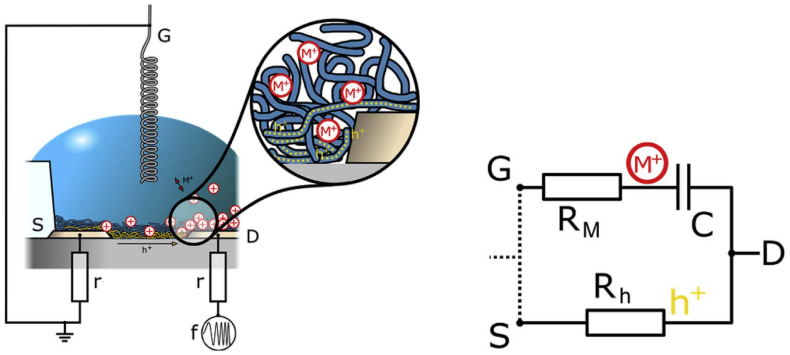


Figure 2.1: Scheme of the device cross section and the simplified equivalent circuit. M+ label for cations, h+ for holes. Figure is taken from [29].

As in the Bernards model, OECT can be described as consisting of two circuits, electronic and ionic one, with one resistor accounting for ion conduction in the polymer and hole transfer, a second resistor to account for the ion dependent conductivity of the electrolyte and the electrolyte/gate interface replaced by a geometric capacitor. A simple "2R1C" model of the OECT as an equivalent parallel circuit of one resistor with a serial resistor and capacitor was then used to mimic the transistor frequency and transient response [3].

[3]The non-idealities of such a simple model were addressed in [30].

Using impedance spectroscopy to measure behaviour of different metal chloride salts (immersed in the electrolyte) on the device has shown that the values of impedance plateaus at low and high frequencies depend on electrolyte nature. The values of those plateaus also change depending on the concentration of solution; for low frequency impedance value $|Z|_{low}$ increases with concentration, while the impedance value $|Z|_{high}$ decreases with increase of concentration on high frequencies. Paired with previous notion of impedance dependency on electrolyte natures suggested two uncorrelated ion-related outputs that can be used in differentiation of cation concentrations. Therefore, both $\Delta\,|Z|_{low}$ and $\Delta\,|Z|_{high}$ can serve to identifying cation's nature. The above mentioned paper also reported on an interesting relation between the relative position of the impedance modulus modulation points with the plot of cation radius versus electrolyte conductivity.

These obtained results motivated a more practical approach for dynamic cation detection, namely application of digital voltage inputs to give analog current modulated output. Exposing the salts to a square-pulsed gate voltage V_G at both low and high frequency and measuring drain current I_D, different current modulations ΔI could be recorded and then used for cation discrimination.

2.2 OECT Data Sequencing

This brings us to the task at hand in this work which was to distinguish relative concentrations of 25 different NaCl-KCl-$CaCl_2$ electrolyte compositions.

A single OECT device was exposed to electrolytes of different configurations; namely calcium, sodium and potassium ions in blends of different relative concentrations. Periodically transient currents, ion characteristics and applied voltages were recorded. In order to exclude systematic deviations in the data, a single device was used and the measurements were made in a single time period.

Two voltage stimulations at $V_{G_1} = 100$ mV and $V_{G_2} = 350$ mV lasting 0.5 ms were applied, separated by a resting period of the same duration ($V_G = 0$ mV).

Measurement duration of 200 ms secured collection of 99 current sequences for a single concentration of specific cation type.

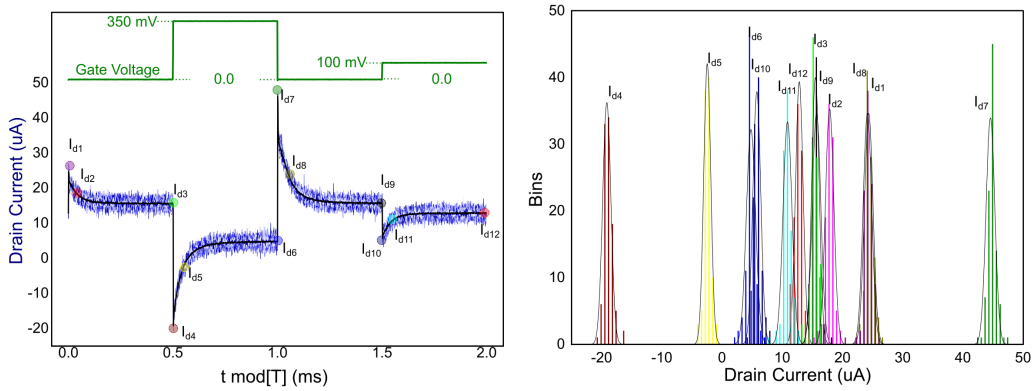


Figure 2.2: a) 2-ms signal sampling of the drain current in a single 200-ms measurement of 0.1 M KCl_{aq}. b) Statistic distributions of the 12 characteristic values taken from the drain current signal of a single 200-ms measurement in 0.1 M KCl_{aq}. Figure is taken from [28].

Figure 2.2 a) shows the single measurement on a 0.1 M KCl_{aq} sample. As discussed in [18], approach to steady state in a form of a spike and recovery of current indicates that hole transport in the organic film occurs at a relatively slow rate and the transient current is dominated by hole extraction from the film. This behaviour can be observed in drain current exponential growth after the initiation of positive gate voltage polarization, and decay afterwards.

In order to specify the cations from the data, relevant information descriptors needed to be determined to be fed into the subsequent deep learning algorithms. Paper [28] suggests taking characteristic values from the drain current, such as the 12 specific current values signifying distinct points of spikes, transients and steady state plateaus. Statistical distribution of those values is shown in 2.2 b) where relative separation and symmetry around a mean value can be noted. To alleviate the input data, the number of descriptors was reduced down to six current modulations.

$\Delta I_{\text{std}}^{100} = I_{d3} - I_{d12}$, as the steady-state drain current modulation at V_G = 100 mV.

$\Delta I_{\text{trs}}^{100} = I_{d2} - I_{d11}$, as the 50 μs transient drain current modulation at V_G = 100 mV.

$\Delta I_{\text{spk}}^{100} = I_{d1} - I_{d10}$, as the 1 μs spike drain current modulation at V_G = 100 mV.

$\Delta I_{\text{std}}^{350} = I_{d9} - I_{d6}$, as the steady-state drain current modulation at V_G = 350 mV.

$\Delta I_{\text{trs}}^{350} = I_{d8} - I_{d5}$, as the 50 μs transient drain current modulation at V_G = 350 mV.

$\Delta I_{\text{spk}}^{350} = I_{d7} - I_{d4}$, as the 1 μs spike drain current modulation at V_G = 350 mV.

In order to test the sufficiency of these values for the recognition of cation concentrations, multivariate data analysis was performed.

3 Principal Component Analysis

A simple and convenient way of testing the separation of data points is the principal component analysis (PCA). It belongs to unsupervised learning algorithms subset of machine learning and focuses on transformations of dataset to create a new more interpretable representation of data in service of visualization or compression of data [31].

Data consists of 25 electrolytes with series of 99 vectors of six components or in the context of machine learning, 6 features, for the total of 2475 $\mathbb{R}^6$ vectors. Those vectors can be visualised as points in a 6-dimensional space. Principal component analysis can facilitate the representation of high dimensional data by dimensionality reduction in which we reduce the dimensions of a d-dimensional dataset by projecting it onto a $k < d$ dimensional subspace while retaining most of the information. PCA method can be described as a rotation of dataset so as to uncorrelate the datapoints by their features, and reduce the number of features that don't provide sufficient information.

First of all, the algorithm seeks the axis of the larges variance that is noted as the first principle component PC1 which represents the direction (vector without orientation) in which the data contains the most informations. The direction in which the features are reciprocally most correlated.
After that the algorithms seeks the principal component that is perpendicular to the previous ones, until it finds all the mutual features of the dataset. Directions that are found are called principal components and determine the principal variance direction in the data.

PCA algorithm is as follows:
1. Standardize and scale the data.
2. Calculate eigenvectors and eigenvalues from the covariance matrix.
3. Sort the eigenvalues in descending order and choose k eigenvectors that belong to k largest eigenvalues, where k is the dimension of the new feature subspace ($k \leq d$)
4. Construct of projection matrix W from k chosen eigenvectors.

5. Transform dataset X using W resulting in k dimensional subspace of features Y.

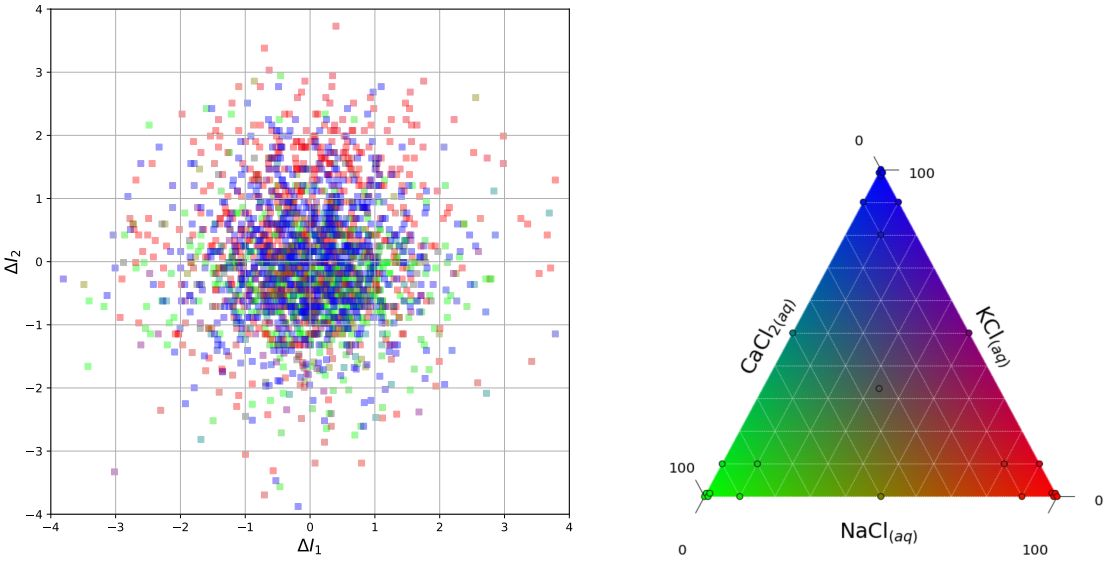
Let $\boldsymbol{x}$ be a d-dimensional random vector and $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ be n independent copies of $\boldsymbol{x}$. A single input vector denoted as $\boldsymbol{x}_{(i)} = (x_1, x_2, x_3, x_4, x_5, x_6)_i^T \in \mathbb{R}^n, n = 6$ stands for a single observation of features.
All the observations form a data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}, m = 2745$:

$$\boldsymbol{X} = \begin{pmatrix} (x_{(1)})^T \\ ... \\ (x_{(m)})^T \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} \\ ... & ... & ... & ... & ... & ... \\ x_{m,1} & x_{m,2} & x_{m,3} & x_{m,4} & x_{m,5} & x_{m,6} \end{pmatrix} \tag{3.1}$$

First step is to preprocess and scale the data by transforming to mean value $\bar{x} = 0$ and variance $s^2 = 1$. Standardization is done by substracting the mean and dividing by sample standard deviation.

$$\tilde{x}_{i,j} \leftarrow \frac{x_{i,j} - \bar{x}_j}{s_j}, \qquad \bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}, \qquad s_j = \sqrt{\frac{1}{n}(x_{i,j} - \bar{x}_j)} \tag{3.2}$$

where j is specifying the feature column. Most of the time we do not know μ (the population average) and we estimate it with $\bar{x}$ (the sample average).



(a) Dataset plot of components ΔI_1 and ΔI_2.

(b) Ternary diagram of the color scheme.

Figure 3.3: **a** Components equal $\Delta I_1 = I_{d1} - I_{d2}$ and $\Delta I_2 = I_{d3} - I_{d4}$. **b** Concentrations of solutions are represented as points on the ternary diagram.

Standardised dataset is shown in 3.3 in a plot of two features, namely ΔI_1 and ΔI_2, where data points are color-coded to the cation type and concentration where $CaCl_2$, KCl and NaCl are represented in green, blue and red gradients, respectively. The concentrations used in measurements were mostly consisting of larger percentage of one cation and so, for example pure green dots show the solutions with high concentrations of $CaCl_2$, while the mixture of equal KCl and NaCl concentrations will appear on the plot as a violet dot. As it can be seen, it is difficult to discern useful information in the cluster such as 3.3 (a).

Consider variance along a direction $\boldsymbol{u}$ of all data points.

$$\begin{aligned} \mathrm{Var}(\boldsymbol{u}) &= \frac{1}{n}\sum_{\boldsymbol{x}} \left\|(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \cdot \boldsymbol{u}\right\|^2 \\ &= \frac{1}{n}\sum_{\boldsymbol{x}} \boldsymbol{u}^T(\boldsymbol{x}-\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})^T\boldsymbol{u} \\ &= \boldsymbol{u}^T\left[\frac{1}{n}\sum_{\boldsymbol{x}}(\boldsymbol{x}-\bar{\boldsymbol{x}})(\boldsymbol{x}-\bar{\boldsymbol{x}})^T\right]\boldsymbol{u} \\ &= \boldsymbol{u}^T\ \boldsymbol{\Sigma}(\boldsymbol{x})\boldsymbol{u} \end{aligned} \tag{3.3}$$

$\boldsymbol{\Sigma}$ is an empirical covariance matrix [32] where each element represents the covariance between two features.

If $\boldsymbol{u} \in \mathbb{R}^d$ then $\boldsymbol{u}^T\boldsymbol{\Sigma}\boldsymbol{u}$ is the variance of $\boldsymbol{u}^T\boldsymbol{x}$ and $\boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u}$ is the sample variance of $\boldsymbol{u}^T\boldsymbol{x}_1, ..., \boldsymbol{u}^T\boldsymbol{x}_n$. The sample variance measures how spread (i.e. diverse) the points are in direction $\boldsymbol{u}$.

Empirical covariance of $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ is the matrix $\boldsymbol{S} = \{s_{j,k}\}_{j,k=1,...,d}$ where $s_{j,k}$ is the empirical covariance of the two features $x_{i,j}$ and $x_{i,k}, i = 1, ..., n$.

$$\boldsymbol{S} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i-\bar{\boldsymbol{x}})(\boldsymbol{x}_i-\bar{\boldsymbol{x}})^T, \quad \bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i = \bar{\boldsymbol{x}}_1, ..., \bar{\boldsymbol{x}}_6 \tag{3.4}$$

Mean vector $\bar{\boldsymbol{x}}$ is a d dimensional vector with mean values of individual columns of dataset features. Because data matrix $\boldsymbol{X}$ was already standardized, the covariance matrix of the data is actually Pearson correlation matrix: correlation is covariance for standardized variables.

$$\boldsymbol{S} = \frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X} \equiv \boldsymbol{R} \tag{3.5}$$

Next step is to compute the decomposition of real symmetric covariance matrix $\boldsymbol{S} \in \mathbb{R}^{d \times d}$ to obtain eigenvalues and eigenvectors: + where $\boldsymbol{E} = Diag(e_1, ..., e_d)$ consists of eigenvalues of $\boldsymbol{S}$ and $\boldsymbol{V} = (\boldsymbol{v}_1, ..., \boldsymbol{v}_d)$ is an orthogonal matrix and columns of $\boldsymbol{V}$ are the corresponding eigenvectors of $\boldsymbol{S}$. For $\boldsymbol{V}$ to be an orthogonal matrix, it must be true that $\forall i = 1, ..., d, \|v_i\|_2 = 1$ and $\forall j \neq k, \boldsymbol{v}_j \cdot \boldsymbol{v}_j = 0$.
The values of empirical covariance matrix $\boldsymbol{E}$ are variances of $\left\{\boldsymbol{V}^T \boldsymbol{x}_i\right\}_{i=1,...,n}$, for example e_j is the variance of $\boldsymbol{v}_j^T \boldsymbol{x}_i$ measuring the spread of the "cloud" of data points $\{\boldsymbol{x}_i\}$ in the direction of $\boldsymbol{v}_j$.

