University of Novi Sad
Faculty of Sciences
Department of Mathematics
and Informatics

Recurrent Neural Networks with Application on Drought Prediction

Master Thesis

Nemanja Filipović

Supervisor: dr Sanja Brdar

Novi Sad, 2019

'If you don't know how, observe the phenomena of nature, they will give you clear answers and inspiration."

Nikola Tesla

Abstract

The aim of this thesis is the application of recurrent neural networks with different optimization algorithms on problem of drought prediction. There are more types of droughts, meteorological, hydrological, agricultural and socioeconomic. In this thesis we focused on agricultural drought which is dependent on soil moisture content and for that reason we developed model which is able to predict soil moisture content in the first layer based on meteorological parameters from past days and also meteorological parameters for future days using Long Short Term Memory (LSTM) network. In the first part of the thesis we elaborated on mathematics foundations of neural networks and explained in detail how LSTM network works. In the second part of the thesis we presented obtained results on predicting agriculture drought in Serbia with data from Copernicus Climate Change Service.

Acknowledgements

I would like to express deepest and sincere gratitude to my thesis supervisor dr. Sanja Brdar for her motivation and support during research on this topic and also I want to thank dr. Gordan Mimic for his help with meteorological data and to all others from BioSense Institute who contributed to final form of this work by their suggestions.

Finnaly, the most important I sincerely want to thank to my family for their unfailing support and help during my studies.

Contents

1 Introduction **10**

2 Artificial Neural Network **12**
2.1 History of ANN 12
2.2 Perceptron 13
2.3 Perceptron Algorithm 15
2.3.1 Convergence of Perceptron Algorithm 15
2.4 Multilayer Perceptron 17

3 Recurrent Neural Network **18**
3.1 The Basics of RNN 18
3.2 Backpropagation Through Time 20
3.2.1 Vanishing and Exploding Gradient 22
3.3 Long Short Term Memory 23
3.3.1 Backpropagation Through Time in LSTM 24
3.3.2 Parameters Learning 27

4 Optimization Algorithms **28**
4.1 The Basics of Mathematical Optimization 28
4.2 Gradient Descent Method 32
4.2.1 Convergence Analysis 35
4.2.2 Gradient Descent Variants 36
4.3 First Order Optimizers 39
4.3.1 Momentum 39
4.3.2 Nesterov Accelerated Gradient 40
4.3.3 Adagrad 41
4.3.4 Adadelta 42
4.3.5 RMSprop 43

4.3.6 Adam 44
4.3.7 AdaMax 45
4.3.8 Nadam 46

5 Drought Prediction **48**
5.1 Data 48
5.2 Experiments 50
5.3 Results 53

6 Conclusion **66**

Bibliography **70**

Biography **71**

A Neural Networks **72**
A.1 RNN 72
A.1.1 Activation and Loss Functions 72
A.1.2 Derivative of Softmax and Cross Entropy Function . . 72
A.1.3 Derivative of Hyperbolic Tangent Function 74

List of Figures

2.1 Biological and artificial neuron 12
2.2 Perceptron 14
2.3 Multilayer perceptron 17

3.1 Unfolded RNN 19
3.2 Sequence models 20
3.3 LSTM 23
3.4 Unfolded memory unit of LSTM 26

4.1 Convex and non convex set 29
4.2 Tangents on convex function 29
4.3 Convex and non convex function 30
4.4 Stochastic and normal gradient 37
4.5 Saddle point 38
4.6 Local and global optimum 39
4.7 Momentum 39
4.8 Nesterov vector update 40

5.1 Summer 2012. 49
5.2 Input and output 52
5.3 Stacked LSTM network with p = 50 53
5.4 Loss functions after 200 epochs for stacked LSTM with p = 50 54
5.5 Scatter plot true vs predicted data 55
5.6 Palic 56
5.7 Sombor 57
5.8 Novi Sad 57
5.9 Zrenjanin 57
5.10 Kikinda 58

5.11 Banatski Karlovac 58
5.12 Loznica 58
5.13 Sremska Mitrovica 59
5.14 Valjevo 59
5.15 Beograd 59
5.16 Kragujevac 60
5.17 Smederevska Palanka 60
5.18 Veliko Gradiste 60
5.19 Crni Vrh 61
5.20 Negotin 61
5.21 Zlatibor 61
5.22 Sjenica 62
5.23 Pozega 62
5.24 Kraljevo 62
5.25 Kopaonik 63
5.26 Kursumlija 63
5.27 Krusevac 63
5.28 Cuprija 64
5.29 Nis 64
5.30 Leskovac 64
5.31 Zajecar 65
5.32 Dimitrovgrad 65
5.33 Vranje 65

A.1 Activation function 73

List of Tables

5.1 Meteorological stations in Serbia 51
5.2 Mean absolute error and mean absolute percentage accuracy for different p on test data 54
5.3 Mean absolute error and mean absolute percentage accuracy for different optimizers on test data 55
5.4 Statistical tests 56

List of Abbreviations

ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
SGD	Stochastic Gradient Descent
NAG	Nesterov Accelerated Gradient

Chapter 1

Introduction

Artificial neural networks (ANNs) are machine learning models inspired by biological neural networks which are widely used today and have very big domain of applications such as image classification, speech recognition, natural language processing, computer vision, etc. The background of neural networks is strong mathematics and the following mathematics disciplines are the fundamentals for understanding this machine learning model: mathematical optimization, linear algebra, probability, statistics as well as high dimensional calculus.

Among challenges that today's world face, the important one is certainly climate change, which causes droughts and floods all over the world more frequently and more intense than in all history on record [1]. Motivation for this thesis is to combine these two things, which is very popular in 21st century. We will focus on drought prediction problem in Serbia using recurrent neural networks (RNN). Meteorological drought causes hydrological drought which causes agricultural drought which further causes socio-economic drought [2] and since the soil moisture content is correlated with agricultural drought that is a variable which we would like to forecast based on other meteorological parameters.

In summer of 2012 in Serbia there was a drought which caused decrease of maize yield for 50% [3] and further consequence was the occurrence of aflatoxin in milk [4]. Estimated agricultural production loss was up to USD 2 billion [5]. The summer of 2015 and 2017 was also warm and dry and in spring of 2014 there was a flood [6] [7] [8]. All this things indicate that we have experienced effects of climate change.

This thesis is organized in six chapters, where the first and the last chap-

ters provide introduction and conclusion. In the second chapter we introduced artificial neural networks with historical view and represented perceptron and multilayer perceptron as a basic neural networks. The chapter three explains RNNs which are class of ANN well suited for time series data and for regression problems. We also introduced LSTM network and explained backpropagation through time in such network. In the chapter four we focused on the most popular first order optimization algorithms in neural networks with introduction based on convex mathematical optimization. Finally, in the chapter five we applied RNN on meteorological time series in order to predict soil moisture content with focus on different optimizer algorithms.

Chapter 2

Artificial Neural Network

In this section we introduce Artificial Neural Network by providing its brief history and then proceed with explaining perceptron, the most simplified neural network, convergence of perceptron algorithm and multilayer perceptron.

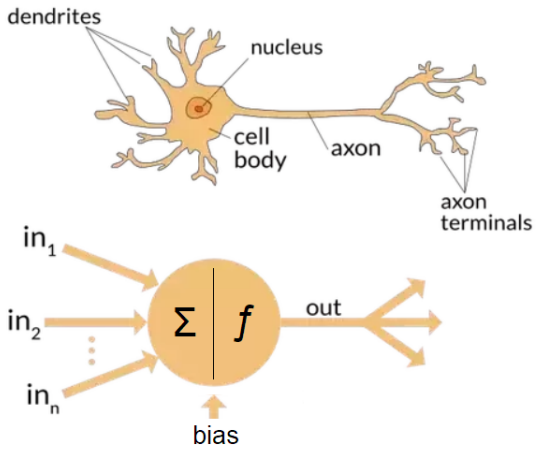


Figure 2.1: Biological and artificial neuron [9]

2.1 History of ANN

The first ANN is a result of collaboration between neurophysiologist Warren Sturgis McCulloch and logician Walter Harry Pitts from 1943 That is a math-

ematical model which imitates neurons from human brain. In 1949 Donald Olding Hebb published a book "The organization of behavior" where he exposed theory of Hebbian learning which claims that synaptic's value between two neurons is increasing when one neuron activates the other one. Frank Rosenblatt developed perceptrons in 1958 which was compound of variable synaptics weights and activation function. He also introduced the method of improving the networks weights in order to decrease classification error. Durin 50's and 60' there were two main approaches in this area artificial intelligence and connectionism. Connectionism was trying to build multilayer perceptron which can be able to classify in nonlinear separable space. This problem was solved during 80's when the method of back propagation was published [10]. In the 21st century neural networks expand with more powerful computers and there is a new field called Deep Learning.

2.2 Perceptron

ANN consists of neurons and edges between them. The most simplified neural network is perceptron [11]. We can define perceptron as a function composition:

$$f(g(x)), \qquad x \in \mathbb{R}^n \tag{2.1}$$

where g is linear function $g : \mathbb{R}^n \to \mathbb{R}$, $g(x) = \theta^T x + \theta_0$; $\theta \in \mathbb{R}^n$, $\theta_0 \in \mathbb{R}$ and f is a step function:

$$f(y) = \begin{cases} +1, & y \leq a \\ -1, & y > a. \end{cases} \qquad y, a \in \mathbb{R} \tag{2.2}$$

Perceptron takes input vector x then multiplies it with coefficient vector θ and then result goes through activation function f. Perceptron itself can be used for binary classification problem [10]. Let we have $\left\{x^1, x^2, ..., x^N\right\}$ data from two classes C_1 and C_2. Suppose that classes are linearly separable, we will show that perceptron can find line which separates classes. First, we can rewrite:

$$g(x) = \theta^T x + \theta_0 = \begin{bmatrix} \theta^T & \theta_0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \theta'^T x' \tag{2.3}$$

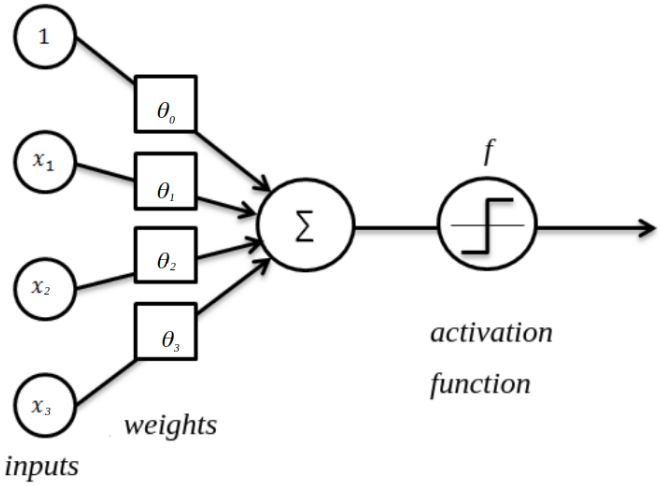


Figure 2.2: Perceptron
Source: `http://ataspinar.com/2016/12/22/the-perceptron/`

In further notation we can simply use $g(x) = \theta^T x$. We would like to find vector θ such that $\theta^T x = 0$ and

$$\begin{cases} \theta^T x > 0, & \forall x \in C_1 \\ \theta^T x < 0, & \forall x \in C_2. \end{cases} \tag{2.4}$$

We further define additional variable t that takes value $t^n = 1$ when $x^n \in C_1$, and $t^n = -1$ when $x^n \in C_2$. We introduce perceptron criterion function which is piecewise error function.

$$J_p(\theta) = -\sum_{x^n \in X_M} \theta^T (x^n t^n) \tag{2.5}$$

where X_M is set of misclassified samples.

This function is always non-negative and minimum, zero value, is reached when there is no misclassified samples. Obviously, perceptron criterion function is continuously and piecewise linear so we can use pattern by pattern gradient descent rule in order to find minimum.

Gradient is defined as $\nabla\theta J_p(\theta) = -\sum_{x^n \in X_M} x^n t^n$. Therefore, we have:

$$\theta_{k+1} = \theta_k - \eta_k \nabla J_p(\theta_k) = \theta_k + \eta_k \sum_{x^n \in X_M} x^n t^n \tag{2.6}$$

2.3 Perceptron Algorithm

1) Start with arbitrary solution θ_0.

2) Find set of misclassifed samples X_M.

3) Update $\theta_{k+1} = \theta_k + \eta_k \sum_{x^n \in X_M} x^n t^n$.

4) Go to the step 2) until there is no misclassified samples.

2.3.1 Convergence of Perceptron Algorithm

Let θ^* be solution of problem and α real, positive number.

$$\theta_{k+1} - \alpha\theta^* = \theta_k - \alpha\theta^* + \eta_k \sum_{x^n \in X_M} x^n t^n \tag{2.7}$$

$$\|\theta_{k+1} - \alpha\theta^*\|^2 = \|\theta_k - \alpha\theta^*\|^2 + \eta_k^2 \left\| \sum_{x^n \in X_M} x^n t^n \right\|^2 + 2\eta_k \sum_{x^n \in X_M} x^n t^n (\theta_k - \alpha\theta^*)^T \tag{2.8}$$

From $\sum_{x^n \in X_M} \theta_k^T x^n t^n < 0$, we have:

$$\|\theta_{k+1} - \alpha\theta^*\|^2 \leq \|\theta_k - \alpha\theta^*\|^2 + \eta_k^2 \left\| \sum_{x^n \in X_M} x^n t^n \right\|^2 - 2\eta_k \alpha \sum_{x^n \in X_M} x^n t^n \theta^{*T} \tag{2.9}$$

We define $\beta^2 = \max_{X_M \subseteq C_1 \cup C_2} \left\| \sum_{x^n \in X_M} x^n t^n \right\|^2$, in other words β^2 is the maximum value of vector norm among all possible subsets of given data.

Similarly, define $\gamma = \max\limits_{X_M \subseteq C_1 \cup C_2} \sum\limits_{x^n \in X_M} x^n t^n \theta^{*T}$. Notice, that the sum in this equation is negative. Therefore we can rewrite equation as follows:

$$\|\theta_{k+1} - \alpha\theta^*\|^2 \leq \|\theta_k - \alpha\theta^*\|^2 + \eta_k^2\beta^2 - 2\eta_k\alpha\gamma. \tag{2.10}$$

If we choose $\alpha = \frac{\beta^2}{2\gamma}$ and then apply upper inequality successively for k, k-1, ..., 0, then we have:

$$\|\theta_{k+1} - \alpha\theta^*\|^2 \leq \|\theta_0 - \alpha\theta^*\|^2 + \beta^2\left(\sum_{t=0}^{k}\eta_t^2 - \sum_{t=0}^{k}\eta_t\right) \tag{2.11}$$

Choose the sequence $\{\eta_t\}_{t=0}^k$, $\eta_t \in \mathbb{R}$ such that satisfy the following conditions:

$$1) \lim_{k\to\infty}\sum_{t=0}^{k}\eta_t = \infty \tag{2.12}$$

$$2) \lim_{k\to\infty}\sum_{t=0}^{k}\eta_t^2 < \infty \tag{2.13}$$

Then, there is k_0 such that right side of inequality 2.11 is non-positive. Therefore:

$$0 \leq \|\theta_{k_0+1} - \alpha\theta^*\| \leq 0 \tag{2.14}$$

In other words:

$$\theta_{k_0+1} = \alpha\theta^* \tag{2.15}$$

So, we proved convergence of perceptron algorithm if the condition is satisfied.

2.4 Multilayer Perceptron

In the previous section we introduced single layer perceptron and we proved that perceptron algorithm converges to line which separates classes from binary classification problem. Now, we are going to introduce multilayer perceptron which is able to classify non linear problems and also able to solve classification problems with more than two classes [10].

Multilayer Perceptron is an ANN with nodes and edges where each node represents neuron and the edges between nodes are directions. In other words, nodes and edges of ANN form a directed graph. Multilayer perceptron have more than two layers, input layer, output layer and at least one hidden layer (see Fig. 2.3).

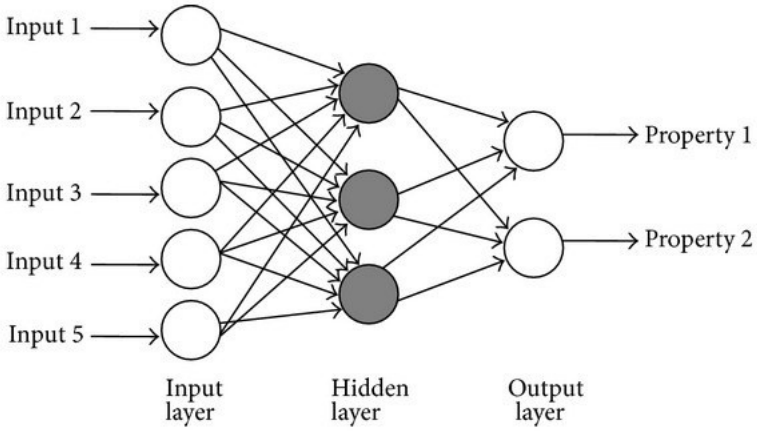


Figure 2.3: Multilayer perceptron [12]

In the multilayer perceptron each step activation function is replaced with differentiable non linear function in order to be able to calculate derivative. One popular function with such property is sigmoid function:

$$f(t) = \frac{1}{1 + e^{-at}}, \tag{2.16}$$

where a is the parameter of slope.

Multilayer perceptron is feed forward network which means there is only forward direction from one node to another, there are no cycles nor loops between nodes.

Chapter 3

Recurrent Neural Network

Recurrent neural networks are special class of ANNs. They are widely used in time series forecasting, handwriting recognition, natural language processing, speech recognition etc. RNNs are well suited for data which are given as sequences or time series, on other hand it is not good choice for standard classification problem. Nodes and edges of RNN create a directed graph with cycles and loops and this is the big difference with feed forward neural networks. RNNs are also known as neural networks with memory because they are capable to learn long-term dependencies along sequences. When the RNN produces an output it also takes the previous output or what it learned on the previous input, that is a reason why this neural network is called RNN [13].

3.1 The Basics of RNN

The simple RNN which we are going to introduce is vanilla RNN [14]. This RNN has input layer, hidden layer and output layer and there is a loop from hidden to hidden layer, this is very important property which distinguish RNN from other neural networks. That loop allows us to consider sequence problems and force network to learn long term dependency of sequences.

The RNN models dynamical system [15] which we can define as follows:

$$h_t = F(h_{t-1},\ x_t), \tag{3.1}$$

where x_t is the input at time t, h_t is the hidden state at time t and F is

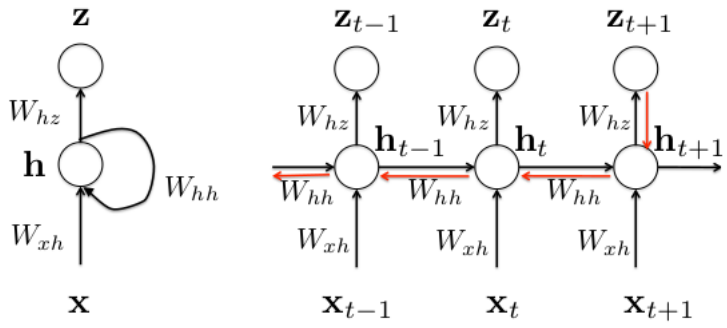


Figure 3.1: Unfolded RNN [15]

nonlinear mapping. We can derive:

$$h_t = F(F(h_{t-2},\ x_{t-1}),\ x_t) = F_2(h_{t-2},\ x_{t-1},\ x_t) \tag{3.2}$$

Let further derive for $t-2, t-3, ..., 1$. We obtain:

$$h_t = G(x_t,\ x_{t-1},\ ...,\ x_2,\ x_1), \tag{3.3}$$

where G is final composition of F functions. Therefore the current hidden state depends on whole past sequences.

Let us define Vanilla RNN (presented in Fig. 3.1) with hyperbolic tanget activation function in hidden unit and softmax activation function in the output unit:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{3.4}$$

$$z_t = softmax(W_{hz}h_t + b_z) \tag{3.5}$$

where b_h, b_z are bias vectors and W_{xh}, W_{hh}, W_{hz} are weighted matrices from input to hidden, hidden to hidden and hidden to output layer, respectively. Output prediction is z_t and activation functions are $tanh$ and $softmax$ (for definitions see A.1.1).

There are few types of architectures of RNN depending on input and output numbers (see Fig. 3.2). One to one is typically for image classification problem where for example input is image with fixed size and output is class. One to many type is used for sentiment analysis when we have as an input image with fixed size and the output is a sentence of words which describes picture. Many to one and many to many can be used for forecasting where for example an input can be weather data from previous days and output can be one day or few another days weather forecast. [16].

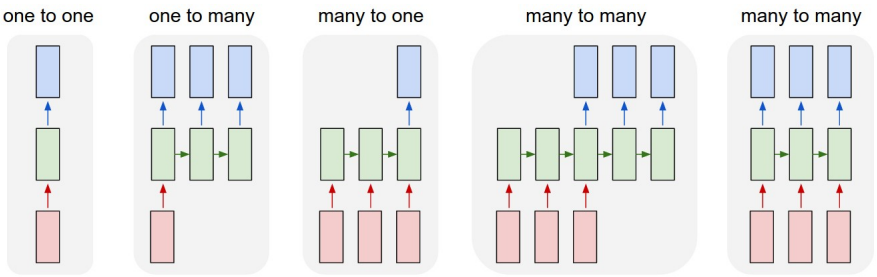


Figure 3.2: Sequence models [16]

The total loss of RNN is the sum over all timesteps or sequences:

$$J = \sum_t J_t \tag{3.6}$$

where t denotes time step and J_t the loss at time step t.

If we have sequence labeling problem, we will use for loss function cross entropy function

$$J_t = \sum_k y_k \log z_k \tag{3.7}$$

where $\mathbf{y}$ is vector which represents class, if we have 1 on i-th position and zero elsewhere than that is class i. Vector $\mathbf{z}$ represents probability distributions of the sample belonging to some class. Therefore, for vectors $\mathbf{y}$ and $\mathbf{z}$ is satisfied that sum of all entries is equal to one.

3.2 Backpropagation Through Time

Let's define $\alpha_t = W_{hz}h_t + b_z$, so we have $z_t = softmax(\alpha_t)$. Take derivative with respect to α_t (see A.1.2) and obtain the following:

$$\frac{\partial J}{\partial \alpha_t} = -(y_t - z_t) \tag{3.8}$$

The weight W_{hz} is shared across all time steps, therefore we can differentiate at each time step t and then sum all:

$$\frac{\partial J}{\partial W_{hz}} = \sum_t \frac{\partial J}{\partial z_t} \frac{\partial z_t}{\partial W_{hz}} \tag{3.9}$$

We can calculate gradient with respect to bias b_z:

$$\frac{\partial J}{\partial b_z} = \sum_t \frac{\partial J}{\partial z_t} \frac{\partial z_t}{\partial b_z} \tag{3.10}$$

Consider time step $t \to (t+1)$ and weight W_{hh} (see Fig. 3.1) we want to find gradient with respect to W_{hh}

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial W_{hh}} \tag{3.11}$$

W_{hh} is shared across all time steps according to the recursive definition in Eq. 3.4 and the hidden state h_{t+1} is dependent on hidden state h_t. Therefore at time $(t-1) \to t$ we have:

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} \tag{3.12}$$

To simplify further notation let denote with:

$$\frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \quad \dots \quad \frac{\partial h_{k+1}}{\partial h_k} \tag{3.13}$$

where $1 < k < t$.

When we back propagate from t to 0 to calculate gradient with respect to W_{hh} at time step $(t+1)$ for output z_{t+1} we obtain:

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \tag{3.14}$$

Aggregate the gradients w.r.t. W_{hh} over the whole time sequence with back propagation, we obtain:

$$\frac{\partial J}{\partial W_{hh}} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \tag{3.15}$$

Similarly we can calculate gradient w.r.t W_{xh}. Consider the time step $(t+1)$

$$\frac{\partial J_{t+1}}{\partial W_{xh}} = \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial W_{xh}} \tag{3.16}$$

Because h_{t+1} depends on h_t and x_{t+1}, we need to back propagate on h_t also. Therefore, consider at time step t:

$$\frac{\partial J_{t+1}}{\partial W_{xh}} = \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial W_{xh}} + \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial W_{xh}} \tag{3.17}$$

When we summing up through all t from t to 0 via back propagation we obtain the following:

$$\frac{\partial J_{t+1}}{\partial W_{xh}} = \sum_{k=1}^{t} \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{xh}} \tag{3.18}$$

Now, we can take derivative w.r.t W_{xh} over the whole sequence:

$$\frac{\partial J}{\partial W_{xh}} = \sum_{t} \sum_{k=1}^{t+1} \frac{\partial J_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{xh}} \tag{3.19}$$

3.2.1 Vanishing and Exploding Gradient

Considering term $\frac{\partial h_{t+1}}{\partial h_k}$ in Eq. 3.19 and deriving this expression for $k = t - n$ we get

$$\frac{\partial h_{t+1}}{\partial h_{t-n}} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \quad \ldots \quad \frac{\partial h_{t-n+1}}{\partial h_{t-n}} \tag{3.20}$$

In Eq. 3.20 we have n matrix multiplications, when we have for entries let's say some approximately q then we obtain:

$$q^n = \begin{cases} 0, & q < 1 \\ \infty, & q > 1 \end{cases} \tag{3.21}$$

which results in vanishing or exploding gradient.

This is main weakness of RNN, in the next section we are going to introduce special kind of RNN called long short term memory (LSTM) which is able to handle this problem.

3.3 Long Short Term Memory

Vanilla RNN use *tanh* activation function to make correlation between h_t, x_t and h_{t-1}. LSTM for that problem use memory cell which we are going to explain in detail in this section.

The memory cell c_t has x_t and h_{t-1} as input, the output is h_t same as for vanilla RNN, but it has input gate, output gate and forget gate which controls what is input and output of memory cell c_t (see Fig. 3.3). This property allow LSTM to handle vanishing problem.