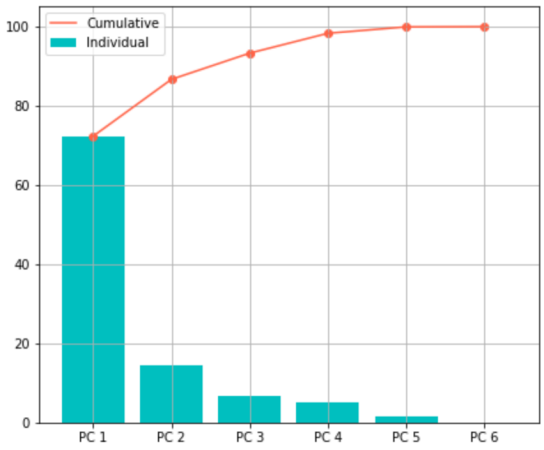


Figure 3.4: Variance of principal components.

To perform dimensionality reduction without losing too much information we can sort eigenvalues in a descending order and choose k eigenvectors corresponding to the largest k eigenvalues. Eigenvectors obtained from the procedure show the direction in the space $k \leq d$ and can be thought of as unit vectors of the new transformed "principle" axis. On the figure 3.4 we can see the percentage of each corresponding eigenvalue compared to their sum, i.e. the variance of the principle components.

$$\boldsymbol{V}_k = (\boldsymbol{v}_1, ..., \boldsymbol{v}_k) \in \mathbb{R}^{d \times k} \tag{3.6}$$

Projection matrix $\boldsymbol{V}$ is constructed by selection of k eigenvectors, and the resulting

output subspace $\boldsymbol{Y}$ can be calculated as:

$$\hat{\boldsymbol{Y}} = \boldsymbol{X} \cdot \boldsymbol{V} \tag{3.7}$$

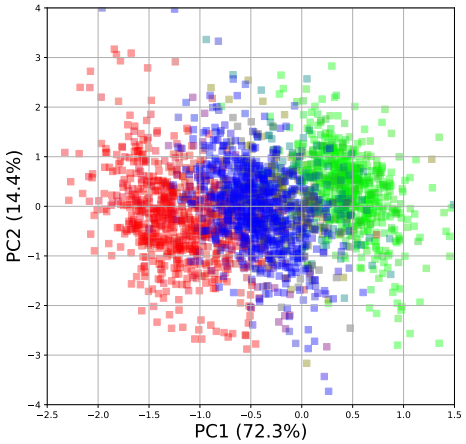


Figure 3.5: The product of Principal Component Analysis on data showing first two principle components.

The results of the procedure can be seen on figure 3.5. Compared to 3.3 (a), performing PCA on the data has achieved apparent separability of data points. Transformed variables values corresponding to original data points are called components scores, and sometimes factor scores.

3.1 Remarks

Few points from the last section needed to be addressed. Features of dataset are measured in μA and that represents a problem. The scale of the units at 10^{-6} makes the computations more difficult and it would be useful to adjust the mean and variance of the data, and for that reason some sort of feature scaling should be performed. Principle component analysis is very sensitive to scaling, and results differ depending on the method. There are multiple different approaches but the default go to methods are rescaling via min-max normalisation or standardisation.

The later consists of determining the distribution mean and standard deviation of each feature and using previously stated equation 3.2 to calculate new values of datapoints. Standardisation is used to scale the data into a distribution close to normal.

Performing PCA on the standardised data and ploting first two principle components is shown on figure 3.6.

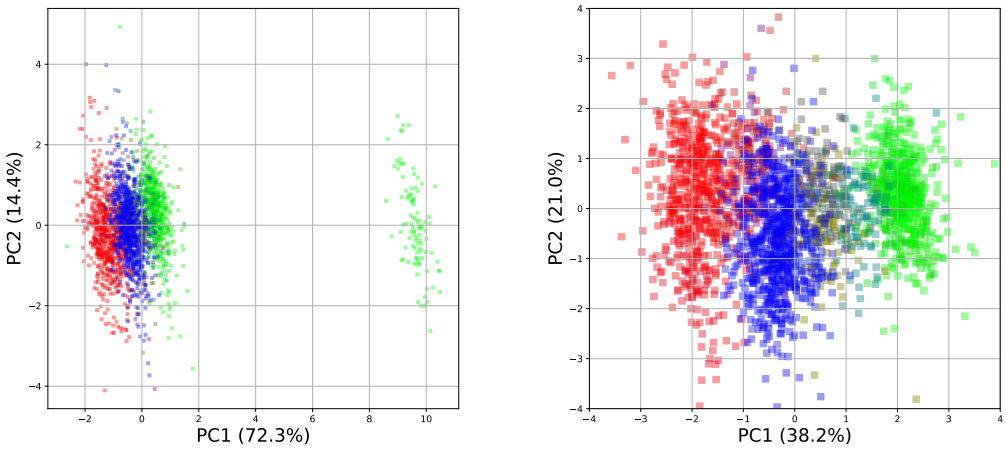


Figure 3.6: Results of performing PCA on full data left, and on reduced data right.

This plot is the same as in figure 3.5 but without constraining limits on the abscissa. As it is visible on the left part of figure, there exist outlier data points far removed from the cluster. Those points can be traced back to 99 measurements made on class $\boldsymbol{y}_9 = (0.01, 0, 0.99)$. These deviations can be attributed to some kind of experimental error and therefore removing those points from the dataset can be considered. The right part of the figure also shows first two principle components but on the reduced dataset and the amount of explained variance is now changed.

Loadings

Consider eigendecomposition of the symmetric covariance matrix S.

$$\boldsymbol{S} = \frac{1}{N-1}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{V}\boldsymbol{E}\boldsymbol{V}^T \tag{3.8}$$

The columns of $\boldsymbol{V}$ are the eigenvectors showing the directions of maximal variance of data, and $\boldsymbol{E}$ is a diagonal matrix with corresponding eigenvalues, the values of variance for each direction. Other option is to perform Singular Value Decomposition (SVD) of the centered data matrix $\boldsymbol{X}$.

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T \tag{3.9}$$

The columns of $\boldsymbol{V}$ are known as the right-singular vectors equal to eigenvectors obtained from decomposition of covariance matrix $\boldsymbol{S}$.The elements along the diagonal of $\boldsymbol{D}$ are known as the singular values of the matrix $\boldsymbol{X}$ and they are related to eigen-

values of covariance matrix as:

$$\boldsymbol{E} = \frac{\boldsymbol{D}^2}{N-1} \tag{3.10}$$

Columns of $\boldsymbol{U}$ are the left singular vectors:

$$\frac{\boldsymbol{XV}}{\sqrt{\boldsymbol{E}}} = \sqrt{(N-1)}\boldsymbol{U} \tag{3.11}$$

The results of PCA are usually discussed in form of *principal components* noted by $\hat{\boldsymbol{Y}}$ (also known as *PC scores*).

$$\hat{\boldsymbol{Y}} = \boldsymbol{X} \cdot \boldsymbol{V} = \boldsymbol{U} \cdot \boldsymbol{D} \tag{3.12}$$

Eigenvectors $\boldsymbol{V}$ determine the *principal axes* (the directions of maximal variance of data), and they can be viewed as weights by which the original variables $\boldsymbol{X}$ should be multiplied to project them onto those axes to produce *principal components*. This procedure is equivalent to taking dot product $\boldsymbol{U} \cdot \boldsymbol{D}$.

Loadings in PCA are principal axes scaled by the square roots of the respective eigenvalues.It should be noted that sometimes eigenvectors and loadings are used interchangeably, but that is not advised. Loadings matrix is given as:

$$\boldsymbol{L} = \boldsymbol{V}\frac{\boldsymbol{D}}{\sqrt{N-1}} = \boldsymbol{V}\sqrt{\boldsymbol{E}} \tag{3.13}$$

The variance of each PC equals the corresponding eigenvalue, and by scaling them with a square root of the eigenvalue, the variance of PCs are standardized to 1 and allow values in loadings matrix to be interpreted as correlations between the original variables and the principal components. Phrased through the cross-covariance matrix with $\boldsymbol{X}$ as the original dataset and $\tilde{\boldsymbol{Y}}$ as the standardized principal components:

$$Cov(\boldsymbol{X}, \tilde{\boldsymbol{Y}}) = \frac{\boldsymbol{X}^T\tilde{\boldsymbol{Y}}}{N-1} = \frac{\boldsymbol{X}^T(\sqrt{N-1}\boldsymbol{U})}{N-1} = \frac{\boldsymbol{DVU^TU}}{\sqrt{N-1}} = \boldsymbol{V}\sqrt{\boldsymbol{E}} = \boldsymbol{L} \tag{3.14}$$

Where standardization of PCs can be either by multipying the matrix of left singular vectors $\boldsymbol{U}$ by a factor of $\sqrt{N-1}$ or by scaling the dot product $\boldsymbol{X} \cdot \boldsymbol{V}$ with the square root of eigenvalue matrix $\boldsymbol{E}$ as shown in equation (3.11).

3.2 PCA Results

	PC1	PC2	PC3	PC4	PC5	PC6
	38.2%	21.%	16.2%	13.9%	10.6%	0.1%
$\Delta I_{\mathrm{std}}^{100}$	0.09	-0.38	0.9	0.2	0.07	0.
$\Delta I_{\mathrm{std}}^{350}$	-0.39	-0.55	-0.03	-0.66	0.33	0.
$\Delta I_{\mathrm{trs}}^{100}$	0.53	-0.39	-0.33	0.45	0.51	-0.
$\Delta I_{\mathrm{trs}}^{350}$	0.17	-0.81	-0.23	0.08	-0.51	0.
$\Delta I_{\mathrm{spk}}^{100}$	0.95	0.09	0.06	-0.27	-0.03	0.06
$\Delta I_{\mathrm{spk}}^{350}$	0.95	0.08	0.06	-0.28	-0.03	-0.06

Table 3.1: Factor loadings matrix for the PCA exploiting the six different current descriptors.

First Row of 3.1 show the proportion of overall variance explained. As expected for PCA, first principle components PC1 selects the direction of the maximal variance, in this particular case, of 38.2%. Significant portion of total variance ($\sim$ 60%) can be projected to PC1;PC2 plane. PC6 shows negligible variance ratio of 0.1% indicating some form of dimensionality reduction but more careful consideration of importance of six descriptors show that all of them are necessary for identifying cations [28]. Later used neural networks benefit from all available information.

Each entry of the matrix contains loading factors representing the correlation between the original variable and the principle component. $\Delta I_{\mathrm{spk}}^{100}$ and $\Delta I_{\mathrm{spk}}^{350}$ correlate strongly with PC1 and they are the only ones to contribute to PC6 by a very small factor, while others don't play any role in explaining the variation on PC6.
The negative loadings represent negative correlation among the variables, for example all steady state and transient variables have negative contribution to PC2, meaning that to get component scores from original dataset, they should be multiplied by negative weights to transform them onto PC2. Because all six descriptors influence first two principle components, the data can be meaningfully considered in the 2D PC1;PC2 plane.

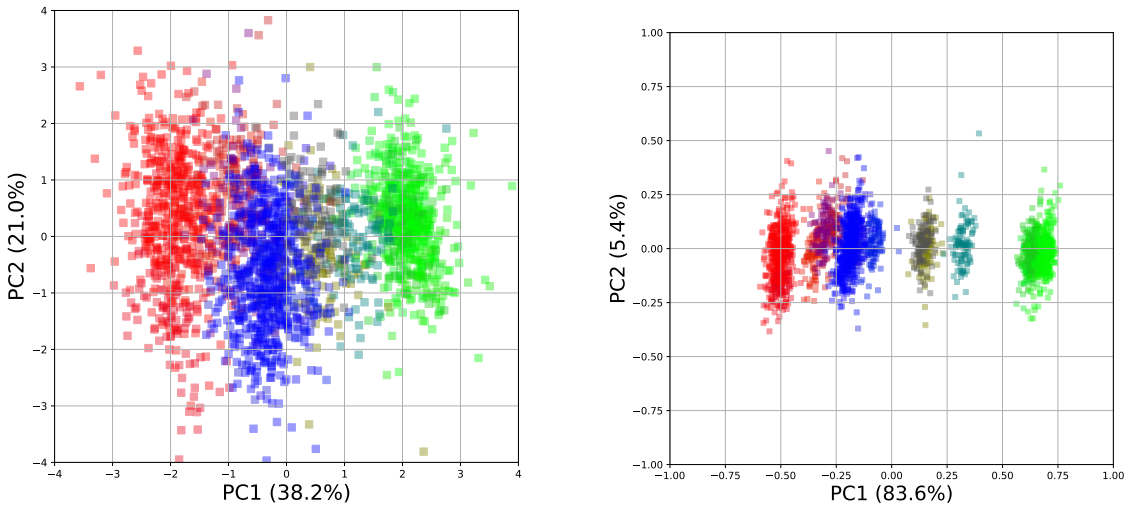


Figure 3.7: PCA on reduced data scaled by StandardScaler on the left, and MinMaxScaler on the right.

Alternatively, we would also want to preserve the shape of the original distribution and for that, min-max scaling comes in handy. This method rescales the original dataset using this formula:

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.15}$$

Plot of resulting transformed data after applying MinMax scaler and performing PCA seems more separable on the PC1 axis, 83.6% of data variance can be attributed to the first principle component. Transformed matrix $\hat{\boldsymbol{Y}}$ is now also rescaled to (-1,1) range.

On both plots of PC1;PC2 plane in figure 3.7, there are multiple distinguishable clusters that can be separated into three main parts - calcium, potassium and sodium rich domains, with calcium being more separated.

Other subdomains are corresponding to $Na^{+}_{(aq)}/Ca^{2+}_{(aq)}$ (1:1) (brown color) and the $K^{+}_{(aq)}/Ca^{2+}_{(aq)}$ (1:1) (cyan), and even a NaCl - KCl region (violet) on the MinMax plot.

Analysis by principal components concludes that information provided by six descriptors distinguishes the 25 mixed electrolytes and is viable for use in further machine learning algorithms.

4 Artificial Neural Network

It seems that many of the discoveries related to science, are in one way or another, inspired by nature, and that is the case for the artificial neural network.
Frank Rosenblatt took the basis in Hebbian theory [33] and McCulloch-Pitts artificial neuron [34] to develop the perceptron, a predecessor to modern deep learning models [35] [9] [10]. Rosenblatt bypassed the complexities of modelling the biological neural system by instead creating an analog model that contained the core principles.

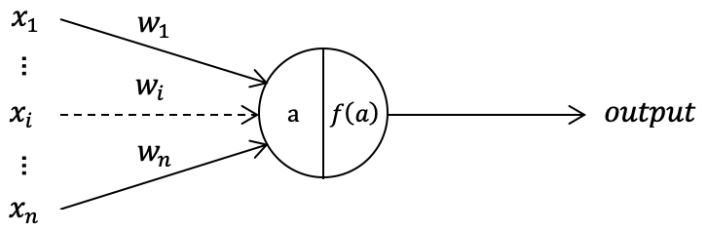


Figure 4.1: A simplified version of Rosenblatt's perceptron.

As it can be seen in figure 4.1, perceptron is somewhat reminiscent of a biological neuron. Weighted sum of inputs represents the activation of the "neuron" $a = \sum_i^n w_i x_i$ and determines the output through the activation function $f(\cdot)$ restricting it to the set $\{0, 1\}$ or $\{-1, 1\}$.
The perceptron can be thought of as a mathematical model for making decision based on weighing the input information. These "weights" express the importance of individual inputs related to the output, and their values are adjusted by comparing the output of the perceptron to the target output. Nielsen [9] excellently explains the decision making that can be made by using a perceptron.
Formally, input vector $\boldsymbol{x}$ is transformed into a feature vector $\phi(\boldsymbol{x})$ and a nonlinear activation function is applied to produce the output [14].

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T \phi(\boldsymbol{x})) \tag{4.1}$$

In the case of perceptron, step function is used as the activation function.

$$f(a) = \begin{cases} +1, a \geq 0 \\ -1, a < 0 \end{cases} \tag{4.2}$$

Bias component can be added to feature vector as $\phi(\boldsymbol{x}) = 1$ and can be viewed as

a measure of how easy it is for the perceptron to activate [9]. Perceptron was the first model able to learn the weights given examples of inputs as a binary linear classifier [10].

4.1 Architecture of a Neural Network

Having briefly mentioned the perceptron, we come to the introduction of a neural network, also known as the multilayer perceptron. Despite the name, the model of the neural network comprises multiple layers of logistic regression models rather than that of multiple perceptrons [14].
The network consist of multiple layers of "neuron" units which can be divided into three parts as shown in figure 4.2. Note that all of the layers are *fully connected*.