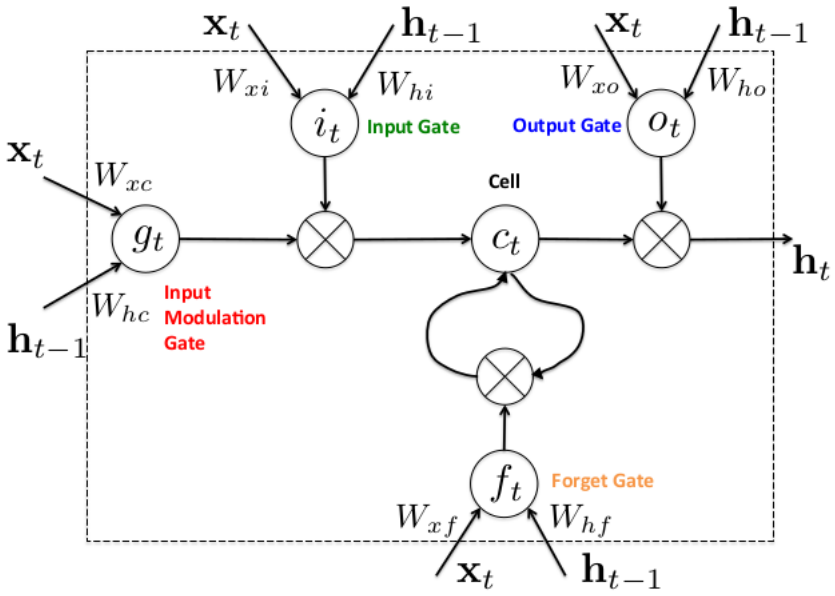


Figure 3.3: LSTM [15]

Let's define gates with sequence data $\{x_1, x_2, ..., x_T\}$ and weights W_{xi}, W_{hi} for input gate, W_{xc}, W_{hc} for input modulation gate, W_{xo}, W_{ho} for output gate and W_{xf}, W_{hf} for forget gate where first weight corresponds to input x_t and second weight corresponds to previously state h_{t-1}.

$$\mathbf{f_t} = \sigma(W_{xf}\mathbf{x_t} + W_{hf}\mathbf{h_{t-1}} + b_f) \tag{3.22}$$

$$\mathbf{i_t} = \sigma(W_{xi}\mathbf{x_t} + W_{hi}\mathbf{h_{t-1}} + b_i) \tag{3.23}$$

$$\mathbf{o_t} = \sigma(W_{xo}\mathbf{x_t} + W_{ho}\mathbf{h_{t-1}} + b_f) \tag{3.24}$$

$$\mathbf{g_t} = tanh(W_{xc}\mathbf{x_t} + W_{hc}\mathbf{h_{t-1}} + b_c) \tag{3.25}$$

$$\mathbf{c_t} = \mathbf{c_{t-1}} \odot \mathbf{f_t} + \mathbf{i_t} \odot \mathbf{g_t} \tag{3.26}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot tanh(\mathbf{c_t}) \tag{3.27}$$

$$z_t = softmax(W_{hz}\mathbf{h_t} + b_z) \tag{3.28}$$

3.3.1 Backpropagation Through Time in LSTM

Let the groundtruth at time t be y_t and for loss function we choose least squared function or cross entropy function. Considering the last layer with weight W_{hz} we can calculate derivative w.r.t to z_t and W_{hz}

$$dz_t = y_t - z_t \tag{3.29}$$

$$dW_{hz} = \sum_t \mathbf{h_t} dz_t \tag{3.30}$$

$$d\mathbf{h_T} = W_{hz} dz_T \tag{3.31}$$

where we only calculate gradient at last time step T, for all t we will discuss later. Calculate derivative with respect to gates and memory cell

$$d\mathbf{o_t} = tanh(\mathbf{c_t}) d\mathbf{h_t} \tag{3.32}$$

$$d\mathbf{c_t} = (1 - tanh(\mathbf{c_t})^2)\mathbf{o_t} d\mathbf{h_t} \tag{3.33}$$

$$d\mathbf{f_t} = \mathbf{c_{t-1}} d\mathbf{c_t} \tag{3.34}$$

$$d\mathbf{c_{t-1}} += \mathbf{f_t} \odot d\mathbf{c_t} \tag{3.35}$$

$$d\mathbf{i_t} = \mathbf{g_t} d\mathbf{c_t} \tag{3.36}$$

$$d\mathbf{g_t} = \mathbf{i_t} d\mathbf{c_t} \tag{3.37}$$

for calculating gradient w.r.t $tanh(c_t)$ see A.1.3. Derivatives w.r.t weights which correspond to input x, W_{xo}, W_{xi}, W_{xf} and W_{xc}:

$$dW_{xo} = \sum_t \mathbf{o_t}(1 - \mathbf{o_t})\mathbf{x_t} d\mathbf{o_t} \tag{3.38}$$

$$dW_{xi} = \sum_t \mathbf{i_t}(1 - \mathbf{i_t})\mathbf{x_t} d\mathbf{i_t} \tag{3.39}$$

$$dW_{xf} = \sum_t \mathbf{f_t}(1 - \mathbf{f_t})\mathbf{x_t} d\mathbf{f_t} \tag{3.40}$$

$$dW_{xc} = \sum_t (1 - \mathbf{g_t}^2)\mathbf{x_t} d\mathbf{g_t} \tag{3.41}$$

and because these weights are shared across the whole sequence we take the sum over t.

Similarly, we have:

$$dW_{ho} = \sum_t \mathbf{o_t}(1 - \mathbf{o_t})\mathbf{h_{t-1}} d\mathbf{o_t} \tag{3.42}$$

$$dW_{hi} = \sum_t \mathbf{i_t}(1 - \mathbf{i_t})\mathbf{h_{t-1}} d\mathbf{i_t} \tag{3.43}$$

$$dW_{hf} = \sum_t \mathbf{f_t}(1 - \mathbf{f_t})\mathbf{h_{t-1}} d\mathbf{f_t} \tag{3.44}$$

$$dW_{hc} = \sum_t (1 - \mathbf{g_t}^2)\mathbf{h_{t-1}} d\mathbf{g_t} \tag{3.45}$$

and corresponding hidden states at the current time step $(t-1)$:

$$\begin{aligned} d\mathbf{h_{t-1}} = \mathbf{o_t}(1-\mathbf{o_t})W_{ho}d\mathbf{o_t} + \mathbf{i_t}(1-\mathbf{i_t})W_{hi}d\mathbf{i_t} \\ +\mathbf{f_t}(1-\mathbf{f_t})W_{hf}d\mathbf{f_t} + (1-\mathbf{g_t})^2 W_{hc}d\mathbf{g_t} \end{aligned} \tag{3.46}$$

$$d\mathbf{h_{t-1}} += W_{hz}dz_{t-1} \tag{3.47}$$

where we calculate derivative of hidden state from two sources, one from the gates (Eq. 3.22 - 3.25) and the other one from Eq. 3.31 at time step $(t-1)$.

Now, we are going to explain how to derive gradients in detail, especially Eq. 3.35. Let we have least square objective function

$$J(\mathbf{x}, \theta) = min \sum_t \frac{1}{2}(y_t - z_t)^2 \tag{3.48}$$

where $\theta = \{W_{hz}, W_{xo}, W_{xi}, W_{xf}, W_{xc}, W_{ho}, W_{hi}, W_{hf}, W_{hc}\}$. For simplicity in the following we will use $J_t = \frac{1}{2}(y_t - z_t)^2$.

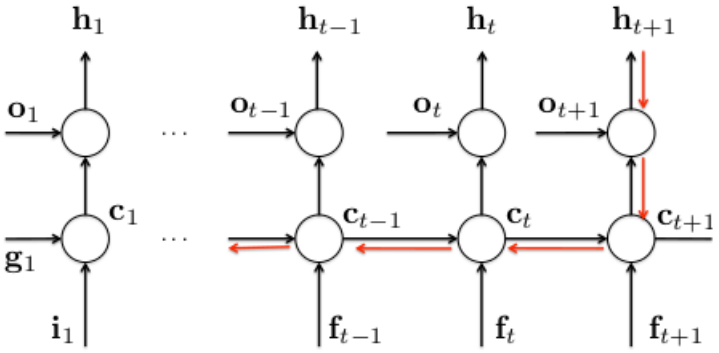


Figure 3.4: Unfolded memory unit of LSTM [15]

Take derivative at time step T w.r.t $\mathbf{c}_T$

$$\frac{\partial J_T}{\partial \mathbf{c}_T} = \frac{\partial J_t}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T} \tag{3.49}$$

At the time step $T-1$ take the derivative of J_{T-1} w.r.t $\mathbf{c}_{T-1}$:

$$\frac{\partial J_{T-1}}{\partial \mathbf{c}_{T-1}} = \frac{\partial J_{T-1}}{\partial \mathbf{h}_{T-1}}\frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} \tag{3.50}$$

As we can see on Fig. 3.4 the error is not propagated from $\mathbf{h}_{T-1}$ only, it is propagated from $\mathbf{c}_T$ as well. Therefore, we have:

$$\frac{\partial J_{T-1}}{\partial \mathbf{c}_{T-1}} = \frac{\partial J_{T-1}}{\partial \mathbf{c}_{T-1}} + \frac{\partial J_{T-1}}{\partial \mathbf{c}_T} \tag{3.51}$$

$$\frac{\partial J_{T-1}}{\partial \mathbf{c}_{T-1}} = \frac{\partial J_{T-1}}{\partial \mathbf{h}_{T-1}} \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{c}_{T-1}} + \frac{\partial J_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{c}_T} \frac{\partial \mathbf{c}_T}{\partial \mathbf{c}_{T-1}} \tag{3.52}$$

Now, using Eq. 3.26 at time step T-1 and Eq. 3.52 we can write:

$$d\mathbf{c_{T-1}} += \mathbf{f_T} \odot d\mathbf{c_T} \tag{3.53}$$

Similarly, we can derive Eq. 3.35 at any time step.

3.3.2 Parameters Learning

Forward: From the time step 1 to T we can use Eq. 3.22 - 3.26 to update states of forward neural network.

Backword: We can backpropagate error from T to 1 using Eq. 3.32 - 3.47. When we calculate derivatives we can use gradient based method to update parameter $\theta = \{W_{hz}, W_{xo}, W_{xi}, W_{xf}, W_{xc}, W_{ho}, W_{hi}, W_{hf}, W_{hc}\}$:

$$\theta_{k+1} = \theta_k - \eta_k d\theta_k \tag{3.54}$$

where η_k is learning rate at step k.

Chapter 4

Optimization Algorithms

In this section we are going to introduce the most popular optimization algorithms which neural networks typically use. Mathematics background of this algorithm is mathematical optimization, in other words finding point for which function has minimum value. We will focus on convex optimization problems, problem in neural network is non convex optimization problem in general but we can consider non convex function as convex on segments.

4.1 The Basics of Mathematical Optimization

First of all, we need to give some definitions[17].

Definition 1. *A set C is convex if the line segment between any two point from C lies in C, i.e., if for any $\theta_1,\ \theta_2\ \in\ C$ and any γ with $0 \leq \gamma \leq 1$ we have:*

$$\gamma\theta_1 + (1-\gamma)\theta_2\ \in\ C \tag{4.1}$$

Definition 2. *A function $J : \mathbb{R}^n \to \mathbb{R}$ is convex if dom J is convex set and if for all $\theta_1,\ \theta_2\ \in$ dom J and γ with $0 \leq \gamma \leq 1$, we have:*

$$J(\gamma\theta_1\ +\ (1-\gamma)\theta_2)\ \leq\ \gamma J(\theta_1)\ +\ (1\ -\ \gamma)J(\theta_2) \tag{4.2}$$

Geometrically speaking, function is convex if for every tangent of that function graph is above tangent line (see Fig. 4.2).

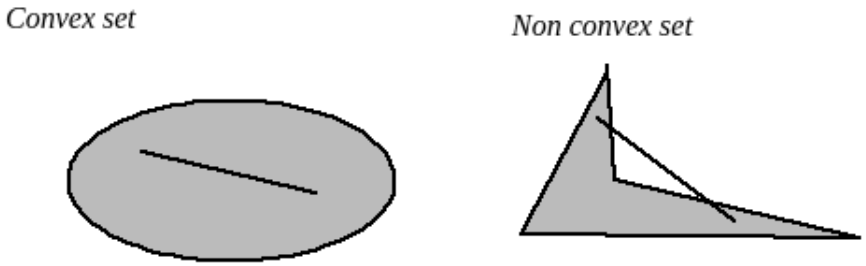


Figure 4.1: Convex and non convex set

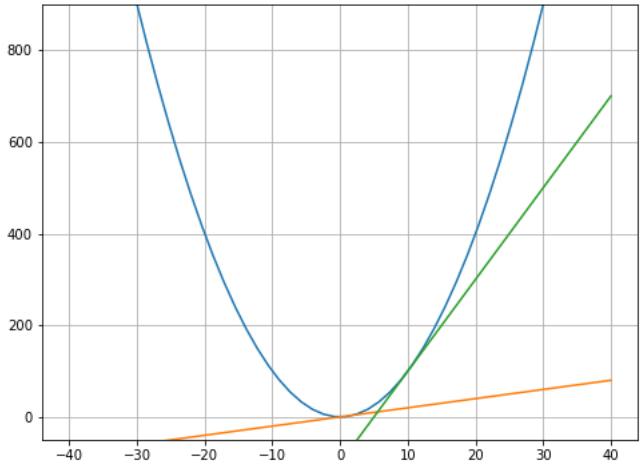


Figure 4.2: Tangents on convex function

Finding minimum of function we can write as follows:

$$\min_{\theta} J(\theta) \tag{4.3}$$

where $J : \mathbb{R}^n \to \mathbb{R}$ is convex and twice continuously differentiable function and θ belongs to some space S. We will also assume that there is optimal point θ^* such that $\theta^* = \arg\min_{\theta} J(\theta)$ and J is twice differentiable means that

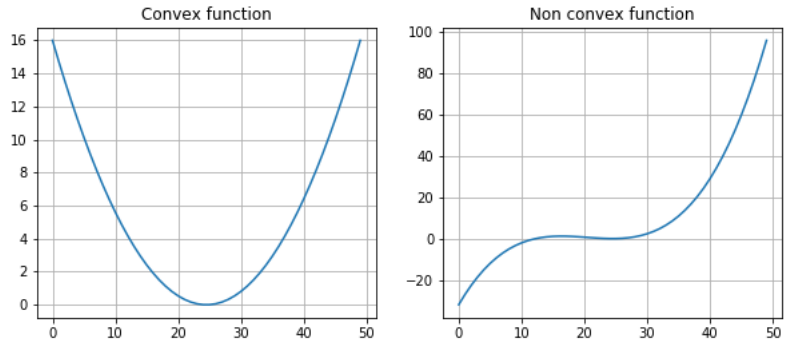


Figure 4.3: Convex and non convex function

derivative function of J has derivative in each point from domain.