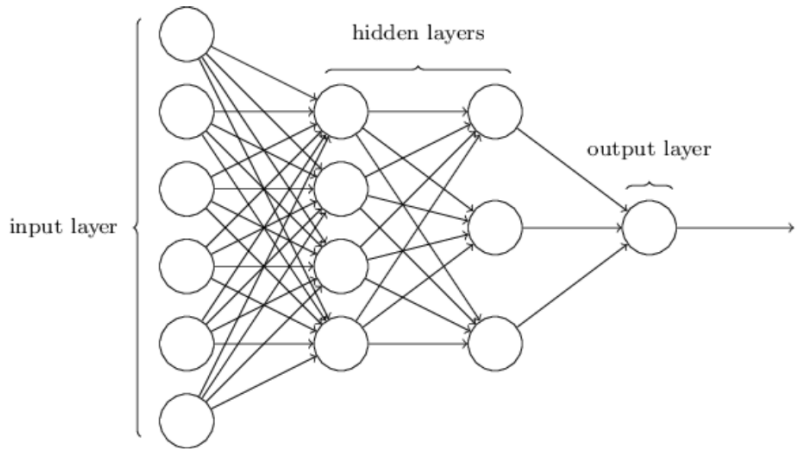


Figure 4.2: Example of a neural network. Figure taken from [9].

The network architecture is defined through the number of layers, neurons and their connections, while the weights (and biases) are defined through parametrization. There are multiple ways to label the neural network by number of layers. One can choose all of the layers, including input and output layers, or omit them and focus on the number of hidden layers or even just use the number of connection layers. Because this work uses shallow networks with only one hidden layer that will be the convention here, for example, the network in 4.2 would be called a 2-layer network. Neurons of the network are also called nodes, and in that context, a neural network is a directed, acyclic graph of layers.

4.2 Learning

How does a neural network work? Well, first we need to specify the problem we are trying to solve, whether it be some kind of regression of classification, and there are numerous problems that neural network are able to solve. Because this work deals with classification of different electrolyte compositions, focus and the examples provided will be on the use of neural network as a classifier.

The most famous classification problem concerns the classification of handwritten digits from the MNIST database. We are provided with a large set of data consisting of images of handwritten digits and their corresponding value (0 through 9). Our goal is to use these images as inputs[4] to the neural network for which it should output the correct values. We initialize the weights to some arbitrary numbers and we try to "feed" the network an image from the dataset - resulting in a completely wrong prediction. But don't give in to despair! We can compare the predicted value with the target one to measure our error and use the backpropagation algorithm to tweak the values of weights and improve our networks performance through training. This is called supervised learning. Previously used PCA is an example of unsupervised learning.

For a classifier, we assume there exists some function $f^* : x \to y$ that maps the input x to a category y. The goal of the feedforward network is to adjust the parameters θ to approximate this function f^* [10]:

$$y = f(x; \theta) \tag{4.3}$$

The process by which the neural network filters the input data can be thought of as finding a transformed representation of the data. The number of hidden units can be interpreted as the dimensionality of the representation space. Having a larger number of hidden units will enable the network to learn more complex representations, but that can sometimes be unwarranted and also expensive. In case of a small number of hidden units and a fully connected network, some information can be lost [1].

[4]Of course, inputs can be transform and standardized. Resolution of grayscale digit images is 28 x 28 pixels and input to the neural network can be formed as a 784 long array of corresponding pixel values.

A single input is defined as $\{x_1, ..., x_d\} = \boldsymbol{x}$. When we provide the neural network with an input $\boldsymbol{x}$, the information flows through the hidden units of each layer and produces $\hat{\boldsymbol{y}}$ as the output in the process of forward propagation [10]. To activate a single unit in the subsequent fully connected layer, we apply the activation function to the weighted sum from all the units in the previous layer with an added bias term. Activations in the first layer will use input values:

$$a_j^{l=1} = h\left(\sum_k w_{jk}^1 x_k + b_j^1\right) \tag{4.4}$$

The connection between j-th neuron in l-th layer to k-th neuron in the $(l-1)$ layer will be labelled with w_{jk}^l [9]. Bias term for each layer is defined as b_j^l. Activation function $h(\cdot)$ is differentiable and nonlinear.

The activation of a j-th neuron in the l-th layer will then be:

$$a_j^l = h\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = h(\sum_j z_j^l) \tag{4.5}$$

We can define z_j^l as the weighted input for j-th neuron in l-th layer. In the simplified vectorized form:

$$\boldsymbol{z}^l \equiv \boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \tag{4.6}$$

$$\boldsymbol{a}^l = h(\boldsymbol{z}^l) \tag{4.7}$$

There are multiple choices for the activation function $h(z)$, as shown in 4.8.

$$h(z): \qquad \sigma(z) \equiv \frac{1}{1+e^{-z}}, \qquad \tanh(z) \equiv \frac{e^z - e^{-z}}{e^z + e^{-z}}, \qquad \text{ReLU}(z) = \max(0, z) \tag{4.8}$$

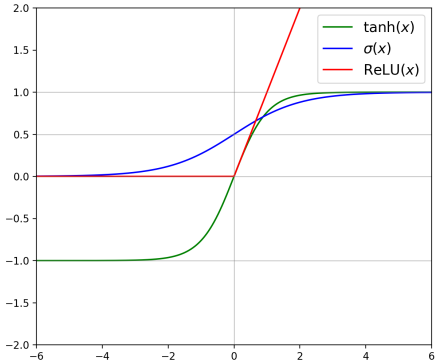


Figure 4.3: Sigmoid, ReLU and tanh activation functions.

Although sigmoid units are useful for Bernoulli output distributions, their use as hidden units in feedforward networks is discouraged because of their saturation across most of their domain which makes gradient-based learning difficult. Good alternative option for hidden units are rectified linear units, which have demonstrated universal approximation properties in a shallow networks. [10]

A linear activation function would result in a linear transformation but many complex problems cannot be solved only through use of linear transformations.

In supervised learning type of machine learning the model learns when it compares obtained values which we feed-forward through the network with the target values. We use the a cost (error, loss) function to determine the "deviation" of obtained values from target ones. Cost function comprises of errors for each data point in the training set [14]:

$$C(\boldsymbol{\theta}) = \frac{1}{N} \sum_{x}^{N} C_x(\boldsymbol{\theta}) \tag{4.9}$$

Each cost function can be specified for the task at hand, and in our case we will use categorical cross-entropy for a many-class classification problem [1]. Our problem is an instance of *single-lable, multiclass classification*.

General term for cross-entropy function in a multilayer neural network is:

$$C = \sum_{j} [y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L)] \tag{4.10}$$

The y_j are target values, and a_j^L are activations in the final output layer L given through feed-forward.

The choice of output units is coupled with the cost function [10]. Softmax output units are usually used in classification problems, where softmax functions represent the probability distribution over k different classes.

$$S(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}} \tag{4.11}$$

Because all the output activations are positive and sum up to 1, we can interpert softmax layer output as a probability distribution.

The learning part of neural network training happens during the backpropagation. For neural network to classify examples correctly, the minimization of the cost function is required. Rumelhart et al. [36] have originally considered a technique involving gradient descent.

To determine how to change weights and biases to minimize the cost function, gradient with respect to those weights and biases can be calculated. It would be useful to picture a function of two variables $F(x, y)$ that is differentiable and has a minima. Gradient of the function provides a direction with which we can search for the minimum of the function. We define a small, positive parameter η called learning rate which determines the rate of the descent and comprises a small step in the direction of the negative gradient [9] [14].

$$\nabla F(x,y)|_p = \begin{bmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \end{bmatrix} \tag{4.12}$$

$$x^{'} = x - \eta \nabla_x \tag{4.13}$$

$$y^{'} = y - \eta \nabla_y \tag{4.14}$$

Given a 2D function $F(x, y)$ gradient descent method finds the point $p = (x_0, y_0)$ such that $F(x_0, y_0) \rightarrow$ min.

In the case of a neural network, the cost function is a complicated function of all the weights and biases, let us say of m parameters $\theta_1, ...\theta_m$.

$$\nabla C \equiv \left(\frac{\partial C}{\partial \theta_1}, ..., \frac{\partial C}{\partial \theta_m} \right)^T \qquad \theta_i^{'} = \theta_i - \eta \nabla C \tag{4.15}$$

Take a look at equation (4.9) - to calculate the complete cost function we need to sum up the cost functions for all of the examples from the entire dataset. To speed up the process we can use a technique called *stohastic gradient descent*. ∇C is estimated by taking a random sample of m training inputs $x_1, ..., x_m$ which are referred to as a mini-batch [9].

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{x_j} \tag{4.16}$$

The epoch of training is completed when all the training inputs are exhausted. Variants of SGD methods are known as *optimizers*.

Backpropagation relates how the cost function changes as we tune the parameters of the neural network. In order to calculate the gradient, we define an error of a neuron j in the layer l as:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{4.17}$$

All other required information is contained in the equations of backpropagation.

$$\delta^L = \nabla_a C \odot h^{'}(z^L) \qquad \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot h^{'}(z^l) \tag{4.18}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{4.19}$$

$\odot$ denotes the elementwise product of two vectors, sometimes called the Hadamard product [9]. First two equation stand for the calculation of error of individual neurons of the network, and the last two equations give us the partial derivatives with respect to weights and biases. We can update the parameters accordingly:

$$w_k^{'} = w_k - \eta \frac{\partial C}{\partial w_k} \qquad b_l^{'} = b_l - \eta \frac{\partial C}{\partial b_l} \tag{4.20}$$

Algorithm which our neural network uses is stated below and is taken from [9]:
The outer loop generates the mini-batches of training examples and progresses through the epochs of training.

1. Input a set of training examples

2. For each training example x: Set the corresponding input activation $a^{x,1}$ and perform the following steps:

   - Feedforward: For each l=2,3,...,L compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$

   - Output error $\delta^{x,L}$: Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma^{'}(z^{x,L})$

   Rate of change of the cost function with respect to any bias in network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{4.21}$$

   Rate of change of the cost function with respect to any weight in network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{4.22}$$

   - Backpropagate: For each $l = L-1, ..., 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma^{'}(z^{x,l})$

3. Gradient descent: For each $l = L, L-1, ...2$ update the weights accordint to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$ and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

Weights and biases need to be initialize to provide the network with a starting point. First simple approach that comes to mind is to initialize weights and biases using independent Gaussian random variables, normalized with mean 0 and standard deviation 1. The problem with this approach is that in the case of large number of neurons, the activation of a particular neuron will be saturated. The saturation of neurons will cause slow learning in the network because very small activations in neurons will barely affect the cost function. If we initialize weights with normalized Gaussians, learning with be slow because activations would often have values near 0 or 1 [9]. The derivative of cost function with respect to weights is:

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_{k}^{l-1}\delta_{j}^{l} \tag{4.23}$$

From this equation we can see that for small activation $a \approx 0$ the gradient term will also be small. The weights output from low activation neurons won't change much during gradient descent which makes it "learn slowly". A weight in final layer will also "learn slowly" if the output neuron has either low (≈ 0) or high (≈ 1) activation. Common term that is used to describe that kind of behavior in a neuron is called saturation. Choice of cost function influences the saturation of output neurons, while choice of weight initialization can help with saturation of hidden neurons.

Better approach is to initialize weights with Gaussian random variables that have the mean of 0, and the standard deviation $\frac{1}{\sqrt{n_{in}}}$, where n_{in} signifies the number of input weights to that neuron. This way, the Gaussian distribution of weighted sums z is more sharply peaked, and that solves the problem of saturation.

For biases, it doesn't matter if we initialize them with standard deviation of 1 or if we let them be 0, and make the gradient descent change them to appropriate values [9]. Xavier Glorot and Bengio have proposed the weight initialization technique with the goal to equalizes the variance of the outputs of a layer to the variance of its inputs [10] known as the *Xavier initialization*. The initalization depends on the choice of the activation function, and they have used a logistic sigmoid function. The *He initialization* [37] applies the same idea in conjunction with the rectified linear function. A variant of that initializer is used in this work: samples for weights initial values are drawn from a uniform distribution within [-limit, limit], where limit $= \sqrt{3 \cdot 1/n}$ where n is the average of the numbers of input and output units.

4.3 Modifications

The code for the neural network in this work was constructed mainly on basis provided by Micheal Nielsen "Neural Networks and Deep Learning" resource [9]. While the code retains many similarities it was modified and adapted for solving the problem of classifying different electrolyte compositions. A comparable neural network was also built in the Keras iteration [45].

In order to be certain of our models predictive ability, we need to test the network on examples for which it hasn't been exposed to before. If the model performs well on previously unobserved inputs, we say that it can *generalize* well.

To deal with this concern available standardized data was split into disjoint subsets in a form of hold-out validation. For each of the 25 classes, 99 measurements were made. A random selection was made from these 99 measurements, and 70 examples were added to the training dataset, 15 were put into a test dataset, and 14 to the validation dataset, for a distribution of (71%, 15%, 14%).
Total number of measurements that were given as data is 2475, and subsets were sampled in such a way that the training set consists of 1750, test set has 375, and validation set contains 350 measurements. The network is trained using the training set and evaluated with the validation set, and test set is used to get the results of a fully trained network.

When it comes to *generalization*, two issues can occur. If the network performs well on the training set, but worse on the test set we say that the network is *overfitting*. On the other hand, if it doesn't achieve sufficiently low training error it is *underfitting* [10].
Input consists of 6 units that are fed to the network and output is "one-hot" encoded into 25 units, one for each class, meaning that the true class corresponding to the example input is represented as a "one" in an array of zeros.

Compare the amount of given data to the MNIST dataset, which consists of 60,000 training images plus 10,000 test images, with a 784 long input vector and 10 classes of digits. A simple sigmoid neural network with one hidden layer with 100 hidden

units trained for 30 epochs can achieve 95% accuracy[5] [9]. Therefore, we should expect for our network to perform with less precision based solely on the quantity and shape of our data.

For the neural network used for testing recognizability of electrolyte composition, 1 hidden layer with 100 ReLU units was chosen after experimentation with various combinations and for the output layer, softmax activation function was used. The process of choosing the optimal parameters of the model is called *hyperparameter optimization*. For our network, this includes the number of epochs for training, size of the mini batch, learning rate η.
To combat the problem of overfitting a technique called L2 regularization, also known as weight decay, can be imployed [10].

$$C_R = C_0 + \frac{\lambda}{n}\sum_w w^2 \tag{4.1}$$

To create the regularized cost function C_R, a term is added to the original cost function C_0 with a regularization parameter $\lambda > 0$ which penalizes the choice of large weights [9]. The update for weights for the standard SGD is changed to:

$$w \leftarrow (1 - \frac{\eta\lambda}{n})w - \frac{\eta}{m}\nabla_w \tag{4.2}$$

Improvements can also be made by using more efficient variations of optimizers instead of the standard Stochastic Gradient Descent algorithm. There is no consensus on which optimizer algorithm is the overall best. Popular contenders are SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam.
RMSprop algorithm (without momentum) was implemented as follows:
Accumulation variables which maintain a moving (discounted) average of the square of gradients are initialized to zero $r = 0$ and learning rate η and decay rate ρ are provided. A small stabilization constant ϵ is used.
Gradient g of the minibatch of examples $\left\{x^{(1)}, ..., x^{(m)}\right\}$ with corresponding targets y^i is calculated.

$$r \leftarrow \rho r + (1 - \rho)g^2 \tag{4.3}$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{r + \epsilon}}g \tag{4.4}$$

[5] A well designed convolutional neural network achieves 99% accuracy.

Class	KCl	NaCl	$CaCl_2$
1	0	0	1
2	0	0.1	0.9
3	0	0.01	0.99
4	0	0.5	0.5
5	0	0.9	0.1
6	0	0.99	0.01
7	0	1	0
8	0.1	0	0.9
9	0.01	0	0.99
10	0.1	0.1	0.8
11	0.01	0.01	0.98
12	0.1	0.8	0.1
13	0.1	0.9	0
14	0.01	0.98	0.01
15	0.01	0.99	0
16	0.5	0	0.5
17	0.5	0.5	0
18	0.8	0.1	0.1
19	0.9	0	0.1
20	0.9	0.1	0
21	0.33	0.33	0.33
22	0.98	0.01	0.01
23	0.99	0	0.01
24	0.99	0.01	0
25	1	0	0

Table 4.1: Reference table for 25 classes and their concentrations of ions. It is convenient to use the three dimensional footprint of concentration mixtures in the form of RGB color index as on the ternary diagram 3.3.

Hrvatski prijevod: Referentna tablica za 25 klasa i njihov postotak koncentracije iona. Boje klasa odgovaraju RGB indeksu.

5 Results

5.1 *Classification Task*

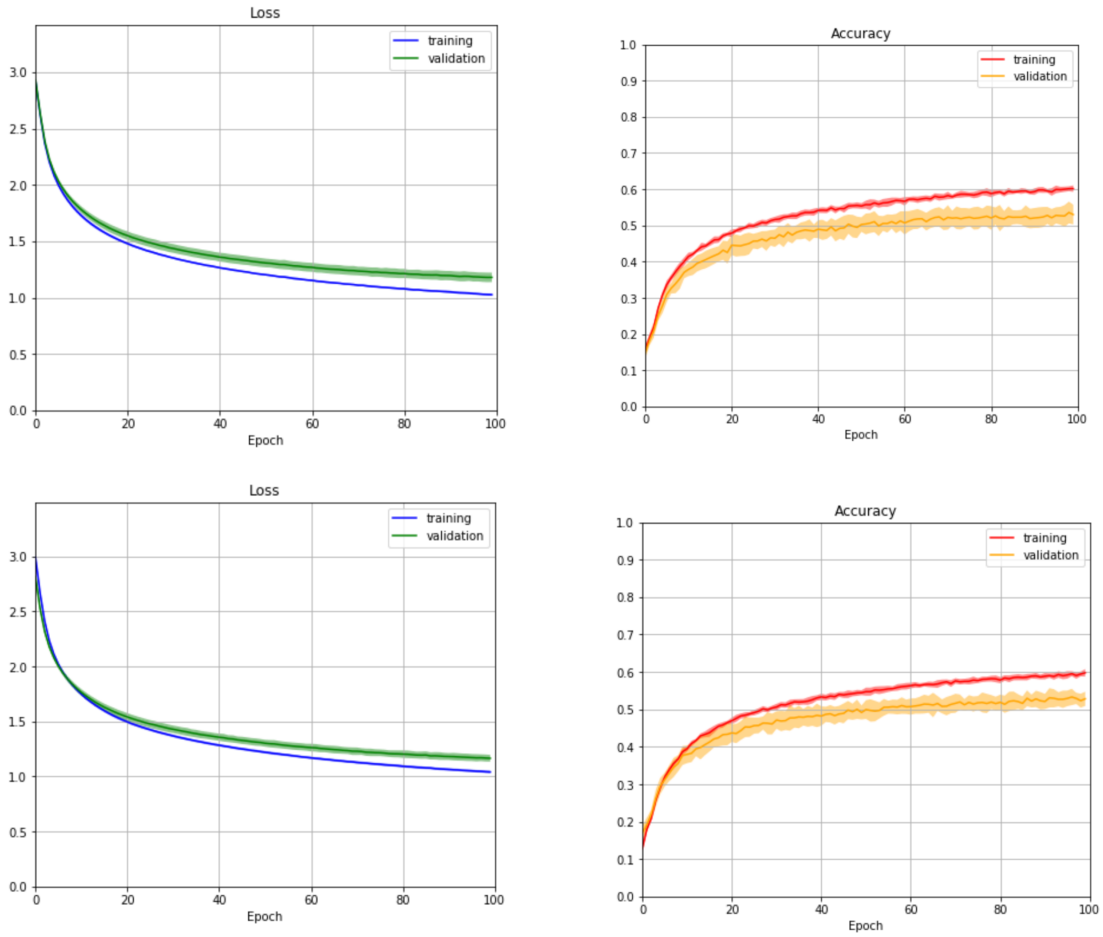


Figure 5.1: a) Neural network with 6 inputs, 100 ReLU hidden units, 25 softmax outputs trained for 100 epochs, RMSprop optimizer, minibatch size 25, $\eta = 0.001$, $\rho = 0.9$, $\lambda = 0.0$, achieving 54 % accuracy on the last epoch. b) Equivalent Keras version of the neural network with same hyperparameters.

The concentrations of 25 classes are noted in reference table 4.1 and each of them is marked with color according to their RGB color index.

Neural network with 6 inputs was trained for 100 epochs 10 times and averaged as shown in figure 5.1 which plots the loss and accuracy values over the time of the training. Each training run produces variations in network performance that is stochastic and that is presented as standard deviation in the form of filled area. The performance on the test set after the network was trained produces 54 % average overall accuracy. Without using any kind of regularization, network shows slight signs of overfitting, but results are nevertheless valid.

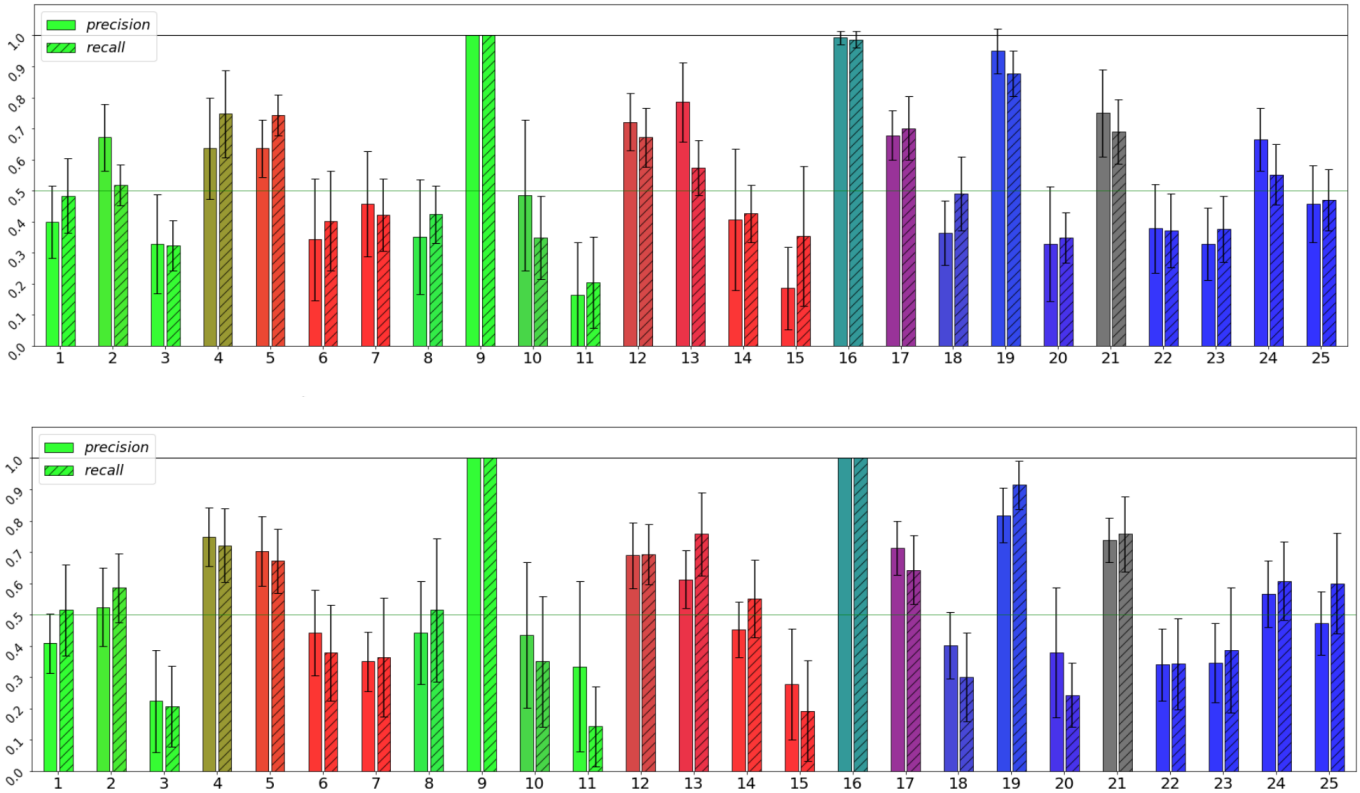


Figure 5.2: Precision and recall scores for a) neural network with hyperparameters stated in figure 5.1 trained for 100 epochs b) equivalent Keras implementation.

After experimenting with different network architectures and comparing results it seems that using the 6 current differences as inputs to the network has an upper bound limit to overall accuracy at around 60 %.

Equivalent neural network was built using Keras package which achieved similar results as shown in b) proving the validity of the "hand-made" neural network. Keras implementation is straight-forward and more efficient, estimated time for running a 100 epoch training is around 10 seconds compared to the 40 seconds of the solely python network.

The nature of classifying 25 classes calls for other measures of networks success which specify the accuracy of predicting individual classes.
The precision is the ratio $t_p/(t_p + f_p)$ where t_p is the number of true positives and f_p the number of false positives. The recall is defined as the ratio $t_p/(t_p + f_n)$ where t_p is the number of true positives and f_n the number of false negatives .

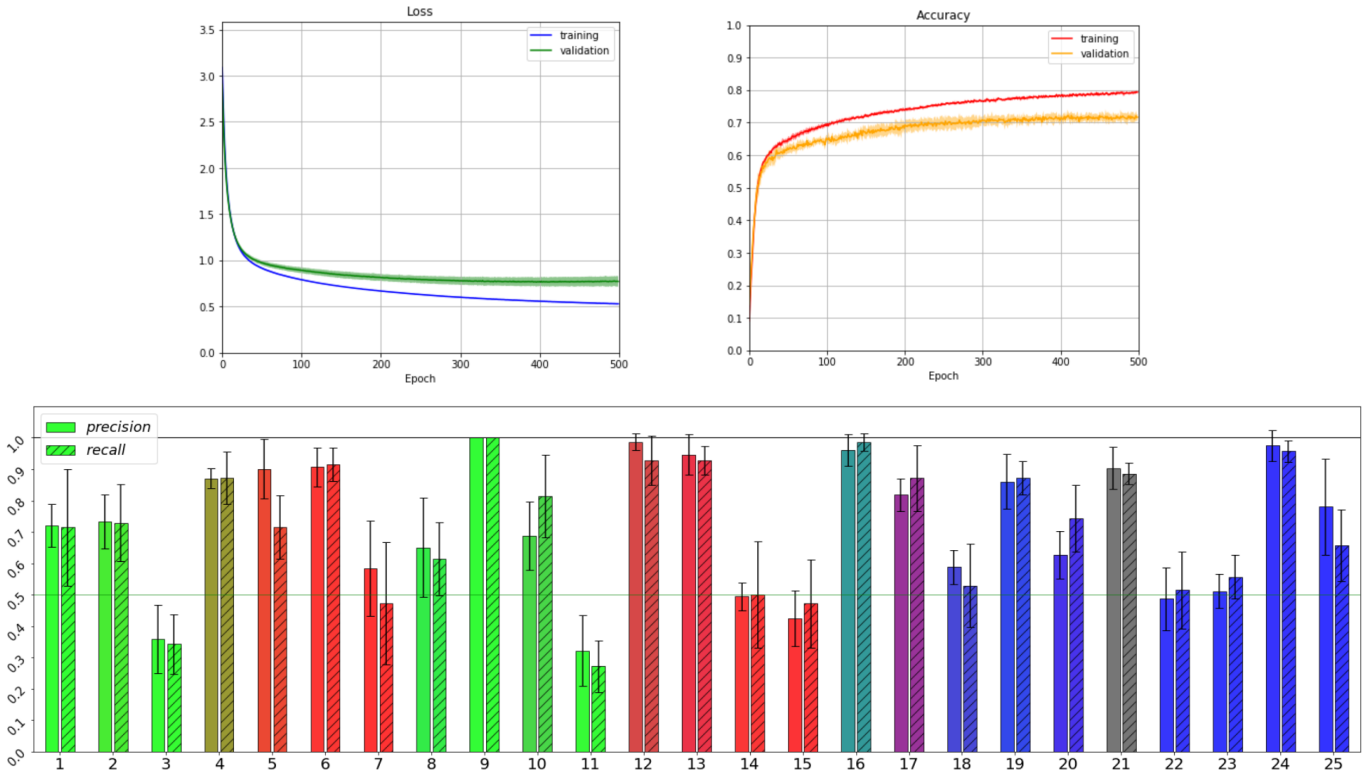


Figure 5.3: Keras neural network with 12 inputs, 25 ReLU hidden units, 25 softmax outputs trained for 500 epochs 5 times, RMSprop optimizer, minibatch size 25, $\eta = 0.001$, $\rho = 0.9$, $\lambda = 0.0$, achieving 71 % average accuracy on the test set.

A random choice would give sensitivity of 4% while a random trial would result in bellow 8% sensitivity in 95% of trials [28]. Compared to that, both python neural network and keras implementation achieve good and similar results of approximately 54% average accuracy. Precision and recall scores in figure 5.2 show very high accuracy in recognizing 9,16 and 19-th class. Repeating the training of the network multiple times shows that 9-th class, which is noted in PCA section in figure 3.6 as being significantly separated from the main data cluster, very quickly and consistently scores 100 % for both precision and recall accuracy. Removing the critical measurements would reduce the number of classes, but it doesn't improve the results of other classes by a significant amount.

Although PCA has informed us that 6 descriptors of current differences contain the information needed for distinction of 25 classes, it seems that it would be valuable to experiment with original 12 current points as inputs and compare the results. Even with using only 25 hidden units, the accuracy is improved by 15% and the upper bound limit is lifted to above 70%. The success of the 12 input network compared to 6 input one shows that the neural network excels with the surplus of information.

5.2 Threshold Task

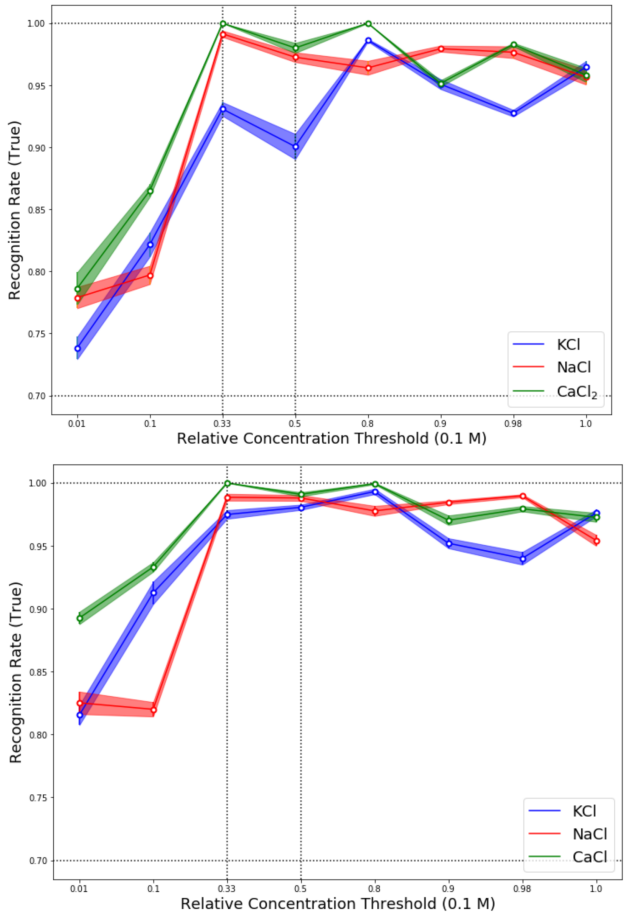


Figure 5.4: Training the neural network: a) 100 epochs with 6 inputs b) 500 epochs for 12 inputs

Neural network was used for a second task, that of specifying the sensitivity of the network to a chosen ionic concentration. Neural network was slightly modified, using 50 hidden ReLU units and a single output node with a sigmoid activation function. Targets y were changed depending on whether specified ionic concentration passed a certain threshold (1) or not (0). For each ionic concentration neural network was trained for 100 epochs 5 times and the averaged results are shown in figure 5.4.

This task shows significantly higher accuracy than that of classifying 25 solutions because the amount of data that was distributed among three cation types was more appropriate to the task complexity. Neural network is better at classifying majority ions and equimolar blends than minority ion detection but overall, accuracy ranges from 75 % to 100%. Higher sensitivity of the network in classification of Ca^{2+} can be attributed to a different ion valence.

5.3 Software and Packages

The neural network and Principal component analysis were realised through Python programming language [38] which has recently experienced a surge in popularity which could be attributed to its minimalist iteration and emphasis on readability. Python allows the use of different packages that facilitate user's problem solving such as Numpy [39], a library designed to ease numerical computing and Scipy [40] focused on scientific computing. To create visualizations of data, packages like Matplotlib [41] and Plotly [42] were used. Pandas [44] library was used for data manipulation and analysis.