Let's define optimal points in more mathematical sense:

Definition 3. *A point θ^* is a local minimizer of the objective function J if there exists $\varepsilon > 0$ such that $J(\theta^*) \leq J(\theta)$, for all θ such that $||\theta^* - \theta|| \leq \varepsilon$.*

Definition 4. *A point θ^* is a global minizer of the objective function J if $J(\theta^*) \leq J(\theta)$ for all θ from domain of J.*

Theorem 1. *Let $J : \mathbb{R}^n \to \mathbb{R}$, $S \subseteq \mathbb{R}^n$ is convex set. If J is convex on S then $J(\theta_1) \geq J(\theta_2) + \nabla J(\theta_2)^T(\theta_1 - \theta_2)$ for all θ_1 and θ_2 from S.*

Proof. Let's define $\omega = \gamma\theta_1 + (1-\gamma)\theta_2$ and $\theta_1, \theta_2 \in S$. J is convex, by definition we have

$$J(\gamma\theta_1 + (1-\gamma)\theta_2) \leq \gamma J(\theta_1) + (1-\gamma)J(\theta_2) \tag{4.4}$$

$$J(\omega) - J(\theta_2) \leq \gamma J(\theta_1) - \gamma J(\theta_2) \tag{4.5}$$

$$\frac{J(\omega) - J(\theta_2)}{\gamma} \leq J(\theta_1) - J(\theta_2) \tag{4.6}$$

Take the limit when $\gamma \to 0$

$$\lim_{\gamma\to 0}\frac{J(\omega)-J(\theta_2)}{\gamma}=\lim_{\gamma\to 0}\frac{J(\theta_2+\gamma(\theta_1-\theta_2))-J(\theta_2)}{\gamma}=\nabla^T J(\theta_2)(\theta_1-\theta_2) \tag{4.7}$$

Therefore we have:

$$\nabla^T J(\theta_2)(\theta_1-\theta_2)\leq J(\theta_1)-J(\theta_2) \tag{4.8}$$

□

The theorems 2 and 3 with simple proofs give us first and second order necessary conditions for function to have minimum value and also second order sufficient conditions in theorem 4 [18].

Theorem 2. *(First order necessary condition) Let function $J\in C^1$ and let θ^* be a local minimizer of function J, then $\nabla J(\theta^*)=0$.*

Proof. Let define function $\Phi(\lambda):=J(\theta^*+\lambda d)$, $\mathbf{d}\in\mathbb{R}^n$ arbitrary and fixed and $\lambda\in\mathbb{R}^+$. Note that function $\Phi:\mathbb{R}\to\mathbb{R}$, that is function of one variable and we know that first derivative at minimum is equal zero.

$$\Phi(0)=J(\theta^*)\leq J(\theta^*+\lambda d)=\Phi(\lambda) \tag{4.9}$$

$$\Phi(0)\leq\Phi(\lambda),\quad \lambda\geq 0\quad\Rightarrow\Phi'(0)=0 \tag{4.10}$$

$$\Phi'(\lambda)=\nabla J(\theta^*+\lambda d)^T d \tag{4.11}$$

$$\Phi'(0)=\nabla J(\theta^*)^T d=0 \tag{4.12}$$

$$\Rightarrow\nabla J(\theta^*)=0. \tag{4.13}$$

because $\mathbf{d}$ is an arbitrary and fixed vector. □

Theorem 3. *(Second order necessary condition) Let function $J\in C^2$. If θ^* is a local minizer then $\nabla J(\theta^*)=0$ and $\nabla^2 J(\theta^*)\geq 0$*

Proof. Now again consider function $\Phi(\lambda) = J(\theta^* + \lambda d)$, we know that $J \in C^2$, $\Phi'(0) = 0$ and $\Phi''(0) \geq 0$.

$$\Phi''(\lambda) = d^T \nabla^2 J(\theta^* + \lambda d) d \tag{4.14}$$

$$\Phi''(0) = d^T \nabla^2 J(\theta^*) d \geq 0 \tag{4.15}$$

$$\Rightarrow \nabla^2 J(\theta^*) \geq 0 \tag{4.16}$$

because d is an arbitrary and fixed vector.

□

Theorem 4. *(Sufficient conditions) Let $J \in C^2$. If $\nabla J(\theta^*) = 0$ and $\nabla^2 J(\theta^*) > 0$ then θ^* is strict local minimizer.*

Proof. Because the Hessian is continuous and positive definite at θ^* we can choose a radius $r > 0$ such that $\nabla^2 J(\theta)$ remains positive for all θ in the open ball $B = \{\omega \mid ||\omega - \theta^*|| < r\}$. Taking any nonzero vector p with $||p|| < r$, we have $\theta^* + p \in B$

$$J(\theta^* + p) = J(\theta^*) + p^T \nabla J(\theta^*) + \frac{1}{2} p^T \nabla^2 J(\omega) p \tag{4.17}$$

$$= J(\theta^*) + \frac{1}{2} p^T \nabla^2 J(\omega) p \tag{4.18}$$

where $\omega = \theta^* + tp$ for $t \in (0,\ 1)$. Since $\omega \in B$ we have $p^T \nabla^2 J(\omega) p > 0$ and therefore $J(\theta^* + p) > J(\theta^*)$.

□

4.2 Gradient Descent Method

We are going to introduce the algorithm which produces the minimizing sequence θ_k, k = 1,... where $\theta_{k+1} = \theta_k + \eta_k d_k$ and $\eta_k \geq 0$. The d_k is called step or search direction and η_k is called step length or step size. [17]

Descent method means that:

$$J(\theta_{k+1}) \leq J(\theta_k) \tag{4.19}$$

From convexity we know that:

$$J(\theta_{k+1}) - J(\theta_k) \geq \nabla J(\theta_k)^T(\theta_{k+1} - \theta_k) \tag{4.20}$$

left side of this inequality is less or equal zero from 4.19 which implies that $\nabla J(\theta_k)^T(\theta_{k+1} - \theta_k)$ is also less or equal zero. Therefore the search direction in a descent method must satisfy:

$$\nabla J(\theta_k)^T d_k \leq 0 \tag{4.21}$$

We call such direction a descent direction.

For descent direction the natural choice is the negative gradient $d_k = -\nabla J(\theta_k)$.

$$-\nabla J(\theta_k)^T \nabla J(\theta_k) = -||\nabla J(\theta_k)||^2 \leq 0 \tag{4.22}$$

so the negative gradient satisfy Eq. 4.21.

Algorithm

1) Choose arbitrary point θ_0 from *dom* J
2) Set $d_k = -\nabla J(\theta_k)$
3) Line search: Choose step size η_k
4) Update: $\theta_{k+1} = \theta_k + \eta_k d_k$
5) Repeat 2) - 4) until stopping criterion is satisfied.

The stopping criterion is in the form $||\nabla J(\theta_k)|| \leq \varepsilon$ where ε is small and positive number.

Exact Line Search

One of the line search method is exact line search in which η is choosen as:

$$\eta_k = \underset{\eta \geq 0}{\operatorname{argmin}}\, J(\theta_k + \eta d_k) \tag{4.23}$$

Sometimes in practise, we can easily calculate η_k using exact line search method by finding derivative of objective function w.r.t η.

Strong Convexity and Implications

Assume that objective function J is strongly convex which means that there exists $m, M > 0$ such that:

$$mI \preceq \nabla^2 J(\theta) \preceq MI \tag{4.24}$$

For θ_1 and $\theta_2 \in S$ we know from Taylor's theorem:

$$J(\theta_2) = J(\theta_1) + \nabla J(\theta_1)^T(\theta_2 - \theta_1) + \frac{1}{2}(\theta_2 - \theta_1)^T \nabla^2 J(z)(\theta_2 - \theta_1) \tag{4.25}$$

where $z \in [\theta_1, \theta_2]$.

From strong convexity we have:

$$J(\theta_2) \geq J(\theta_1) + \nabla J(\theta_1)^T(\theta_2 - \theta_1) + \frac{m}{2}||\theta_2 - \theta_1||^2 \tag{4.26}$$

for all θ_1 and $\theta_2 \in S$.

Now, we will show that inequality 4.26 can be used to obtain bound $J(\theta) - p^*$ where $p^* = \min_{\theta} J(\theta)$. First, the righthand side of inequality 4.26 is quadratic convex function of θ_2 for fixed θ_1, find the optimal $\hat{\theta_2}$ by setting gradient w.r.t θ_2 to zero:

$$\nabla J(\theta_1) + \frac{2m}{2}(\theta_2 - \theta_1) = \nabla J(\theta_1) + m\theta_2 - m\theta_1 \tag{4.27}$$

$$\Rightarrow \hat{\theta_2} = \theta_1 - \frac{1}{m}\nabla J(\theta_1) \tag{4.28}$$

Replace θ_2 with $\hat{\theta_2}$ in inequlity 4.26

$$J(\theta_2) \geq J(\theta_1) - \frac{1}{m}||\nabla J(\theta_1)||^2 + \frac{1}{2m}||\nabla J(\theta_1)||^2 \tag{4.29}$$

$$J(\theta_2) \geq J(\theta_1) - \frac{1}{2m}||\nabla J(\theta_1)||^2 \tag{4.30}$$

since this holds for any θ_2 we can set for lefthand side p^*. Therefore we have:

$$J(\theta_1) - p^* \leq \frac{1}{2m}||\nabla J(\theta_1)||^2 \tag{4.31}$$

Now, we can use that $\nabla^2 J(\theta) \preceq MI$ and we obtain from Eq. 4.25

$$J(\theta_2) \leq J(\theta_1) + \nabla J(\theta_1)^T(\theta_2 - \theta_1) + \frac{M}{2}||\theta_2 - \theta_1||^2 \tag{4.32}$$

Similarly, minimizing each side over θ_2 we have:

$$J(\theta_1) - p^* \geq \frac{1}{2M}||\nabla J(\theta_1)||^2 \tag{4.33}$$

4.2.1 Convergence Analysis

In this section we will show convergence of gradient descent algorithm with exact line search. Assume J is strongly convex on S in other words there are $m, M > 0$ such that $mI \preceq \nabla^2 J(\theta) \preceq MI$. Define function $\hat{J} : \mathbb{R} \to \mathbb{R}$ $\hat{J}(\eta) = J(\theta - \eta \nabla J(\theta))$ as a function of step size η for descent direction negative gradient.

From inequality 4.32 for $\theta_2 = \theta - \eta \nabla J(\theta)$ and $\theta_1 = \theta$:

$$\hat{J}(\eta) \leq J(\theta) + \nabla J(\theta)^T(\theta - \eta \nabla J(\theta) - \theta) + \frac{M}{2}||\theta - \eta \nabla J(\theta) - \theta||^2 \tag{4.34}$$

$$\hat{J}(\eta) \leq J(\theta) - \eta||\nabla J(\theta)||^2 + \frac{M\eta^2}{2}||\nabla J(\theta)||^2 \tag{4.35}$$

Assume that an exact line search is used and minimized over η both sides. Let η_{exact} represents step size which minimizes $\hat{J}$. Find the minimum value of righthand side of inequality 4.35 by setting gradient w.r.t. η to zero

$$-||\nabla J(\theta)||^2 + \frac{2M\eta}{2}||\nabla J(\theta)||^2 = (M\eta - 1)||\nabla J(\theta)||^2 = 0 \tag{4.36}$$

$$\Rightarrow \eta = \frac{1}{M} \tag{4.37}$$

Therefore we have:

$$\hat{J}(\eta_{exact}) \leq J(\theta) - \frac{1}{2M}||\nabla J(\theta)||^2 \tag{4.38}$$

Subtracting p^* from both sides

$$\hat{J}(\eta_{exact}) - p^* \leq J(\theta) - p^* - \frac{1}{2M}||\nabla J(\theta)||^2 \tag{4.39}$$

and combine with $||\nabla J(\theta)||^2 \geq 2m(J(\theta) - p^*)$ (inequality 4.31) we get:

$$\hat{J}(\eta_{exact}) - p^* \leq (1 - \frac{m}{M})(J(\theta) - p^*) \tag{4.40}$$

Let's rewrite upper inequality in algorithmic way:

$$J(\theta_{k+1}) - p^* \leq c(J(\theta_k) - p^*) \tag{4.41}$$

where $c = 1 - \frac{m}{M}$. Apply recursively for k

$$J(\theta_{k+1}) - p^* \leq c^2(J(\theta_{k-1}) - p^*) \tag{4.42}$$

and continue to applying recursively for k-1, ... 0, we obtain:

$$J(\theta_{k+1}) - p^* \leq c^k(J(\theta_0) - p^*) \tag{4.43}$$

We can conclude that $J(\theta_{k+1}) \to p^*$ when k goes to infinity because $c < 1$ and $c^k \to 0$.

We have proven convergence of gradient descent method with exact line search.

4.2.2 Gradient Descent Variants

In practise, there are three variants of gradient descent methods depending on amount of data which we use for calculating gradient. There is trade off between accuracy of gradient and time needed to calculate gradient [19].

Batch Gradient Descent

Batch gradient descent method uses entire dataset for calculating a gradient, thus this method can be very slow for a big dataset when we are training a neural network.

$$\theta_{k+1} = \theta_k - \eta_k \nabla J(\theta_k) \tag{4.44}$$

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) calculates gradient for each sample S_i from training set, in other words if entire dataset is $S = \{S_i\}_1^N$.

$$\theta_{k+1} = \theta_k - \eta_k \nabla J(\theta_k; S_i) \tag{4.45}$$

This method is faster but the data used for calculating gradient are small therefore we have a high variance.

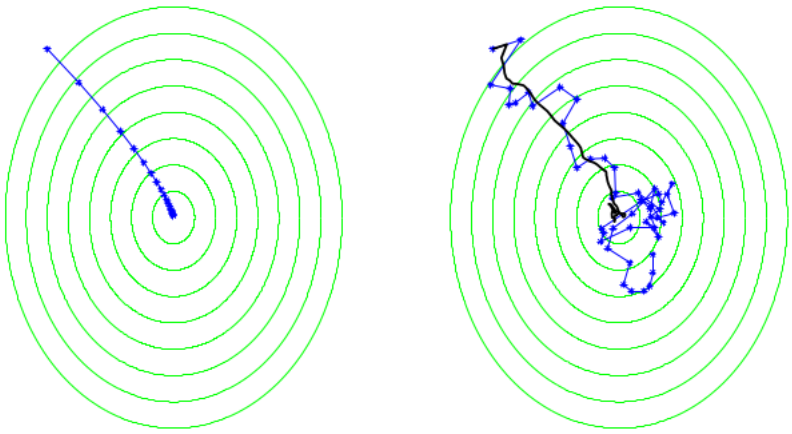


Figure 4.4: Stochastic and normal gradient [20]

Mini Batch Gradient Descent

Mini batch gradient descent is something between batch gradient descent and SGD method. It takes the best from both methods, reduce the variance and it is faster then batch gradient

$$\theta_{k+1} = \theta_k - \eta_k \nabla J(\theta_k; S_{(i+n)}) \tag{4.46}$$

Mini batch calculates the gradient on some subset $S_{(i+n)}$ of entire dataset and that subset is strictly greater than one sample S_i.

Challenges

There are some challenges with gradient descent methods, for example choosing a learning rate. If we choose too small learing rate we can obtain very slow convergence on the other hand for too big learning rate we can skip optimal point or algorithm can diverge.

Definition 5. *A pair* $(\tilde{\theta}_1, \tilde{\theta}_2)$ *and* $\tilde{\theta}_1 \in S_1$, $\tilde{\theta}_2 \in S_2$ *is saddle point of function* J *if:*

$$J(\tilde{\theta}_1, \theta_2) \leq J(\tilde{\theta}_1, \tilde{\theta}_2) \leq J(\theta_1, \tilde{\theta}_2) \tag{4.47}$$

for all $\theta_1 \in S_1$ *and* $\theta_2 \in S_2$. *In other words* $\tilde{\theta}_1$ *minimizes* $J(\theta_1, \tilde{\theta}_2)$ *and* $\tilde{\theta}_2$ *maximizes* $J(\tilde{\theta}_1, \theta_2)$.

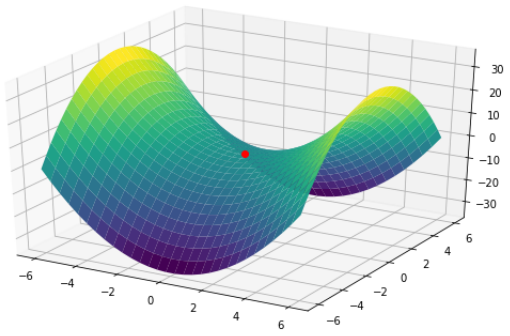


Figure 4.5: Saddle point

These optimization algorithms are typically used to optimize neural networks, in practise the loss objective function is non convex function and therefore when we use gradient descent direction we can finish in local optimum point which is not too good for our problem (see Fig. 4.6). Also there is saddle point [17] which is too difficult for gradient method to avoid because the gradient in all directions is zero.

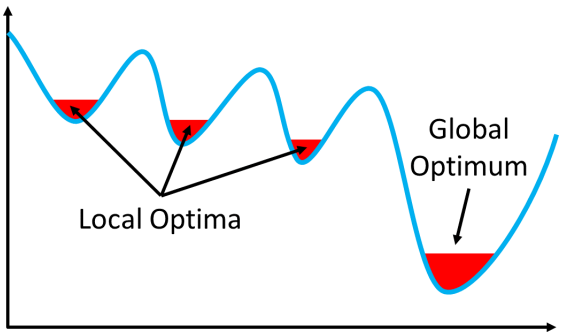


Figure 4.6: Local and global optimum
Source: By Christoph Roser at AllAboutLean.com under the free CC-BY-SA 4.0 license.

4.3 First Order Optimizers

We are focusing on optimization algorithms which use negative gradient for descent direction [19], there are also directions which use Hessian and that optimization algorithms is called second order. Using second order optimization algorithms in practise is not typical because calculating Hessian matrix can be computationally expensive. On the other hand first order optimization algorithms do not use information about surface which is hidden in the Hessian matrix.

4.3.1 Momentum

SGD has problem with ravines, when the surface is more steepest in one direction than in other. Then we have that SGD oscillates and slow converges to optimal point.

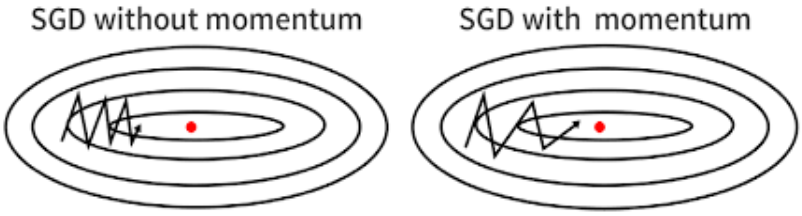


Figure 4.7: Momentum [19]

Momentum is a method which helps SGD to goes faster in relevant direc-

tion, we create a vector v_t which accumulates information about past updates as it follows:

$$v_k = \alpha v_{k-1} + \eta_k \nabla J(\theta_k) \tag{4.48}$$

$$\theta_{k+1} = \theta_k - v_k \tag{4.49}$$

where $\alpha \in (0, 1)$ is parameter which determine fraction of vectors from past time which is used. This parameter is called momentum term and typically choice is 0.9.

We can imagine momentum more intuitively like a ball which is pushed down the hill, the ball represents gradient and hill represents surface, when ball is going down it becomes faster and faster until ball reaches the lowest point.

4.3.2 Nesterov Accelerated Gradient

If we have just ball which goes downhill not considering where it is going, we can have a problem. Therefore we would like to have smart ball which can consider future position of itself and then decide where to go.

Nesterov accelerated gradient is a method which is able to handle this.

$$v_k = \alpha v_{k-1} + \eta_k \nabla J(\theta_k - \alpha v_{k-1}) \tag{4.50}$$

$$\theta_{k+1} = \theta_k - v_k \tag{4.51}$$

We also use momentum αv_{k-1} and calculate gradient on the next position $\theta_k - \alpha v_{k-1}$ and therefore we are using information from future.

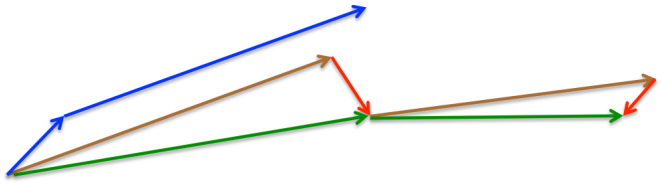


Figure 4.8: Nesterov vector update [19]

Momentum first calculates gradient at current point (first blue vector) and then takes a big jump in direction of accumulated gradient (second blue

vector). Nesterov update works in different way, it first makes a big jump in direction of accumulated gradient (brown vector) and then calculates a correction (green vector). This update prevent us from going too fast.

4.3.3 Adagrad

In previous methods we had a same learning rate for all parameters. In practice, there is a situations when we want a larger learning rate for infrequent parameters and smaller learning rate for frequent parameters.

Let $\theta = \{\theta_1, \theta_2, ..., \theta_m\}$ where each θ_i corresponds to different variable i, $i = 1, 2, ...m$.

Define $g_{k,i} = \nabla J(\theta_{k,i})$ as negative gradient at step k for parameter θ_i.

The SGD for each parameter θ_i at step $k+1$ is as follows:

$$\theta_{k+1,i} = \theta_{k,i} - \eta_k g_{k,i} \tag{4.52}$$

Now we will introduce adagrad (adaptive learning gradient) which modifies learning rate η at each step k for each parameter θ_i based on past gradients that have been computed for θ_i:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta_k}{\sqrt{G_{k,ii} + \varepsilon}} g_{k,i} \tag{4.53}$$

where $G_k \in \mathbb{R}^{m \times m}$ is diagonal matrix. $G_k = \sum_{\tau=1}^{k} g_\tau g_\tau^T$. Look at denominator in Eq. 4.53 we have the sum of the squares of the gradients up to step k and $\varepsilon > 0$ to avoid division by zero.

Since G_k is diagonal matrix which contains gradients w.r.t. all parameters θ_i we can now vectorize our implementation by performing an element-wise matrix-vector multiplication $\odot$ between G_k and g_k

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{G_k + \varepsilon}} \odot g_k \tag{4.54}$$

There is one possible problem with adagrad method. If we again consider denominator of Eq. 4.53 we can see that there is an accumulated sum which keeps growing while we update parameters, it can cause a very small learning rate and algorithm will not be able to use information of gradient.

4.3.4 Adadelta

To prevent very small learning rate in adagrad method a new method called adadelta is developed [21]. Adagrad accumulates all past squared gradients, adadelta instead of that restricts the window of accumulated past squared gradients to some fixed size w.

Storing w previously squared gradients is inefficient, therefore the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $E[g^2]_t$ depends on current gradient and the previous gradient as follows:

$$E[g^2]_k = \alpha E[g^2]_{k-1} + (1-\alpha){g_k}^2 \tag{4.55}$$

where α is similar to momentum term, which determine fraction of past gradients which we are going to use. Typical choice for α is 0.9 again.

Now, we are going to rewrite SGD in following notation:

$$\Delta\theta_k = -\eta_k g_{k,i} \tag{4.56}$$

$$\theta_{k+1} = \theta_k + \Delta\theta_k \tag{4.57}$$

The parameter update for Adagrad method:

$$\Delta\theta_k = -\frac{\eta_k}{\sqrt{G_k + \varepsilon}} \odot g_k \tag{4.58}$$

We now simply replace the diagonal matrix G_k with the decaying average over past squared gradients:

$$\Delta\theta_k = -\frac{\eta_k}{\sqrt{E[g^2]_k + \varepsilon}} g_k \tag{4.59}$$

The denominator of Eq. 4.59 is root mean squared error criterion of the gradient, we can replace it and obtain:

$$\Delta\theta_k = -\frac{\eta_k}{RMS[g]_k} g_k \tag{4.60}$$

The next idea of Adadelta method is correction of units. If the parameter had some hypothetical units, the changes to the parameter should be changes in those units as well. For example, SGD, Momentum and Adagrad does

not satisfy this property. The units in SGD and Momentum relate to the gradient:

$$\text{units of } \Delta\theta \propto \text{units of } g \propto \frac{\partial J}{\partial \theta} \propto \frac{1}{\text{units of } \theta} \tag{4.61}$$

Let's explain on example why $\frac{\partial J}{\partial \theta} \propto \frac{1}{\text{units of } \theta}$. If we have $J(\theta) = 5\theta$. Let write first derivative by definition and then choose for $\Delta\theta = 0.1$.

$$\frac{\partial J}{\partial \theta} = \lim_{\Delta\theta \to 0} \frac{J(\theta + \Delta\theta) - J(\theta)}{\Delta\theta} = \frac{5\theta + 5 * 0.1 - 5\theta}{0.1} = 0.5\frac{1}{0.1} \tag{4.62}$$

$$\Rightarrow \frac{\partial J}{\partial \theta} \propto \frac{1}{0.1} \tag{4.63}$$

The update should have the same hypothetical units as the parameter. First, define another exponentially decaying average of squared parameter updates:

$$E[\Delta\theta^2]_k = \alpha E[\Delta\theta^2]_{k-1} + (1 - \alpha)\Delta{\theta_k}^2 \tag{4.64}$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_k = \sqrt{E[\Delta\theta^2]_k + \varepsilon} \tag{4.65}$$

$\text{RMS}[\Delta\theta]_k$ is unknown at step k we assume that curvature is locally smooth and approximate with $\text{RMS}[\Delta\theta]_{k-1}$. Consider now Eq.4.60 the units in this parameter update follows from gradient so we need to replace η_k with $\text{RMS}[\Delta\theta]_{k-1}$ and obtain change of units in units of θ. Finally, the Adadelta update rule is:

$$\Delta\theta_k = -\frac{RMS[\Delta\theta]_{k-1}}{RMS[g]_k} g_k \tag{4.66}$$

$$\theta_{k+1} = \theta_k + \Delta\theta_k \tag{4.67}$$

4.3.5 RMSprop

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class. RMSprop is also developed to prevent small learning rate in Adagrad.