There is a plentiful support for machine learning and deep learning in python language. For example, Scikit-learn [43] is a useful package for machine learning in Python. A high level framework for building neural networks is Keras [45], deep learning API written in Python. Keras is able to run on top of other open source machine learning libraries, for example, Tensorflow [46] developed by Google Brain team.

The code was written using the Jupyter notebook interface [47], a web-based interactive computational environment which consists of an ordered list of input/output cells that can contain code, text and plots and is convenient for user usage in this particular subject. Jupyter notebook is available bundled in an Anaconda distribution package [48].

6 Conclusion

In this thesis we have successfully shown the applicability of OECT in conjunction with a neural network in the task of recognition of complex solutions, specifically the NaCl-KCl-$CaCl_2$ electrolyte compositions.

First, we have determined the 6 current differences from the measurements of the 99 currents for each electrolyte composition. Performing Principal Component Analysis has shown that these descriptors register enough data separability and that they are suitable to be used as data inputs to a neural network.

Two neural networks with one hidden layer of 100 ReLu units and 25 Softmax outputs were constructed, one coded manually and the other using Keras API, both achieving similar results, with Keras performing significantly faster. Using 6 inputs the network achieves around 55% accuracy in successfully categorizing 25 classes. Testing the 12 inputs, corresponding to current points from which the 6 current differences were obtained, as inputs achieves higher accuracy of around 70%. Increasing the input size helped the network to learn better in the classification task.

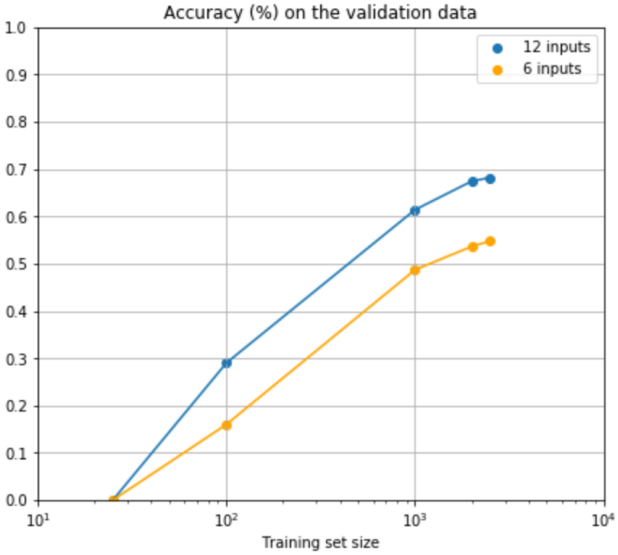


Figure 6.1: Results from using k-fold cross validation (which confirmed previous validation results from 71-15-14 distribution of data) with $k = 4$ for varying training data size and different input size.

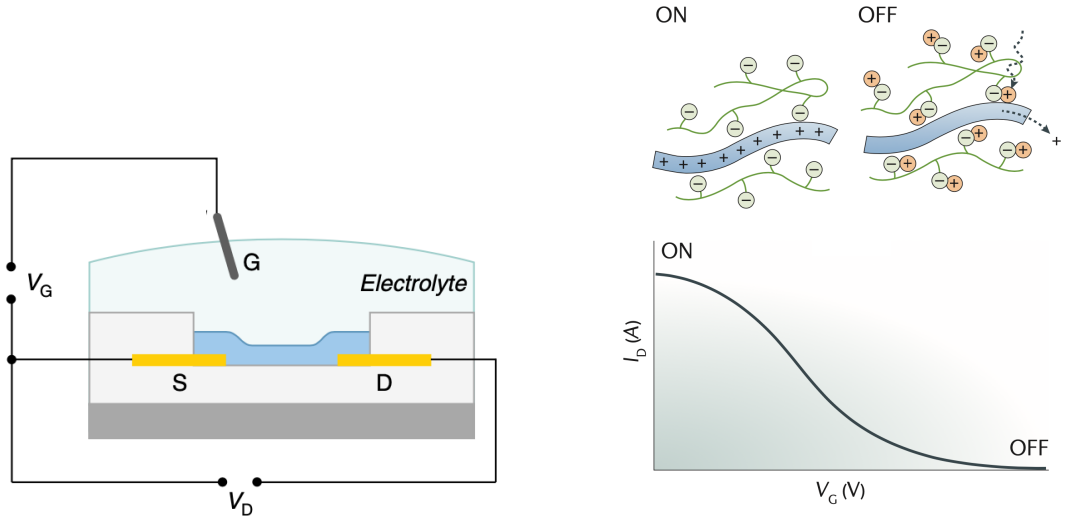
To test that assumption, k-fold cross validation algorithm was used to verify the performance of varying training set size for 6 and 12 inputs shown in figure 6.1. This plot suggests two things, that the neural network would benefit from using a larger input size, maybe a larger distribution of points from measured currents, and that accuracy obviously increases with larger training data size.

Besides the task of classifying 25 classes, neural network was adjusted for a threshold task, from which sensitivity in regards to main three solutions can be tested. The quantity of data was better suited for the threshold task resulting in accuracies higher than 80%, with calcium-rich solution maintaining the highest accuracy trend (5.4).

The success of the first task shows relevance for further application in the statistical identification of unknown cations while the second shows promise the identification of specific ions without labelling [28]. This shows high potential for further research in classifying more complex environments, such as investigation of sensing complex molecules such as tumor markers, and machine learning as valuable and versatile tool in application of neuromorphic engineering.

7 Prošireni sažetak

7.1 Uvod



(a) Tipična struktura OECT-a

(b) Krivulja prijenosa osiromašenog tipa

Slika 7.2: **a** Na shemi poprečnog presjeka OECT-a, označene su upravljačka elektroda (G) i izvor (S) i odvod (D), dok je organski poluvodički kanal označen plavom bojom. Uređaj je uronjen u elektrolitsku otopinu. **b** Krivulja prijenosa $(I_D - V_G)$ za osiromašeni tip operacije OECT je prikazana. Na nultom naponu upravljačke elektrode, polimerski kanal vodi struju šupljina i kako se primljenjeni napon povećava, šupljine su zamijenjene kationima [20].

U ovom radu testirat ćemo primjenjivost strojnog učenja, posebno umjetnih neuronskih mreža, u određivanju relativnih koncentracija različitih sastava elektrolita. Ovaj istraživački napor može se smatrati jednim od brojnih koraka u daljnjem istraživanju inteligentnih senzora koji mogu "osjetiti" različite kemijske ili organske sastave i koji se mogu koristiti u raznim poljima (kao što je medicina) i okolinama.

Specifično, zadatak je implementacija neuronske mreže u problemu klasifikacije relativne koncentracije 25 različitih sastava elektrolita NaCl-KCl-$CaCl_2$, izmjerenih koristeći OECT uređaj.

Organski elektrokemijski tranzistor (OECT) pripada tipu tranzistora koji zamijenjuje uobičajene anorganske vodiče i poluvodiče organskim molekulama ili polimerima. Organski poluvodiči predstavljaju jeftinu, laganu i fleksibilnu alternativu tipičnim poluvodičima [19].

Organskim poluvodičima dobro odgovara struktura tankoslojnog tranzistora (TFT) [16], pa tako OECT-ovi pripadaju podskupu organskih tankoslojnih tranzistora (OTFT) [18] čiji je tanki film izrađen od približno dvodimenzionalnog organskog aktivnog sloja. Organski materijali imaju širok spektar zanimljivih svojstava, poput konfiguriranja molekula kako bi odgovarali određenim zahtjevima kroz sintetsku kemiju i potencijal za jeftinu proizvodnju zahvaljujući ispisu velike površine omogućenom lakoćom nanošenja polimernih prevlaka na razne podloge [17].

Struktura OECT-a sadrži sve glavne karakteristike uobičajenih tranzistora. Upravljačka elektroda G (Gate) uronjena je u elektrolit, a elektrode za izvor S (Source) i odvod D (Drain) pružaju kontakt s kanalom.

Prekidač za uključivanje / isključivanje stvara se kad se ioni ubrizgavaju iz elektrolita u organski film pod nadzorom napona na upravljačkoj elektrodi V_G. Vodljivost kanala se zatim mijenja na temelju doping stanja organskog poluvodiča, a odvodna struja I_D inducirana je odvodnim naponom V_D.

Poluvodički materijal p-tipa koji se uobičajeno koristi u OECT kanalu je poli (3,4-etilendioksitiofen) dopiran poli(stirensulfonatom) skraćeno PEDOT: PSS. PEDOT:PSS održava dobru elektrokemijsku stabilnost u vodenim elektrolitima, pokazuje visoku električnu vodljivost, a može se lako otisnuti na razne podloge i proizvesti tako da ima transkonduktivnost u milisimensima i vrijeme odziva u mikrosekundama [22] [20].

Pecqueur et al. [29] prethodno su potvrdili operativnost OECT-a na širokom rasponu kationa. Istraživanje se temeljilo na prethodnom znanju o frekvencijskom ovisnosti ponašanja sučelja PEDOT:PSS / elektrolit. U režimu stacionarnog stanja ioni specifične konfiguracije akumuliraju se u glavnini PEDOT:PSS i moduliraju vodljivost kanala, dok ionske struje dominiraju odzivom odvodne struje u prijelaznom režimu [28]. Ova dva različita režima ovise o koncentraciji iona i utječu na impedanciju tranzistora na različitim frekvencijama, poput odvođenja primjesa u kanalu na niskim frekvencijama i kapacitivnog vezanja elektrolitskih vrata na visokim frekvencijama.

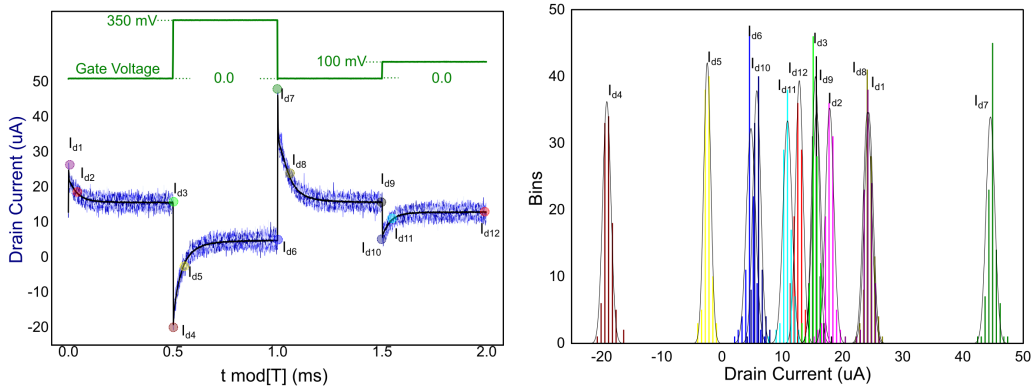
Dobiveni rezultati u tom radu motivirali su praktičniji pristup za dinamičko otkrivanje kationa, naime primjenu digitalnih ulaznih napona za dobivanje analogno strujno moduliranog izlaza. Izlažući soli naponu kvadratnog impulsa V_G na niskoj i visokoj frekvenciji i mjereći I_D odvodne struje, mogu se zabilježiti različite modulacije struje ΔI, koje se dalje mogu koristiti za diskriminaciju kationa.

Olakšavajući postupak i iskorištavajući ionsku dinamiku OECT-a , Pecqeur et al. [28] dizajnirali su protokol pulsa napona kHz pogodan za modulaciju njegovog odziva struje odvoda, u režimu za koji multiparametarska dinamika omogućuje izdvajanje dovoljno visoko-dimenzionalnih podataka koji opisuju ionski sadržaj elektrolita za zadani problem određivanja relativnih koncentracija 25 različitih elektrolitskih sustava.

Jedan OECT uređaj bio je izložen elektrolitima različitih konfiguracija; kalcijevim, natrijevim i kalijevim ionima u smjesama različitih relativnih koncentracija. Zabilježene su periodične prolazne struje, karakteristike iona i primijenjeni naponi.

Kako bi se isključila sustavna odstupanja u podatcima, korišten je jedan uređaj i mjerenja su izvršena u jednom vremenskom razdoblju. Primijenjene su dvije stimulacije napona pri $V_{G_1} = 100$ mV i $V_{G_2} = 350$ mV u trajanju od 0,5 ms, odvojene periodom odmora jednakog trajanja ($V_G = 0$ mV).

Ukupno trajanje od 200 ms osiguralo je mjerenje 99 strujnih sekvenci za jednu koncentraciju određenog tipa kationa.



Slika 7.3: a) Uzorak signala odvodne struje u trajanju od 2-ms za jedno mjerenje 0.1 M KCl_{aq} od 200-ms. b) Statistička raspodjela 12 karakterističnih vrijednosti uzetih iz signala odvodne struje jednog 200-ms mjerenja 0.1 M KCl_{aq}. Slika je uzeta iz [28].

Da bi se iz podataka odredili kationi, trebalo je utvrditi relevantne deskriptore informacija koji će se unijeti u naknadne algoritme strojnog učenja. Članak [28] predlaže uzimanje karakterističnih vrijednosti iz odvodne struje, kao što su 12 specifičnih vrijednosti struje koje označavaju različite točke šiljaka, prijelaznih razdoblja i platoa u stacionarnom stanju. Statistička raspodjela tih vrijednosti prikazana je u 7.3 b) gdje se može primijetiti relativno razdvajanje i simetrija oko srednje vrijednosti.

Kako bi se pojednostavnio oblik ulaznih podataka, broj deskriptora smanjen je na šest modulacija struje.

$\Delta I_{\text{std}}^{100} = I_{d3} - I_{d12}$, modulacija odvodne struje stacionarnog stanja za V_G = 100 mV.

$\Delta I_{\text{trs}}^{100} = I_{d2} - I_{d11}$, kao 50 μs modulacija struje prijelaznog razdoblja za V_G = 100 mV.

$\Delta I_{\text{spk}}^{100} = I_{d1} - I_{d10}$, kao 1 μs šiljak modulacije odvodne struje za V_G = 100 mV.

$\Delta I_{\text{std}}^{350} = I_{d9} - I_{d6}$, modulacija odvodne struje stacionarnog stanja za V_G = 350 mV.

$\Delta I_{\text{trs}}^{350} = I_{d8} - I_{d5}$, kao 50 μs modulacija struje prijelaznog razdoblja za V_G = 350 mV.

$\Delta I_{\text{spk}}^{350} = I_{d7} - I_{d4}$, kao 1 μs šiljak modulacije odvodne struje za V_G = 350 mV.

Za utvrđivanje dostatnosti ovih vrijednosti za prepoznavanje koncentracija kationa, provedena je multivarijatna analiza podataka.

7.2 Analiza glavnih komponenti

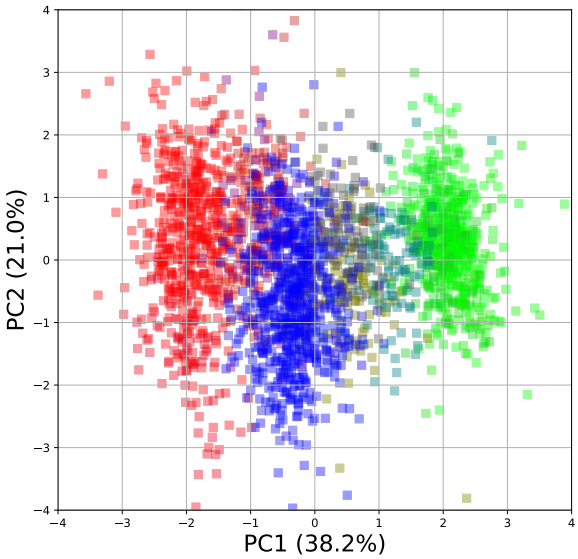
	PC1	PC2	PC3	PC4	PC5	PC6
	38.2%	21.%	16.2%	13.9%	10.6%	0.1%
$\Delta I_{\text{std}}^{100}$	0.09	-0.38	0.9	0.2	0.07	0.
$\Delta I_{\text{std}}^{350}$	-0.39	-0.55	-0.03	-0.66	0.33	0.
$\Delta I_{\text{trs}}^{100}$	0.53	-0.39	-0.33	0.45	0.51	-0.
$\Delta I_{\text{trs}}^{350}$	0.17	-0.81	-0.23	0.08	-0.51	0.
$\Delta I_{\text{spk}}^{100}$	0.95	0.09	0.06	-0.27	-0.03	0.06
$\Delta I_{\text{spk}}^{350}$	0.95	0.08	0.06	-0.28	-0.03	-0.06

Tablica 7.1: Tablica varijance glavnih komponenti i koeficijenata ovisnosti varijabli i glavnih komponenti za šest različitih deskriptora.