RMSprop update rule is:

$$E[g^2]_k = 0.9E[g^2]_{k-1} + 0.1{g_k}^2 \tag{4.68}$$

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{E[g^2]_k + \varepsilon}} g_k \tag{4.69}$$

As we can see, RMSprop is similar to Adadelta, the difference is that we do not replace η_k and difference with Adagrad is the choice of gradient, RMSprop chooses gradient which satisfies Eq. 4.68

4.3.6 Adam

Adam is adaptive moment estimation method which computes adaptive learning rates for each parameter like Adadelta, Adagrad and RMSprop. The difference between Adam and these methods is that Adam also keeps information about past gradients not only squared gradients [22]

$$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k \tag{4.70}$$

$$v_k = \beta_2 v_{k-1} + (1 - \beta_2) {g_k}^2 \tag{4.71}$$

where m_k and v_k are estimates of first moment (mean) and second moment (variance) of the gradients respectively. To counteract initialization bias we can use bias corrected first and second estimates as follows:

$$\hat{m_k} = \frac{m_k}{1 - {\beta_1}^k} \tag{4.72}$$

$$\hat{v_k} = \frac{v_k}{1 - {\beta_2}^k} \tag{4.73}$$

We obtain Adam update rule by placing m_k and v_k on relevant place in Eq. 4.69 (RMSprop update rule) or in Adadelta update rule.

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \varepsilon} \hat{m_k} \tag{4.74}$$

The default values for β_1, β_2 and ε are $0.9, 0.999$ and 10^{-8} respectively.

4.3.7 AdaMax

In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L^2 norm of their individual current and past gradients.

$$v_k = \beta_2 v_{k-1} + (1-\beta_2)|g_k|^2 \tag{4.75}$$

where $|g_k|^2$ represents one element from gradient vector which is related to individual weight.

We can generalize this to L^p norm and replace β_2 with β_2^p, we obtain the following;

$$v_k = \beta_2^p v_{k-1} + (1-\beta_2^p)|g_k|^p \tag{4.76}$$

$$= (1-\beta_2^p)\sum_{i=1}^{k}\beta_2^{p(k-i)}|g_i|^p \tag{4.77}$$

Now, let $p \to \infty$ and define $u_k = \lim_{p\to\infty}(v_k)^{1/p}$ then:

$$u_k = \lim_{p\to\infty}(v_k)^{1/p} = \lim_{p\to\infty}\left((1-\beta_2^p)\sum_{i=1}^{k}\beta_2^{p(k-i)}|g_i|^p\right)^{1/p} \tag{4.78}$$

$$= \lim_{p\to\infty}(1-\beta_2^p)^{1/p}\left(\sum_{i=1}^{k}\beta_2^{p(k-i)}|g_i|^p\right)^{1/p} \tag{4.79}$$

since $\beta_2 < 1$ the first term goes to one and we have:

$$= \lim_{p\to\infty}\left(\sum_{i=1}^{k}\left(\beta_2^{(k-i)}|g_i|\right)^p\right)^{1/p} \tag{4.80}$$

$$= \max\left(\beta_2^{k-1}|g_1|, \beta_2^{k-2}|g_2|, ..., \beta_2|g_{k-1}|, |g_k|\right) \tag{4.81}$$

and we can rewrite in recursively manner:

$$u_k = \max(\beta_2 u_{k-1}, |g_k|) \tag{4.82}$$

Finally, in Adam update rule we can replace $\sqrt{\hat{v_k}} + \varepsilon$ with u_k and obtain Adamax update rule:

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{u_k}\hat{m}_k \tag{4.83}$$

Good default values are again $\eta = 0.002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

4.3.8 Nadam

Adam is combination of RMSprop and momentum, from RMSprop Adam uses the decaying average of past squared gradients and from momentum Adam uses decaying average of past gradients. We have also seen that NAG is superior to momentum, therefore we would like to incorporate NAG into Adam. Nadam is combination of Adam and NAG which we obtain by modifying momentum term m_t from Adam [19].

First, recall the momentum update rule:

$$g_k = \nabla J(\theta_k) \tag{4.84}$$

$$m_k = \alpha m_{k-1} + \eta_k g_k \tag{4.85}$$

$$\theta_{k+1} = \theta_k - m_k \tag{4.86}$$

where J is our objective function, α is momentum term and η_k is step size at step k. Extending Eq. 4.86 we obtain:

$$\theta_{k+1} = \theta_k - (\alpha m_{k-1} + \eta_k g_k) \tag{4.87}$$

From this we can conclude that momentum takes a step in the direction of the previous momentum vector and in the direction of the current gradient.

NAG allow us to perform a more accurate step in the gradient direction by updating the parameters with the momentum step before computing the gradient. Let's write this:

$$g_k = \nabla J(\theta_k - \alpha m_{k-1}) \tag{4.88}$$

$$m_k = \alpha m_{k-1} + \eta_k g_k \tag{4.89}$$

$$\theta_{k+1} = \theta_k - m_k \tag{4.90}$$

We can modify this in order to perform momentum step just once instead two times like above (4.88 - 4.90)

$$g_k = \nabla J(\theta_k) \tag{4.91}$$

$$m_k = \alpha m_{k-1} + \eta_k g_k \tag{4.92}$$

$$\theta_{k+1} = \theta_k - (\alpha m_k + \eta_k g_k) \tag{4.93}$$

In the Eq. 4.87 we use momentum term at step $k-1$ now we replace it with momentum term m_k at current step k. Now, recall the Adam update rule:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \tag{4.94}$$

$$\hat{m_k} = \frac{m_k}{1-\beta_1^k} \tag{4.95}$$

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \varepsilon} \hat{m_k} \tag{4.96}$$

Expand the Eq.4.96 with definitions of $\hat{m_k}$ and m_k:

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \varepsilon} \left(\frac{\beta_1 m_{k-1}}{1-\beta_1^k} + \frac{(1-\beta_1) g_k}{1-\beta_1^k} \right) \tag{4.97}$$

Note that $\frac{\beta_1 m_{k-1}}{1-\beta_1^k} = \hat{m}_{k-1}$ and replace it:

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \varepsilon} \left(\beta_1 \hat{m}_{k-1} + \frac{(1-\beta_1) g_k}{1-\beta_1^k} \right) \tag{4.98}$$

We can now add Nesterov momentum by replacing $\hat{m}_{k-1}$ with $\hat{m}_k$ and obtain Nadam update rule:

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\sqrt{\hat{v_k}} + \varepsilon} \left(\beta_1 \hat{m}_k + \frac{(1-\beta_1) g_k}{1-\beta_1^k} \right) \tag{4.99}$$

Chapter 5

Drought Prediction

In this section we are going to apply RNN model on time series data, in order to make predictions. Problem which we consider is the prediction of soil moisture content in the first layer (0-7cm) based on meteorological data.

Summer of 2012 in Serbia was the warmest summer with record air temperatures for 19 stations in Serbia and also one of the driest summers with average precipitation amount bellow average for reference period 1961-1990 in almost entire Serbia [23]. In 2012, there was a decrease of maize yield in northern part of Serbia (Vojvodina) for 50%, soya 35% and sugarbeet 30% [3]. Estimated agricultural production loss was approximately USD 2 billion, (maize USD 1 billion,sugar USD 130 million, soya USD 117 million) [5]. There was also an occurrence of aflatoxin in milk and that might be a possible consequence of drought [4]. After 2012 there was extremely warm summer in 2015 with again average precipitation bellow average for reference period 1961-1990 [6]. The summer of 2017 was the second warmest summer on record in Serbia, with record high temperatures in Smederevska Palanka and Banatski Karlovac, dry and very dry conditions across most of Serbia, the fourth driest for Novi Sad and fifth driest for Zrenjanin [7].

5.1 Data

We used meteorological data from ERA5 reanalysis datasets [24] for Serbia, for all 28 stations from official observing network (See Table 5.1). ERA5 is a climate reanalysis service, covering the period from 1979 to present. ERA5 is being developed through the Copernicus Climate Change Service (C3S).

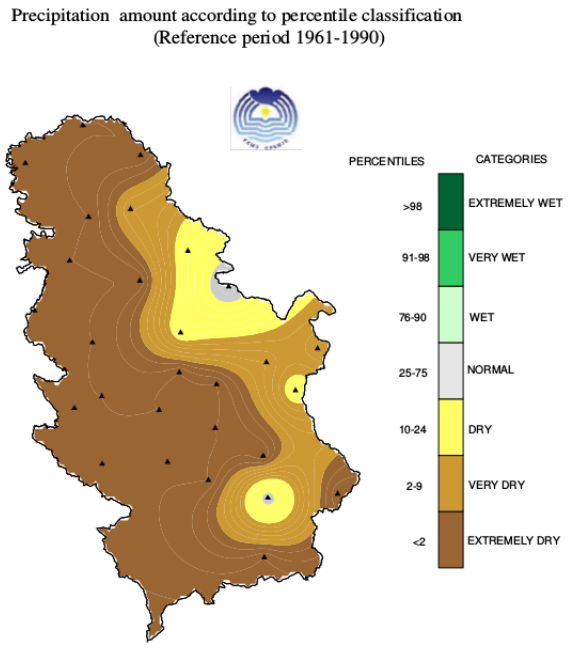


Figure 5.1: Summer 2012. [23]

Copernicus Climate Change Service provides authoritative information about the past, present and future climate. From the Climate Data Store, using API with the Python, we can access to ERA5 reanalysis (ECMWF) of the global climate. Reanalysis combines model data with observations obtained from sources across the world into globally complete and consistent dataset using the laws of physics. Data assimilation is done with horizontal resolution of $0.25° \times 0.25°$ (approximately 25 km) to get hourly data. Temporal coverage is from 1979 to present (with the delay of more than one month). There are many meteorological parameters at surface level, such as: 2m temperature, 2m dewpoint temperature, surface pressure, total precipitation, soil temperature and moisture in 4 layers, surface solar radiation downwards, cloud cover, 10m u-component of wind, 10m v-component of wind etc [25].

For downloading hourly data for Serbia we have used Python package 'cdsapi' and obtain data in NetCDF format, therefore we did data processing in order to make daily data for each of 28 stations in Serbia in period 2011-

2018.

In Table 5.1 we have represented meteorological stations in Serbia with their latitude (°), longitude (°), altitude (m) and soil type where 1 represent coarse, 2 medium, 3 medium fine and 4 fine soil, based on soil texture.

5.2 Experiments

Let's define our problem, we have time series data of meteorological parameters and we would like to make prediction of soil moisture content in the first layer (0 - 7 cm, the surface is at 0 cm). Therefore we have regression problem:

$$y = \varphi(X) \tag{5.1}$$

where independent variable X represents meteorological parameters and dependent variable y is soil moisture content. First, we need to specify which exactly meteorological parameters we are using, as we said in section Data there are a lot of parameters at surface level but since we want to predict soil moisture content we are planing to use the following one:

- Surface solar radiation downwards $(\frac{MJ}{m^2 day})$
- Daily 2m temperature (min and max) (°C)
- Precipitation (mm)
- Vapor pressure deficit $(mbar)$

These parameters define our dependent variable X and soil moisture content $(m^3 m^{-3})$[1], defines independent variable y. Since we have data for X and y we can start with finding function φ, model which can help us to find that nonlinear mapping is RNN, concretely we will use LSTM as a very powerful neural network. The reason why we choose RNN is because we have standard regression problem and our data are time series or in other words sequence data. We will consider the "many to many" sequence problem where for input we use parameters' values from past days and also meteorological parameters for future days, our output is soil moisture content for future days (See Figure 5.2).

[1]- $m^3 m^{-3}$ represents volumetric water content in m^3 per m^3 soil

Name	**Latitude**	**Longitude**	**Altitude**	**Soil**
Palić	46.10	19.77	102	1
Sombor	45.77	19.15	88	2
Novi Sad	45.33	19.85	84	1
Zrenjanin	45.37	20.42	80	2
Kikinda	45.85	20.47	81	3
Banatski Karlovac	45.05	21.03	100	3
Loznica	44.55	19.23	121	2
Sremska Mitrovica	45.10	19.55	82	2
Valjevo	44.32	19.92	176	2
Beograd	44.80	20.47	132	2
Kragujevac	44.03	20.93	185	2
Smederevska Palanka	44.37	20.95	121	2
Veliko Gradište	44.75	21.52	82	2
Crni Vrh	44.12	21.95	1037	4
Negotin	44.23	22.55	42	4
Zlatibor	43.73	19.72	1028	2
Sjenica	43.28	20.00	1038	2
Požega	43.85	20.03	310	2
Kraljevo	43.70	20.70	215	2
Kopaonik	43.28	20.80	1710	2
Kuršumlija	43.13	21.27	384	4
Kruševac	43.57	21.35	166	2
Ćuprija	43.93	21.38	123	2
Niš	43.33	21.90	202	2
Leskovac	42.98	21.95	230	2
Zaječar	43.88	22.30	144	4
Dimitrovgrad	43.02	22.75	450	2
Vranje	42.55	21.92	432	2

Table 5.1: Meteorological stations in Serbia

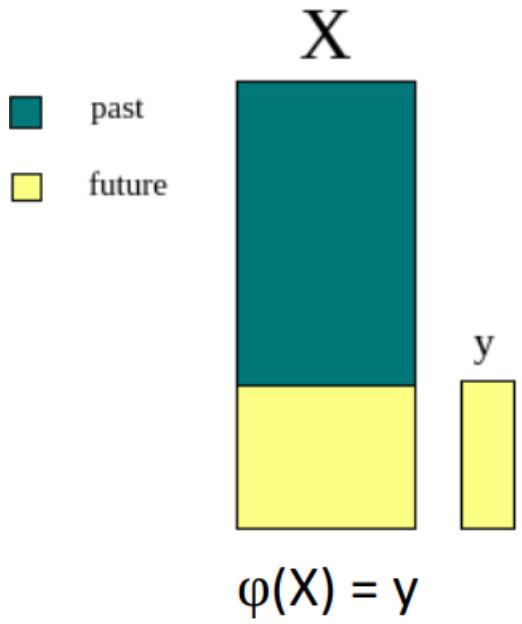


Figure 5.2: Input and output

Dimension of X is $m \times 5$ where each column represents one of the meteorological parameters and m is number of days such that $m = n + p$ where n is the number of future days and p is the number of past days, therefore dimension of y is $n \times 1$. The choice of m, n and p is for discussion but let's say that we want to predict soil moisture for 10 future days ($n = 10$) since for that period we can have a reliable weather forecast, for the number of past days (p) we will use $p = 10, 20, 50, 80$ and compare all methods.

Before training LSTM model we did data standardization and normalization for target value such that volumetric soil content is in range $[0.15,\ 0.45]$ since we have different ranges for different meteorological stations. Splitting data on training and test data is done in following way, data used for training is in period 2011-2016 and data for test is from 2017 and 2018 from all stations. Data from 2017 and 2018 are good choice for test because summer of 2017 was dry and warm compared to the summer of 2018 which was rainy and humid. So the 25% of data is used for test and 75% is used for training.

The neural network model which we use is stacked LSTM with 4 layers and 40 neurons in each layer followed by dense layer with 10 neurons. For

activation function we have used hyperbolic tangent since the range of that function is $[-1, 1]$. The number of layers and neurons is determined through experiments. We also train network using different optimization algorithms and compare results. During training period we have used dropout regularization, dropout means that in each iteration we have randomly deactivated some fraction of neurons, that helped us to avoid overfitting or in other words to maintain similar convergence of both training and test error through time.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 60, 40)	7360
lstm_2 (LSTM)	(None, 60, 40)	12960
lstm_3 (LSTM)	(None, 60, 40)	12960
lstm_4 (LSTM)	(None, 40)	12960
dense_1 (Dense)	(None, 10)	410

Total params: 46,650
Trainable params: 46,650
Non-trainable params: 0

Figure 5.3: Stacked LSTM network with p = 50

In the Figure 5.4 we have training and validation error through time (200 epochs) with dropout and adam optimizer, for loss function we have used mean absolute error function.

5.3 Results

In this section we represent the results what we obtain using stacked LSTM network and compare for different parameters such as number of past days (p) and for different optimization algorithms. We also represent visualization of predicted and true values for all 28 stations in Serbia and compare that time series using statistical tests.

In the table 5.2 we have represented mean absolute error on test data for different choice of past days (p), the number of future days is always 10 (n) and we calculate accuracy using mean absolute percentage error. As we can

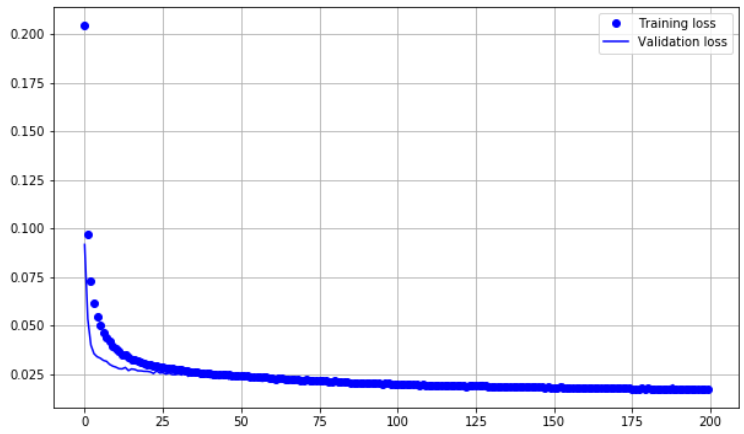


Figure 5.4: Loss functions after 200 epochs for stacked LSTM with p = 50

p	n	MAE	Accuracy
10	10	0.021	92.6%
20	10	0.019	93.2%
50	10	0.017	93.9%
80	10	0.017	93.8%

Table 5.2: Mean absolute error and mean absolute percentage accuracy for different p on test data

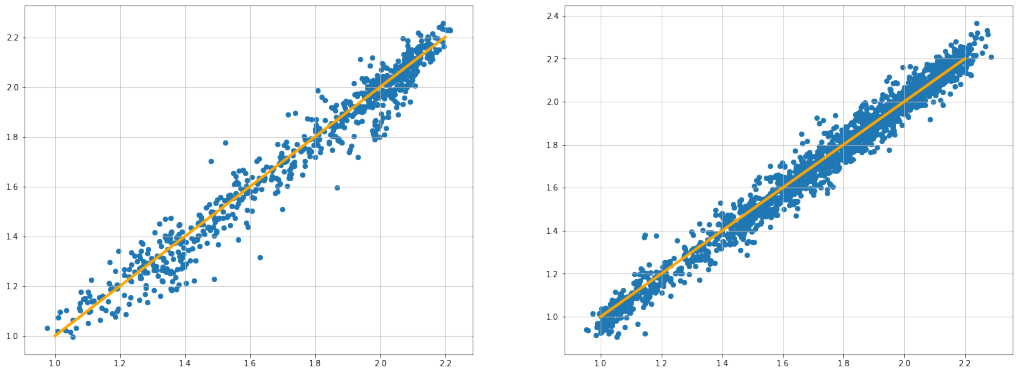
see if we go further in the past we obtain a better accuracy, but the price is the model with more parameters which is taking more time for training.

We also provide mean absolute error and accuracy in table 5.3 for different optimization algorithms on test data which we discuss in section 4.3. We can see that this results confirm theory, SGD gives the good accuracy but with momentum we obtain a slightly better result. RMSprop and Nadam reached the smallest mean absolute error, Also Nesterov, as a better version of momentum, gives the better results than standard momentum.

In the figure 5.5 we represented scatter plot between predicted and true values on test and train data. On each day we have a vector which represents values from all 28 stations, we took a 2 norm of that vector both for predicted (710 points) and true data (2150 points) and then obtained scatter plot.

Optimizer	MAE	Accuracy
SGD	0.032	88.8%
Momentum	0.027	90.2%
Nesterov	0.026	90.5%
Adagrad	0.021	92.3%
Adadelta	0.022	91.9%
RMSprop	0.020	93.2%
Adam	0.021	92.5%
AdaMax	0.021	92.5%
Nadam	0.019	93.1%

Table 5.3: Mean absolute error and mean absolute percentage accuracy for different optimizers on test data



(a) Test data (b) Train data

Figure 5.5: Scatter plot true vs predicted data

Statistic	Train data	Test data
Pearson	0.99	0.98
Kendaltau	0.90	0.86
R2	0.96	0.94
d2	0.99	0.98

Table 5.4: Statistical tests

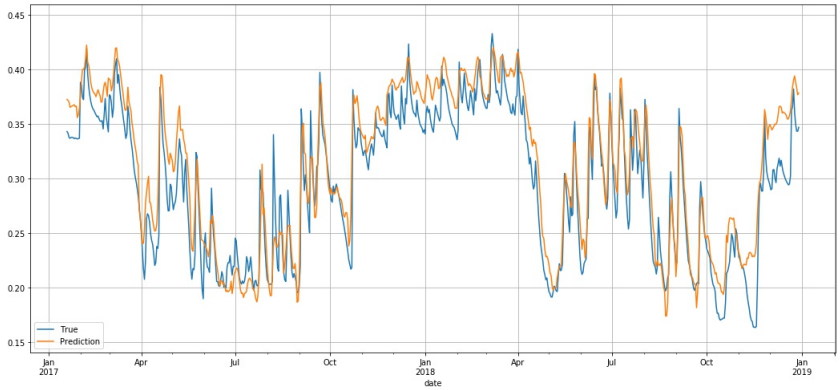


Figure 5.6: Palic

In the table 5.4 we present Pearson, Kendaltau, R2 and d2 statistical tests for predicted and true time series, where for obtaining points of time series we have used the transformation described for scatter plot.

Figures 5.6 - 5.33 include visualizations of real time series of soil moisture content and predicted time series with predicted window of 10 days for period of 2 years (2017 and 2018, test data). This is visualization for stacked LSTM model with 50 past days and adam optimizer.