Jednostavan i prikladan način ispitivanja razdvojenosti točaka podataka je analiza glavnih komponenti (Principal Component Analysis - PCA). Pripada podskupini algoritama učenja bez nadzora i fokusira se na transformacije skupa podataka kako bi se stvorio novi interpretabilniji prikaz podataka u službi vizualizacije ili kompresije podataka [31].

Podaci se sastoje od 25 elektrolita sa nizom od 99 vektora od šest komponenata ili u kontekstu strojnog učenja, 6 značajki, za ukupno 2475 $\mathbb{R}^6$ vektora. Ti se vektori mogu vizualizirati kao točke u 6-dimenzionalnom prostoru. Analiza glavnih komponenata

može olakšati prikaz visokodimenzionalnih podataka smanjenjem dimenzionalnosti u kojem smanjujemo dimenzije d-dimenzionalnog skupa podataka projicirajući ga na $k < d$ dimenzionalni podprostor zadržavajući većinu informacija. PCA metoda može se opisati kao rotacija skupa podataka kako bi se razdvojile točke podataka po njihovim značajkama i smanjio broj značajki koje ne pružaju dovoljno informacija.
Rezultati primjene analize glavnih komponenti na šest deskriptora prikazani su na tablici 7.1. Sva mjerenja tj. opažanja zapisana su u obliku matrice podataka $\boldsymbol{X}$ na kojoj je primljenjen postupak standardizacije.
Prvi red 7.1 prikazuje udio objašnjene ukupne varijance. Kao što se očekivalo za PCA, prve glavne komponente PC1 odabiru smjer maksimalne varijance, u ovom konkretnom slučaju, od 38,2 %. Značajan dio ukupne varijance ($\sim$ 60 %) može se projicirati na ravninu PC1; PC2. PC6 pokazuje zanemariv omjer varijance od 0,1 % što ukazuje na neki oblik smanjenja dimenzionalnosti, ali pažljivije razmatranje važnosti šest deskriptora pokazuje da su svi oni neophodni za identificiranje kationa [28]. Kasnije korištene neuronske mreže imaju koristi od svih dostupnih informacija.



Slika 7.4: Prikaz prve dvije glavne komponente dobivene primjenom PCA na standardizirane podatke. Iz podataka su izbačena 99 problematičnih mjerenja klase $\boldsymbol{y}_9 = (0.01, 0, 0.99)$

Na obje plohe PC1; PC2 ravnine na slici 3.7, postoji više prepoznatljivih grupa koji se mogu razdvojiti u tri glavna dijela - domene bogate kalcijem, kalijem i natrijem, a kalcij je više odvojen. Ostale poddomene odgovaraju $Na^{+}_{(aq)}/Ca^{2+}_{(aq)}$ (1:1) (smeđa) i

$K^{+}_{(aq)}$/$Ca^{2+}_{(aq)}$ (1:1) (tirkizna).
Analiza glavnih komponenata pokazuje da informacije dane od šest deskriptora razlikuju 25 miješanih elektrolita i valjane su za upotrebu u daljnjim algoritmima strojnog učenja. U kasnijim razmatranjima se pokazalo da su i prvotnih 12 točaka valjan odabir ulaznih podataka za neuronsku mrežu.

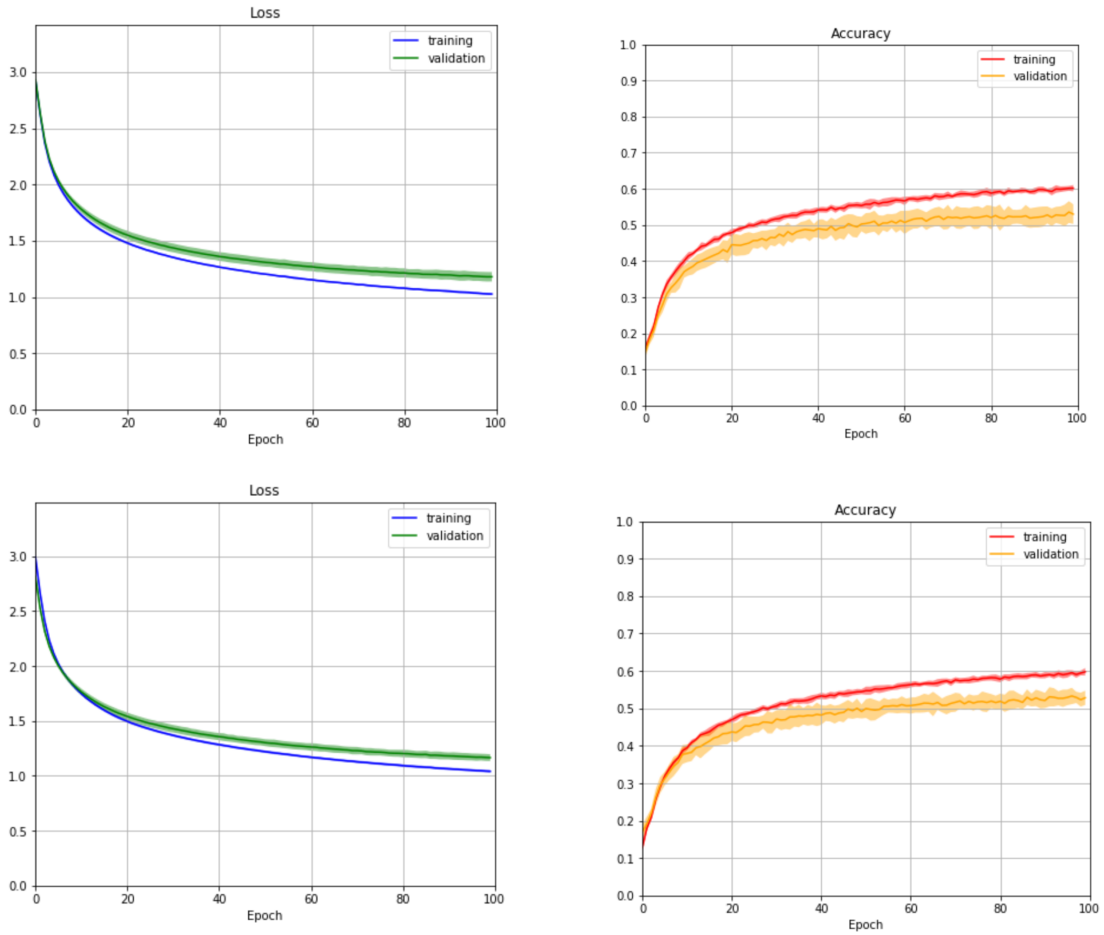
7.3 Neuronska mreža

Kod za neuronsku mrežu u ovom radu konstruiran je uglavnom na temelju knjige Micheala Nielsena "Neural Networks and Deep Learning" [9]. Kod je napisan u Python [38] programskom jeziku. Iako kod sadrži mnogo sličnosti, modificiran je i prilagođen za rješavanje problema klasifikacije različitih sastava elektrolita.
Usporediva neuronska mreža sastavljena je i u Keras [45] iteraciji.

Da bismo bili sigurni u sposobnost predviđanja naših modela, moramo testirati mrežu na primjerima kojima prije nije bila izložena. Ako model dobro predviđa prethodno neviđene observacije, kažemo da dobro *generalizira*.
Da bi riješili ovaj problem, dostupni standardizirani podatci podijeljeni su u razdvojene podskupine. Za svaku od 25 klasa napravljeno je 99 mjerenja. Iz tih 99 mjerenja izvršen je slučajan odabir, tako da je 70 primjera dodano skupu podataka za trening, 15 je stavljeno u testni skup, a 14 u skup podataka za provjeru valjanosti, tvoreći raspodjelu od (71%, 15%, 14%).
Ukupan broj mjerenja je 2475, a podskupovi su odabrani na takav način da se skup za treniranje sastoji od 1750, testni skup ima 375, a skup za validaciju sadrži 350 mjerenja. Mreža se trenira pomoću skupa za treniranje i ocjenjuje se validacijskim skupom, a testni skup koristi se za dobivanje rezultata potpuno istrenirane mreže.

Što se tiče konstrukcije neuronske mreže za raspoznavanje elektrolitskih kompozicija, 1 skriveni sloj od 100 ReLU aktivacijskih jedinica je odabran nakon eksperimentacije sa različitim kombinacijama, dok se softmax aktivacijska funkcija koristi u izlaznom sloju. Kao optimizacijski algoritam odabran je RMSprop bez dodatka momentuma.

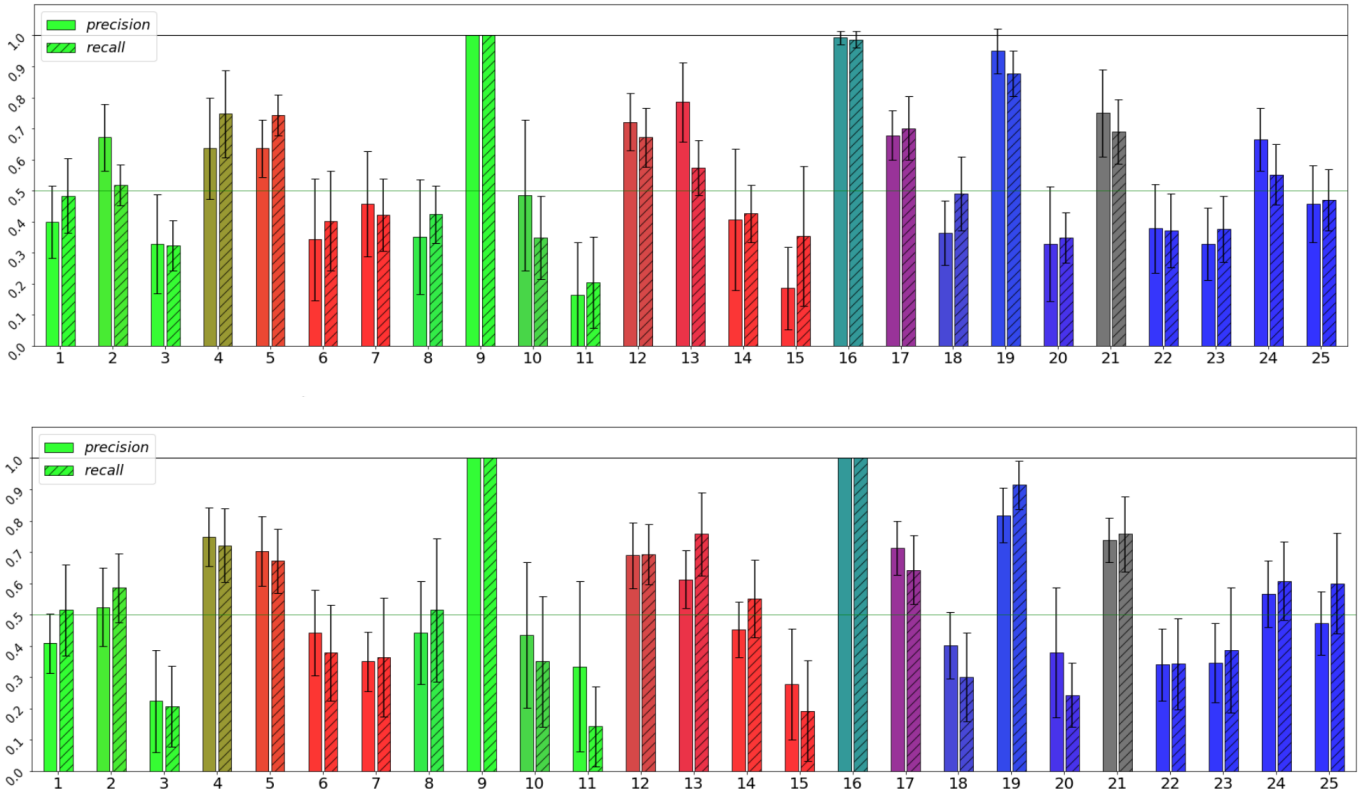
7.4 Rezultati



Slika 7.5: a) Neuronska mreža sa 6 ulaza, 100 ReLU skrivenih jedinica, 25 softmax izlaza trenirana 100 epoha, RMSprop optimizator, veličina uzorka 25, $\eta = 0.001$, $\rho = 0.9$, $\lambda = 0.0$, ostvaruje 54 % točnost u zadnjoj epohi. b) Ekvivalentna Keras verzija neuronske mreže sa jednakim hiperparametrima.

Koncentracije 25 klasa zabilježene su u referentnoj tablici 4.1 na stranici 31. i svaka od njih označena je bojom pripadnog RGB indeksa.

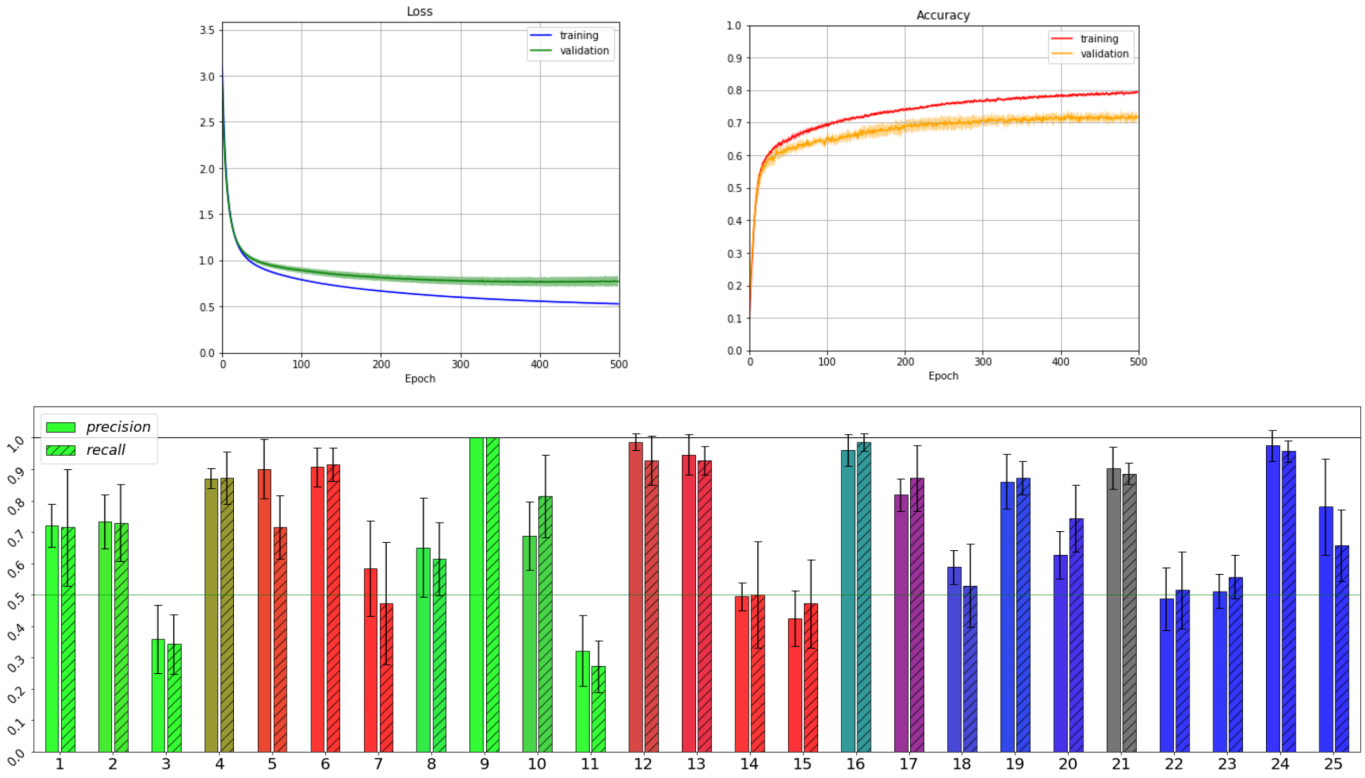
Neuronska mreža sa 6 ulaza trenirana je 100 epoha po 10 puta, a uprosječeni rezultat točnosti i funkcije gubitka u ovisnosti o epohi prikazan je na slici 7.5. Svako treniranje stvara varijacije u izvedbi mreže koja je stohastička i koje se prikazuju kao standardno odstupanje u obliku ispunjenog područja. Izvedba na testnom skupu podataka nakon treniranja mreže daje ukupnu prosječnu točnost od 54%. Bez upotrebe bilo kakve regulacije, mreža pokazuje blage znakove prekomjernog ugađanja, ali rezultati su i dalje prihvatljivi.



Slika 7.6: Ocjene preciznosti i osjetljivosti za a) neuronsku mrežu hiperparametara zadanih na slici 7.5 trenirana 100 epoha b) ekvivalentnu Keras implementaciju.

Nakon eksperimentiranja s različitim mrežnim arhitekturama i usporedbe rezultata, čini se da korištenje 6 vrijednosti razlike struja kao ulaza u mrežu ima gornju granicu ukupne točnosti oko 60%. Ekvivalentna neuronska mreža implementirana pomoću Keras paketa postiže slične rezultate kao što je prikazano u b) dokazujući valjanost "ručno izrađene" neuronske mreže. Implementacija Kerasa je jednostavnija i učinkovitija, procijenjeno vrijeme izvođenja treninga od 100 epoha je oko 10 sekundi u usporedbi s 40 sekundi isključivo Python mreže.

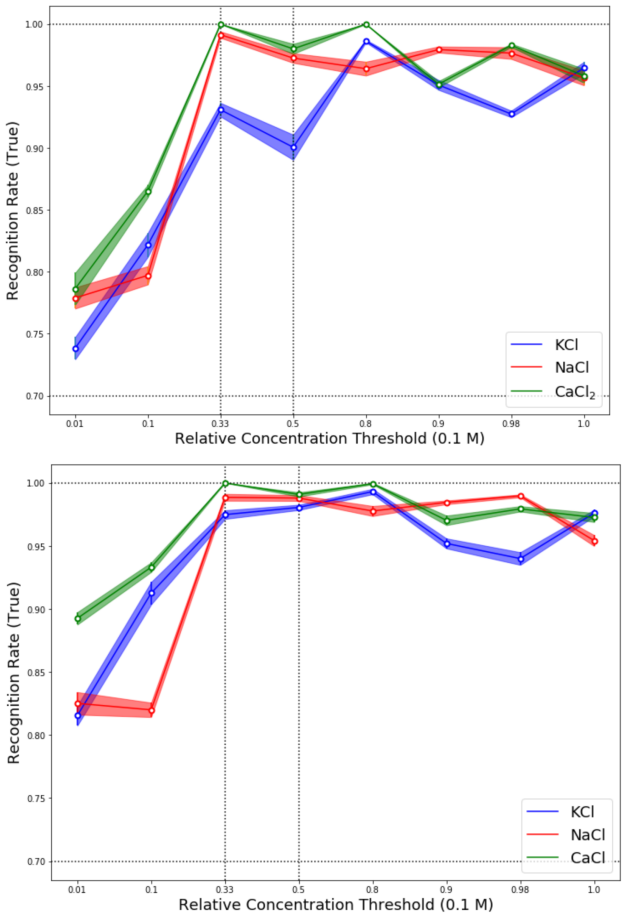
Priroda klasificiranja 25 klasa zahtijeva i druge mjere procjene uspješnosti neuronske mreže koje određuju točnost predviđanja pojedinih klasa.
Preciznost je omjer $t_p/(t_p + f_p)$ gdje je t_p broj stvarno pozitivnih rezultata, a f_p broj lažno pozitivnih rezultata.
Osjetljivost je definiran kao omjer $t_p/(t_p + f_n)$ gdje je t_p broj stvarno pozitivnih rezultata, a f_n broj lažno negativnih.



Slika 7.7: Keras neuronska mreža sa 12 ulaza, 25 ReLU skrivenih jedinica, 25 softmax izlaza trenirana 500 epoha 5 puta, RMSprop optimizator, veličina uzorka 25, $\eta = 0.001$, $\rho = 0.9$, $\lambda = 0.0$, ostvaruje 71% prosječne točnosti na testnom skupu.

Slučajni izbor dao bi osjetljivost od 4%, dok bi slučajni pokus rezultirao osjetljivošću ispod 8% u 95% ispitivanja [28]. U usporedbi s tim, i Python neuronska mreža i Keras implementacija postižu dobre i slične rezultate s približno 54% prosječne točnosti. Rezultati preciznosti i osjetljivosti na slici 7.6 pokazuju vrlo visoku preciznost u prepoznavanju 9., 16. i 19. klase.

Iako je analiza glavnih komponenti potvrdila da 6 deskriptora razlike struja sadrži informacije potrebne za razlikovanje 25 klasa, čini se da bi bilo vrijedno eksperimentirati s izvornih 12 točaka vrijednosti struje kao ulaznih podataka i usporediti rezultate. Čak i ako se koristi samo 25 skrivenih jedinica, točnost se poboljšala za 15%, a gornja granica podigla na iznad 70 %.
Uspjeh mreže sa 12 ulaza u usporedbi s 6 ulaznih vrijednosti još jednom pokazuje da neuronske mreže značajno bolje funkcioniraju ako su opskrbljene viškom korisnih informacija.

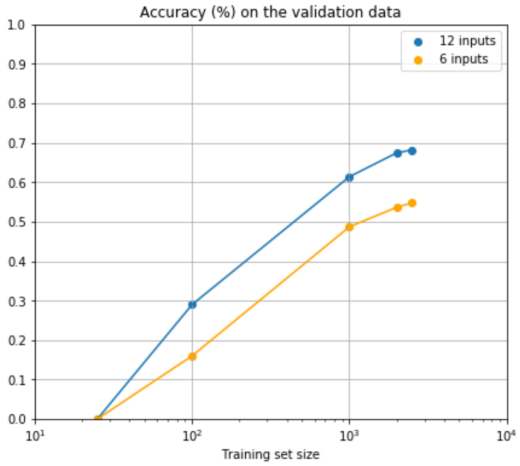


Slika 7.8: Treniranje neuronske mreže ostvarene za zadatak prelaska praga: a) 100 epoha za 6 ulaza, b) 500 epoha za 12 ulaza.

Neuronska mreža korištena je za drugi zadatak, za određivanje osjetljivosti mreže na odabranu ionsku koncentraciju. Prvotna neuronska mreža je prilagođena tom zadaku te koristi 50 skrivenih ReLU jedinica i jedan izlazni čvor s sigmoidnom funkcijom aktivacije. Izlazne vrijednosti (mete) y promijenjene su ovisno o tome je li navedena ionska koncentracija prešla određeni prag (1) ili nije (0). Za svaku ionsku koncentraciju neuronska mreža trenirana je 100 epoha 5 puta, a prosječni rezultati prikazani su na slici 7.8.

Ovaj zadatak pokazuje značajno veću točnost od klasificiranja 25 klasa, jer je količina podataka koja je raspodijeljena između tri tipa kationa bila primjerenija složenosti zadatka. Neuronska mreža je bolja u klasificiranju većinskih iona i ekvimolarnih mješavina od otkrivanja manjinskih iona, ali sveukupno, točnost se kreće od 75% do 100%. Veća osjetljivost mreže u klasifikaciji Ca^{2+} može se pripisati različitoj valenciji iona.

7.5 Zaključak



Slika 7.9: Rezultati korištenja k-struke unakrsne provjere (koja potvrđuje validacijske rezultate 71-15-14 raspodjele podataka) za $k = 4$ za različite veličine skupa podataka za treniranje i različitu veličinu ulaznih podataka.

U ovom radu uspješno je pokazana primjenjivost kombinacije OECT-a i neuronske mreže u zadatku prepoznavanja kompleksnih otopina, specifično sustava elektrolita NaCl-KCl-$CaCl_2$.

Najprije je utvrđeno 6 vrijednosti razlika struje iz 99 mjerenja za svaki sastav elektrolita. Izvođenje analize glavnih komponenti pokazalo je da ovi deskriptori bilježe dovoljno razdvojenosti podataka i da su prikladni za upotrebu kao ulazni podatci u neuronskoj mreži.

Konstruirane su dvije neuronske mreže s jednim skrivenim slojem od 100 ReLu jedinica i 25 Softmax izlaza, jedna kodirana ručno, a druga pomoću Keras API-ja, oboje postižući slične rezultate, s tim da Keras verzija znatno brža.

Korištenjem 6 ulaza mreža postiže oko 55 % točnosti u uspješnoj kategorizaciji 25 klasa. Ispitivanje 12 ulaza, koji odgovaraju vrijednostima struja od kojih su dobiveni 6 deskriptora, je pokazalo veću točnost od oko 70 %. Povećavanje veličine unosa pomoglo je mreži da bolje uči u zadatku klasifikacije.

Za testiranje te pretpostavke korišten je k-struki algoritam unakrsne provjere valjanosti kako bi se usporedila izvedba različitih veličina skupa treninga za 6 i 12 ulaza prikazanih na slici 7.9. Ovaj graf pokazuje dvije stvari, da bi neuronskoj mreži koristilo korištenje većeg broja ulaznih podataka, možda veće raspodjele točaka izmjerenih struja, a da se točnost povećava s količinom podataka za treniranje, što je očekivano.

Uz zadatak klasifikacije 25 klasa, neuronska mreža prilagođena je i zadatku određivanja osjetljivosti mreže na pojedinu ionsku koncentraciju. Količina podataka bila je prikladnija za taj zadatak, što je rezultiralo preciznošću većom od 80%, s otopinom bogatom kalcijem koja je zadržala najviši trend točnosti (7.8).

Uspjeh prvog zadatka pokazuje relevantnost za daljnju primjenu u statističkoj identifikaciji nepoznatih kationa, dok drugi sugerira identifikaciju specifičnih iona bez obilježavanja [28]. To pokazuje velik potencijal za daljnja istraživanja u klasificiranju složenijih okolina, poput istraživanja osjetljivosti na složene molekule kao što su tumorski markeri i strojnog učenja kao vrijednog i svestranog alata u primjeni neuromorfnog inženjerstva.

7.6 Metodički dio

Informatika je kao nastavni predmet relativno nova pojava, ali od velike važnosti za učenike. Zbog naglog razvoja tehnologije i računarstva kojemu svjedočimo u zadnjim desetljećima, Informatika je jedan od predmeta koji je podložan značajnim promjenama te njezin okvir nije u potpunosti utvrđen kao što je kod temeljnih predmeta kao što su Matematika ili Hrvatski jezik [6].

Tim promjenama mogu i samostalno svjedočiti, kako se npr. programski jezik Python nije podučavao tijekom mog pohađanja srednje škole, dok se sada obrađuju neke cjeline koje se spominju i na fakultetskoj razini. Predikcija koju se usudim iskazati jest da će važnost informatičke obrazovanosti i dalje imati uzlazni trend, a na obrazovnim ustanovama će biti teret prilagodbe školovanja modernim izazovima u društvu. Razne kompleksne informatičke teme perkolirat će prema nižim razredima škole, dok će metode podučavanja tih tema biti olakšane razvojem software-a.

Što se tiče Python-a, glavni principi dizajna tog programskog jezika poput jednostavnosti i čitkosti omogućuje lakše podučavanje učenika u računalnom razmišljanju. Uz to, Python kao jezik je jako popularan i svestran, zbog čega je i odabran za korištenje u nastavi većine hrvatskih gimnazija.

Kurikulum za nastavni predmet Informatike za osnovne škole i gimnazije u Republici Hrvatskoj [49] izdan 2018. navodi kategoriju "Računalno razmišljanje i programiranje" kao jednu od četiri glavne domene kojima se realiziraju ciljevi predmeta. Odgojno-obrazovni ishodi su zatim razrađeni u podjeli na razrede za osnovnu i srednju školu te su detaljnije formulirani u obliku preporuka za nastavnike, što omogućava slobodu nastavnika u odabiru metoda za ostvarivanje tih ciljeva.

Računalno razmišljanje i programiranje	Nakon četvrte godine učenja predmeta Informatika u srednjoj školi u domeni Računalno razmišljanje i programiranje učenik: B.4.1 rješava problem koristeći se apstraktnim strukturama podataka B.4.2 stvara aplikaciju s grafičkim korisničkim sučeljem za problem iz stvarnoga života B.4.3 koristi se modeliranjem i simulacijom za predstavljanje i razumijevanje prirodnih fenomena B.4.4 koristi se različitim programskim paradigmama za rješavanje problema iz stvarnoga života B.4.5 definira problem iz stvarnoga života i stvara programsko rješenje prolazeći sve faze programiranja, predstavlja programsko rješenje i vrednuje ga.

Slika 7.10: Primjer odgojno-obrazovnih ishoda domene Računalno razmišljanje i programiranje za 4. razred srednje škole.

[6] U kontekstu obrazovnog sustava Republike Hrvatske

Danas se susrećemo sa brojnim slučajevima primjene umjetne inteligencije, od raspoznavanja slika sve do samovozećih automobila. Vruća tema poput umjetne inteligencije nesumnjivo može zagolicati maštu učenika i ne bi bilo čudno da brojni učenici žele saznati nešto više, što bi im dalo odgovarajuću motivaciju kod učenja te nove i zanimljive teme. I dok UI obuhvaća veliki raspon interesantnih tema (koje bi se mogle iskoristiti u sklopu učeničkih projekata, čak i izvan područja Informatike), uporište za razvoj računalnog razmišljanja kod učenika može se pronaći u strojnom učenju (machine learning).

Kao što je vidljivo iz slike 7.10, razina učeničkog znanja i primjene potrebne za ostvarenje ishoda na razini 4. razreda srednje škole dovoljna je da bi se u program mogla uklopiti cjelina o neuronskim mrežama. Jednostavnost Python-a u kombinaciji sa brojnim paketima olakšava ostvarenje koda te se rješenje naizgled kompleksnog problema može zapisati u desetak linija koda.

I dok ovakav pogled podučavanja neuronskih mreža u srednjoj školi gleda prema budućnosti, pitanje konkretne implementacije će ovisiti o podršci za programski jezik i vezane pakete tj. biblioteke. Unatoč tome što jezik može postati star i nekorišten, glavne ideje strojnog učenja i neuronskih mreža će i dalje biti relevantne, pitanje je jedino u kojem pogledu će napredovati. Stoga bi i oblik neke buduće verzije jezika i koda mogao sadržavati te osnovne sastavnice uvedene u ovoj ideji primjene u nastavi.

Nastavna priprema - Neuronske mreže u nastavi

RAZINA	Opća gimnazija, Prirodoslovno-matematička gimnazija
RAZRED	4. razred
NASTAVNA CJELINA	Umjetna inteligencija
NASTAVNA JEDINICA	Neuronska mreže u Python-u
PREDVIĐENI BROJ SATI	1 sat

1. **GLAVNI CILJ NASTAVNOG SATA**

Učenici će implementirati neuronsku mrežu za raspoznavanje MNIST znamenki u Python-u koristeći Keras paket.

Sat je zamišljen kao dio cjeline o neuronskim mrežama, obrađivao bi se nakon teoretske cjeline o neuronskim mrežama tako da bi učenici trebali biti upoznati sa temeljnim konceptima tog gradiva.

2. **OČEKIVANA UČENIČKA POSTIGNUĆA**

(a) **temeljna znanja**

Učenici će:

- unijeti MNIST bazu podataka u program
- prikazati podatke koristeći Matplotlib biblioteku
- objasniti formate podataka
- konstruirati neuronsku mrežu pomoću Keras paketa
- promijeniti oblik podataka
- objasniti ulogu optimizatora i funkcije troška (gubitka)
- istrenirati neuronsku mrežu i prikazati rezultate

(b) **vještine i sposobnosti**

Učenici će:

- razvijati logičko, analitičko i proceduralno mišljenje
- razvijati sposobnost rješavanja problemskih zadataka

(c) **vrijednosti i stavovi**

Učenici će:

- razvijati vještinu koncentracije
- stjecati vještine točnosti, preciznosti i urednosti u radu
- razvijati povjerenje u vlastite informatičke sposobnosti
- razvijati sposobnost procjene vlastitih informatičkih sposobnosti

2. **KORELACIJE UNUTAR INFORMATIKE I S DRUGIM NASTAVNIM PREDMETIMA**

Matematika

3. **TIP NASTAVNOG SATA**

Sat obrade novog gradiva.

4. **NASTAVNI OBLICI**

Frontalna nastava, individualni rad.

5. **NASTAVNE METODE**

Metoda dijaloga, metoda demonstracije, vizualna metoda, verbalna metoda, problemska nastava

6. **NASTAVNA SREDSTVA**

Računalna prezentacija

7. **NASTAVNA POMAGALA**

Ploča, kreda, PC računalo i LCD projektor

8. **LITERATURA ZA UČITELJA/ICU**

Francois Chollet, Deep Learning with Python, Manning Publications, 2017.

MAKROPLAN (ARTIKULACIJA SATA)

1. **UVODNI DIO SATA (5 min)**

   Motivacija: Što su neuronske mreže? (2 min)

   Naslov: Implementacija Neuronske mreže u Python-u

   Aktivnost obrade: MNIST baza podataka (3 min)

2. **GLAVNI DIO SATA (35 min)**

   Aktivnost obrade: Prikaz podataka (znamenki) (10 min)

   Aktivnost obrade: Konstrukcija neuronske mreže (15 min)

   Aktivnost obrade: Obrada podataka (10 min)

3. **ZAVRŠNI DIO SATA (10 min)**

   Aktivnost obrade: Treniranje neuronske mreže (5 min)

   Aktivnost obrade: Komentiramo rezultate (5 min)

MIKROPLAN (DETALJNA RAZRADA MAKROPLANA)

1. **UVODNI DIO SATA**

- cilj
  - razvijamo motivaciju za korištenje neuronskih mreža
  - detaljno pojašnjavamo problem klasifikacije MNIST znamenki
  - unijeti MNIST bazu podataka u program
- nastavni oblik: Frontalna nastava
- nastavnu metodu: Metoda dijaloga, metoda demonstracije

Kratko ponavljanje : Što su neuronske mreže, što znate o njima?
Danas ćemo naučiti kako napraviti neuronsku mrežu u Python kodu koristeći Keras paket. Zapisujemo naslov:
Naslov: Neuronske mreže u Pythonu
Neuronske mreže služe kao algoritam strojnog učenja koji se može koristi u različitim problemima poput klasifikacije i regresije. Potrebno je odabrati zadatak koji ćemo dati neuronskoj mreži da riješi umjesto nas.

MNIST baza podataka

Problem s kojim ćemo se suočiti je prepoznavanje rukom zapisanih znamenki iz baze podataka MNIST. Taj skup podataka sastoji se odslika znamenki od 0 do 10 u sivim tonovima čija je rezolucija 28 x 28. MNIST se jako poznat problem klasifikacije koji se pojavljuje u brojnim člancima i radovima.
Da bismo koristili MNIST bazu podataka trebamo ju unijeti u naš program koristeći slijedeću liniju koda:

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Podatci su koji se sastoje od slika i njihovih odgovarajućih oznaka znamenki podijeljeni su u dvije kategorije, prvu grupu ćemo koristiti za treniranje neuronske mreže, dok će nam druga služiti da testiramo preciznost istrenirane mreže.

2. **GLAVNI DIO SATA**

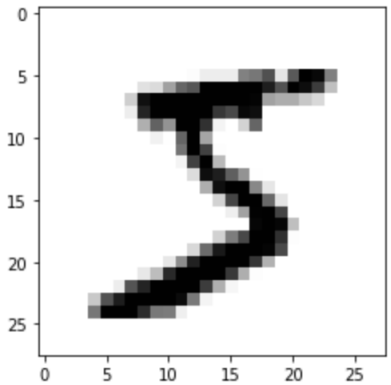
- cilj
  - prikazati podatke koristeći Matplotlib biblioteku
  - objasniti formate podataka
  - konstruirati neuronsku mrežu pomoću Keras paketa
  - promijeniti oblik podataka
- nastavni oblik: Frontalna nastava, individualni rad, problemska nastava
- nastavnu metodu: Metoda dijaloga, metoda demonstracije
- potrebni materijal: Bijela ploča, flomasteri, računalo

Prikaz znamenki

Da bismo olakšali razumijevanje baze podataka koju smo upravo unijeli, korisno je uzeti primjer pojedinačnog slučaja, da vidimo s čime točno radimo. Iskoristit ćemo Matplotlib paket kako bismo prikazali proizvoljnu znamenku.

```
import matplotlib.pyplot as plt


digit = train_images[0]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



Slika 7.11: Znamenka "5" spremljena na početnom položaju Numpy liste slike za treniranje MNIST databaze.

Kao što vidimo na slici 7.11, znamenka se sastoji od 28 x 28 piksela od kojih je svaki označen sa brojkom koja označava boju tog piksela, točnije koeficijent između 0 i 255. Možemo provjeriti hoće li pripadajuća oznaka odgovarati ovoj znamenki:

```
>>> train_labels[0]
5
```

Podatci su spremljeni u obliku Numpy matrica što znači da imamo 60000 slika i njihovih odgovarajućih znamenki što možemo provjeriti koristeći .shape argument.

```
>>> train_images.shape
(60000, 28, 28)
>>> train_labels.shape
60000,
```

U kojem obliku su spremljeni podatci za testiranje i zašto su nam oni korisni?
Podatci za testiranje imaju isti oblik kao i oni za testiranje, ali imamo manji broj uzoraka, točnije 10000. Neuronsku mrežu ćemo trenirati koristeći slike za treniranje, a kako bi provjerili da mreža dobro radi i da ju možemo primjenjivati na skupove podataka koje nije prethodno vidjela testirat ćemo ju na testnim podatcima.
Kada neuronska mreža (ili neki drugi algoritam za strojno učenje) dobro prepoznaje slučajeve koje nije susrela znači da dobro *generalizira*.

Konstrukcija neuronske mreže

Koristeći Keras paket neuronsku mrežu je veoma jednostavno sastaviti, potrebno je samo odabrati arhitekturu mreže.

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(500, activation="relu", input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation="softmax"))
```

Koristeći ovakav kod konstuirali smo mrežu koja se sastoji od dva sloja neurona. Neuronska mreža kroz 784 (28 x 28) ulaznih neurona prima podatke iz slika znamenke. Središnji "skriveni" sloj sastoji se od 500 neurona, točnije ReLU aktivacijskih jedinica. Kao izlaz neuronska mreža daje listu od 10 vjerojatnosti kroz softmax aktivacijsku funkciju. Te vjerojatnosti označuju o kojoj ulaznoj znamenki bi se moglo raditi.

Obrada podataka

Prije nego što krenemo sa treniranjem neuronske mreže, podatci se trebaju prilagoditi u jednostavniji oblik koji odgovara neuronskoj mreži. Slike znamenki spremljeni u obliku matrice 28 x 28 možemo prilagoditi u listu duljine 784, a koeficijente ograničiti na interval [0,1] tako što ih podijelimo sa 255. To će u kodu izgledati ovako:

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

Također, izlazne podatke treba preobličiti iz samo jedne brojke znamenke u listu, gdje će pripadajuća znamenka biti označena sa 1, dok će ostale biti 0.

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

3. **ZAVRŠNI DIO SATA**

- cilj
  - objasniti ulogu optimizatora i funkcije troška (gubitka)
  - istrenirati neuronsku mrežu i prikazati rezultate
- nastavni oblik: Problemska nastava, individualni rad
- nastavnu metodu: Metoda dijaloga

Treniranje neuronske mreže i rezultati

Da bi trenirali neuronsku mrežu potrebno je odabrati funkciju troška ili gubitka (cost, loss function) koji će uspoređivati izlaz naše neuronske mreže sa traženim rezultatima te algoritam koji će uzimati to u obzir i prilagoditi mrežu kako bi davala bolje predikcije.

```
network.compile(optimizer="SGD",
                loss="categorical_crossentropy",
                metrics=["accuracy"])
```

U Keras paketu, algoritmi poput Stohastičkog gradijentnog spusta nalaze se u kategoriji optimizatora pod skraćenicom "SGD". Funkcija troška korištena u ovom problemu kategorizacije naziva se "categorical crossentropy". Napokon smo spremi istrenirati našu neuronsku mrežu koristeći *.fit* metodu koja kao argumente prima slike za treniranje, njihove oznake, broj epoha i broj ulaznih obzervacija koji se koristi u jednoj iteraciji učenja (batch size).

```
>>> network.fit(train_images, train_labels, epochs=1, batch_size=100)
Epoch 1/1
60000/60000 [==========] - 2s 33us/step - loss: 0.9801 - accuracy: 0.7847
```

Dobivena točnost naše mreže na skupu podataka za treniranje u ovom primjeru je 78%, što nije loše, ali bi mogla biti i veća.

Što možemo napraviti kako bi poboljšali rezultate neuronske mreže?

Postoji veliki broj mogućnosti, ali za početak mogli bi povećati broj epoha treniranja mreže. Ugađanje parametara neuronske mreže kako bi postigli bolje rezultate naziva se *hiperparametrizacijom*.

```
Epoch 5/5
60000/60000 [==========] - 2s 29us/step - loss: 0.3187 - accuracy: 0.9124
```

Treniranjem u 5 epoha možemo dobiti točnost od 91.24%. Ta brojka doduše nije reprezentativna pravoj preciznosti neuronske mreže na neviđenim primjerima, zbog čega smo prethodno spremili i dio podataka na kojoj ćemo ju testirati.

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print("test_acc:", test_acc)
10000/10000 [==============================] - 0s 32us/step
test_acc: 0.9196000099182129
```

Na skupu podataka za testiranje dobili smo nešto veću točnost od 91.96%. Kada se govori o razlici između uspjeha algoritma strojnog učenja na podatcima za treniranje u usporedbi sa onima za testiranje pojavljuje se termin *pretjeranog ili premalog ugađanja* (overfitting - underfitting).

Kako bi poboljšali performanse neuronske mreže možete se poigrati sa parametrima, isprobati drugačije optimizatore i slično, te izvjestiti o utjecaju na uspješnost rezultata.

Zaključak

Budući da rješavanje problema klasifikacije prati izloženi postupak obrade ulaznih i izlaznih podataka, konstrukcije neuronske mreže i provjera točnosti, predlažem da se u nastavi može odabrati par različitih i zanimljivih problema jednostavne prirode i učenike podijeliti u grupe te se posvetiti odvojeni sat u kojem bi učenici samostalno primjenjivali stečeno znanje.

Također kao izborni projekt predlažem pisanje programa za crtanje znamenki koji koristi napravljenu neuronsku mrežu za prepoznavanje nacrtanih znamenki.

Bibliography

[1] Francois Chollet, *Deep Learning with Python*, Manning Publications, 2017.

[2] Thomas Mitchell, *Machine Learning*, Mcgraw-Hill Higher Education, 1997.

[3] Monroe, D., "*Neuromorphic computing gets ready for the (really) big time*", Communications of the ACM. 57 (6): 13–15., 2014.

[4] Mead, Carver (1990). "*Neuromorphic electronic systems*", proceedings of the IEEE. 78 (10): 1629–1636., 1990.

[5] Mitrokhin A., Sutor P., Fermüller C. and Aloimonos Y., "*Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception*", Science Robotics, 2019.

[6] Sundaram, S., Kellnhofer, P., Li, Y. et al., "*Learning the signatures of the human grasp using a scalable tactile glove*", Nature 569, 698–702, 2019.

[7] Osborn et al., *Prosthesis with neuromorphic multilayered e-dermis perceives touch and pain*, Science Robotics, Vol. 3, Issue 19, 20 Jun, 2018.

[8] van Doremaele, Eveline R. W. and Gkoupidenis, Paschalis and van de Burgt, Yoeri, *Towards organic neuromorphic devices for adaptive sensing and novel computing paradigms in bioelectronics*, J. Mater. Chem. C, 7, 12754, 2019.

[9] Michael A. Nielsen, "*Neural Networks and Deep Learning*", Determination Press, 2015.

[10] Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT press, 2016.

[11] Millman & Halkias, *Integrated Electronics*, McGraw-Hill electrical and electronic engineering series, McGraw-Hill Publishing Company, 2001.

[12] Grigorescu, Sorin et al., *A Survey of Deep Learning Techniques for Autonomous Driving*, Journal of Field Robotics, 2019.

[13] Kumerički, Krešimir, *Measurability of pressure inside the proton*, Nature, 570, 7759; E1-E2, 2019.

[14] Christopher M. Bishop, *"Pattern Recognition and Machine Learning"*, Springer, 2006.

[15] Jacob T. Friedlein B.A., *Device physics and material science of organic electrochemical transistors*, A thesis submitted to the Faculty of the Graduate School of the University of Colorado, 2017.

[16] Hagen Klauk, *Organic Electronics Materials, Manufacturing, and Applications*, 2006.

[17] George Malliaras and Richard Friend, *An Organic Electronics Primer* Citation: Phys. Today 58(5), 53, 2005.

[18] Bernards, D. and Malliaras, G., *Steady-State and Transient Behavior of Organic Electrochemical Transistors*. Adv. Funct. Mater., 17: 3538-3544. 2007.

[19] Joy C. Perkinson, *Organic Field-Effect Transistors* November 19, 2007.

[20] Rivnay, J., Inal, S., Salleo, A., Owens, R. M., Berggren, M., & Malliaras, *Organic electrochemical transistors*, Nature Reviews Materials, 3, 2018.

[21] Wang, D., Noël, V., & Piro, B., *Electrolytic Gated Organic Field-Effect Transistors for Application in Biosensors—A Review*, Electronics, 5(1), 9. 2016.

[22] Janelle Leger, Magnus Berggren, Sue Carter, *Iontronics: Ionic Carriers in Organic Electronic Materials and Devices*, CRC Press, 2010.

[23] Rivnay, J., Inal, S., Collins, B. et al., *Structural control of mixed ionic and electronic transport in conducting polymers.*. Nat Commun 7, 11287 2016.

[24] Anna Shirinskaya, Gilles Horowitz, Jonathan Rivnay, George G. Malliaras and Yvan Bonnassieux *Numerical Modeling of an Organic Electrochemical Transistor.* Biosensors (Basel), 2018.

[25] Inal, S., Malliaras, G.G. & Rivnay, J., *Benchmarking organic mixed conductors for transistors.*, Nat Commun 8, 1767, 2017.

[26] Khodagholy, D., Rivnay, J., Sessolo, M. et al. *High transconductance organic electrochemical transistors.* Nat Commun 4, 2133, 2013.

[27] Friedlein, J. T., McLeod, R. R., & Rivnay, J., *Device physics of organic electrochemical transistors.* Organic Electronics, 63, 398-414. 2018.

[28] Pecqueur S., Vuillaume D., Crljen Z., Lončarić I. & Zlatić V., *A Neural Network to Decipher Organic Electrochemical Transistors' Multivariate Responses for Cation Recognition*, ACS sensors (submitted), 2020.

[29] Pecqueur S., Guérin D., Vuillaume D. & Alibart F., *Cation Discrimination in Organic Electrochemical Transistors by Dual Frequency Sensing*, Org. Electron. 57, 232-238, 2018.

[30] S. Pecqueur, D. Vuillaume, I. Lončarić & V. Zlatić, *The non-ideal organic electrochemical transistors impedance* Organic Electronics Volume 71 issue, 2019.

[31] Andreas C. Müller, Sarah Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*, O'Reilly Media, Incorporated, 2016.

[32] Kevin P. Murphy, *Machine Learning: A Probabilistic Perspective* The MIT Press, 2012.

[33] Hebb, D. O., *The Organization of Behavior: A Neuropsychological Theory*, 1948.

[34] McCulloch, W.S., Pitts, W., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 115–133, 1943.

[35] Rosenblatt, Frank; *The Perceptron—a perceiving and recognizing automaton*, Report 85-460-1. Cornell Aeronautical Laboratory, 1957.

[36] Rumelhart, D., Hinton, G. & Williams, R., *Learning representations by back-propagating errors*, Nature 323, 533–536 1986.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, arXiv:1502.01852, 2015.

[38] Python programming language, `https://www.python.org/`, 18.9.2020.

[39] NumPy package, `http://www.numpy.org/`, 18.9.2020.

[40] Scipy package, `https://www.scipy.org/`, 18.9.2020.

[41] Matplotlib package, https://matplotlib.org/, 18.9.2020.

[42] Plotly package, https://plotly.com/, 18.9.2020.

[43] Scikit-learn package, https://scikit-learn.org/, 18.9.2020.

[44] Pandas package, https://pandas.pydata.org/, 18.9.2020.

[45] Keras API, https://keras.io/, 18.9.2020.

[46] Tensorflow library, https://www.tensorflow.org/, 18.9.2020.

[47] Jupyter notebook, https://jupyter.org/, 18.9.2020.

[48] Anaconda distribution, https://www.anaconda.com/, 18.9.2020.

[49] Odluka o donošenju kurikuluma za nastavni predmet Informatike za osnovne škole i gimnazije u Republici Hrvatskoj, Ministarstvo znanosti i obrazovanja, 6.3.2018.