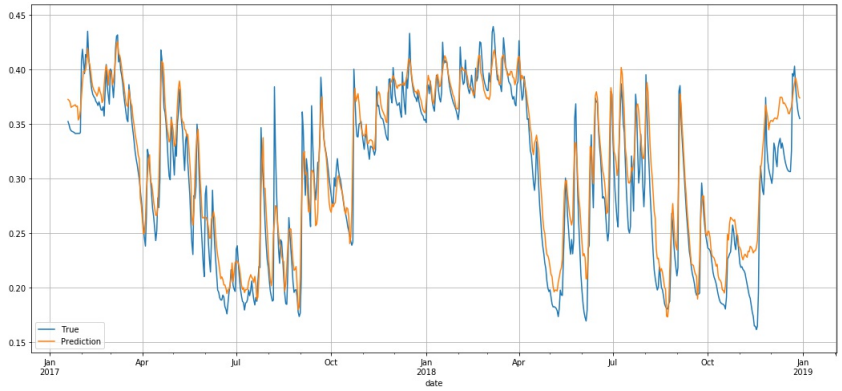


Figure 5.7: Sombor

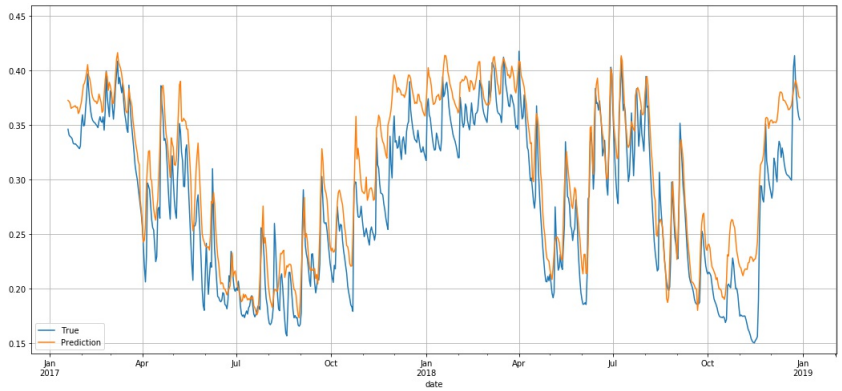


Figure 5.8: Novi Sad

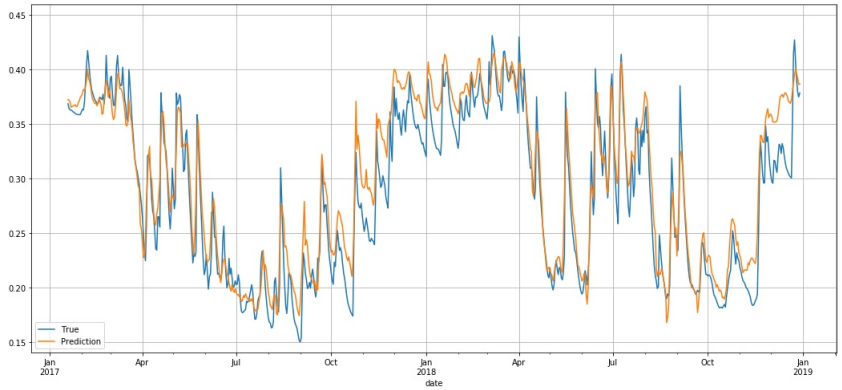


Figure 5.9: Zrenjanin

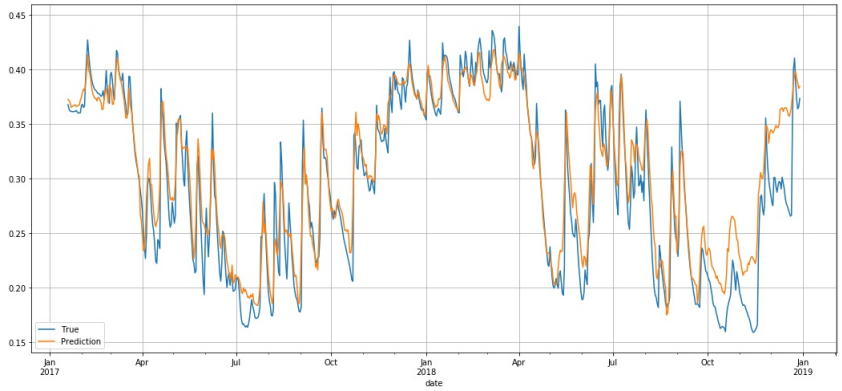


Figure 5.10: Kikinda

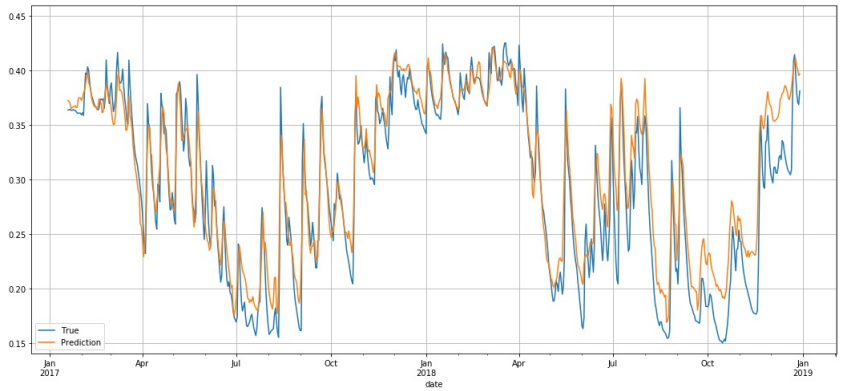


Figure 5.11: Banatski Karlovac

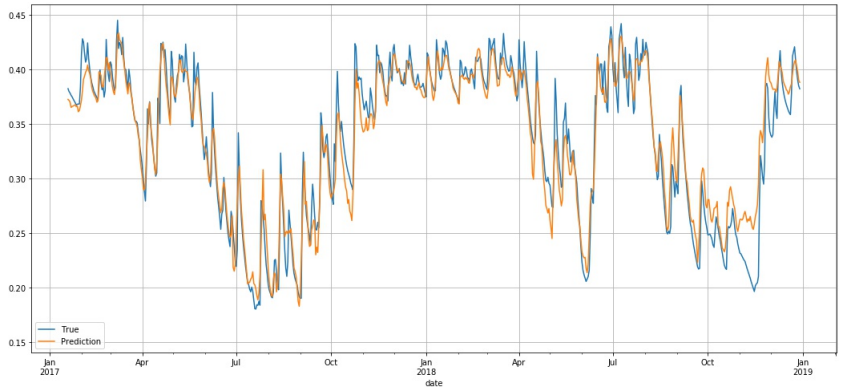


Figure 5.12: Loznica

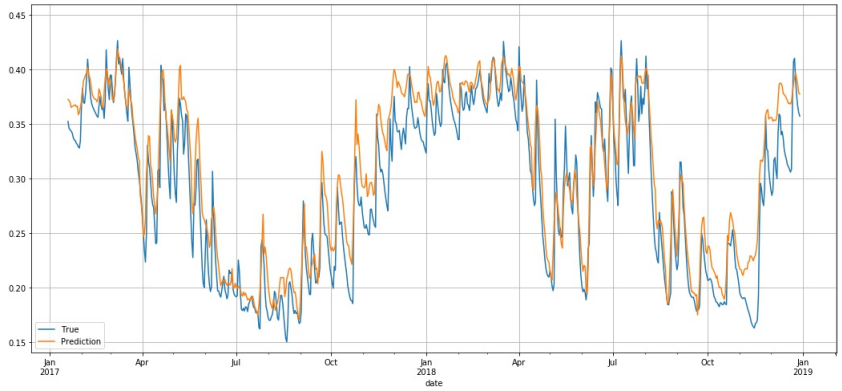


Figure 5.13: Sremska Mitrovica

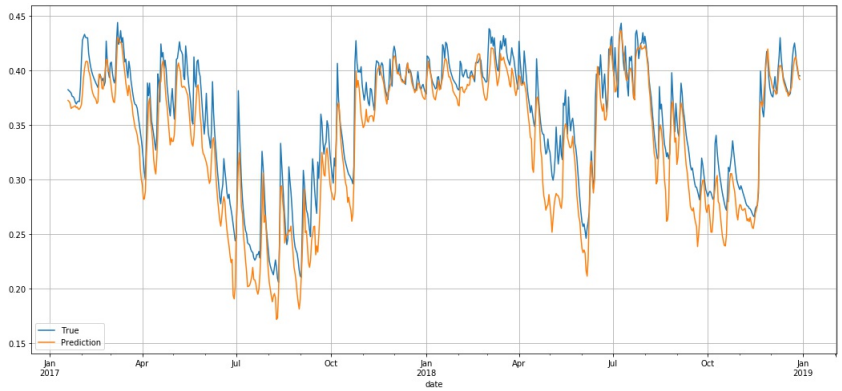


Figure 5.14: Valjevo

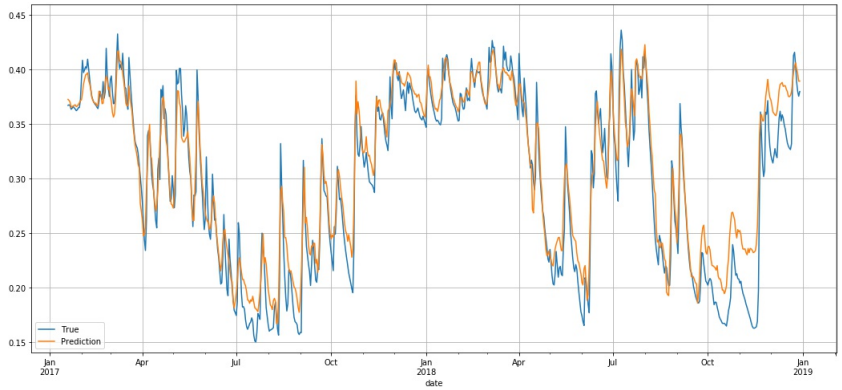


Figure 5.15: Beograd

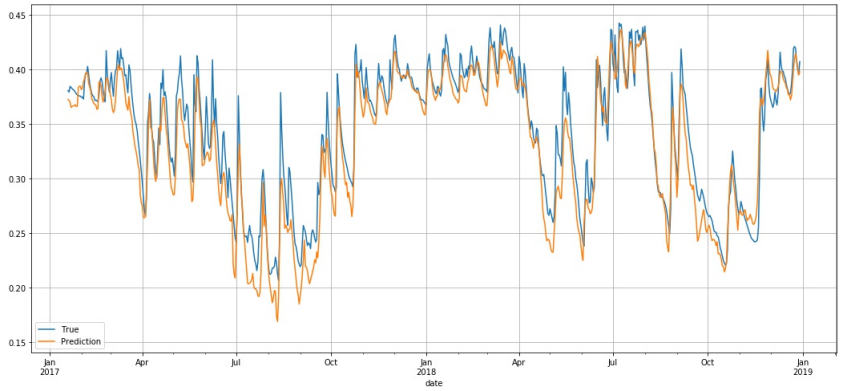


Figure 5.16: Kragujevac

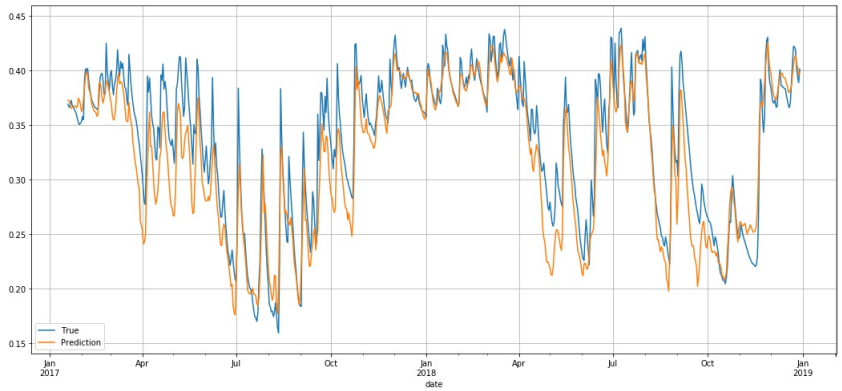


Figure 5.17: Smederevska Palanka

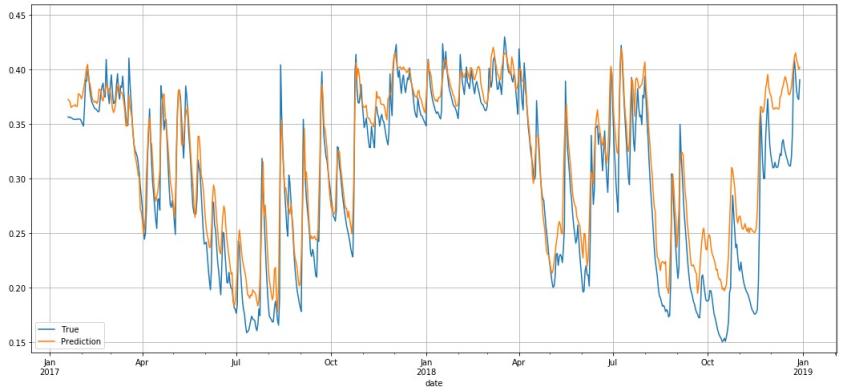


Figure 5.18: Veliko Gradiste

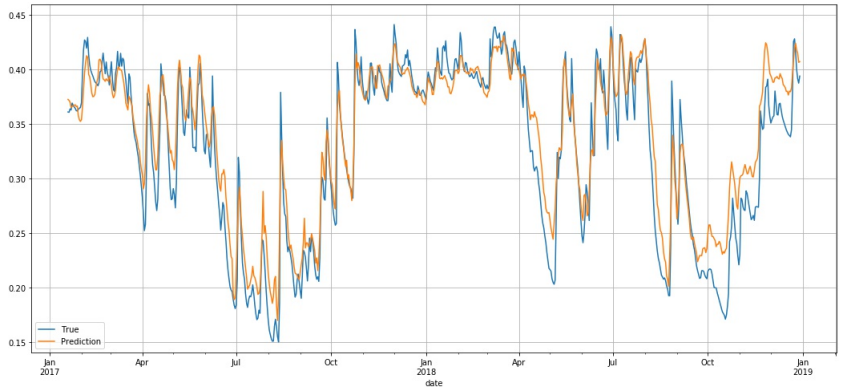


Figure 5.19: Crni Vrh

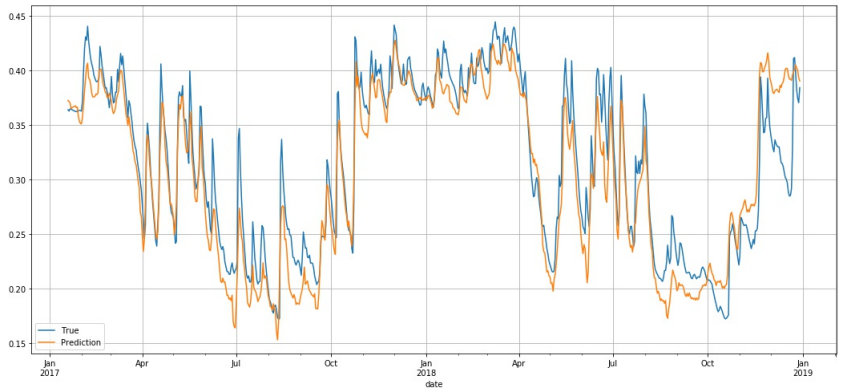


Figure 5.20: Negotin

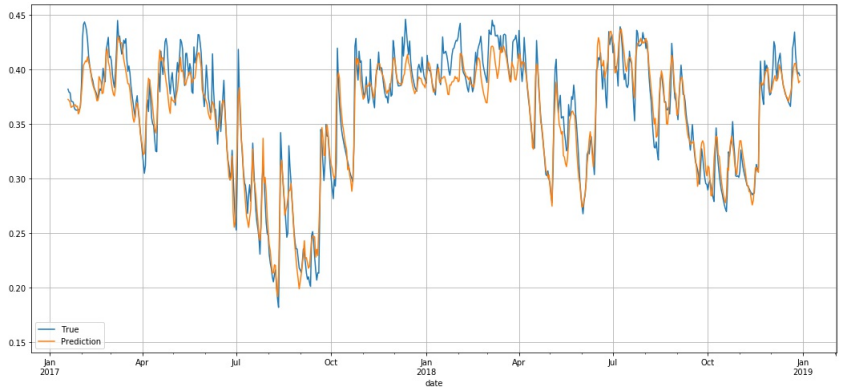


Figure 5.21: Zlatibor

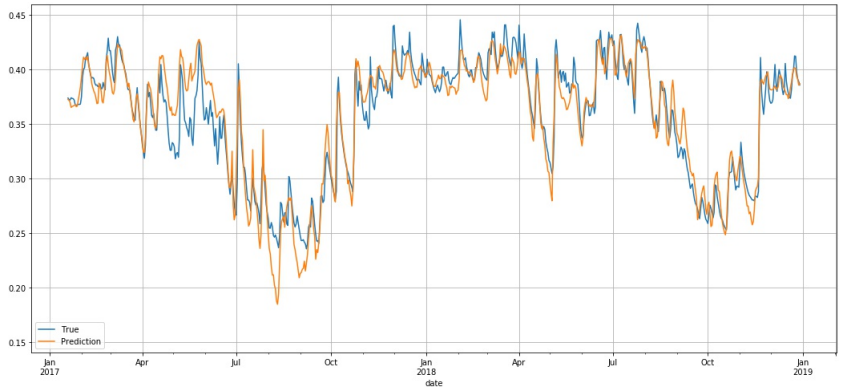


Figure 5.22: Sjenica

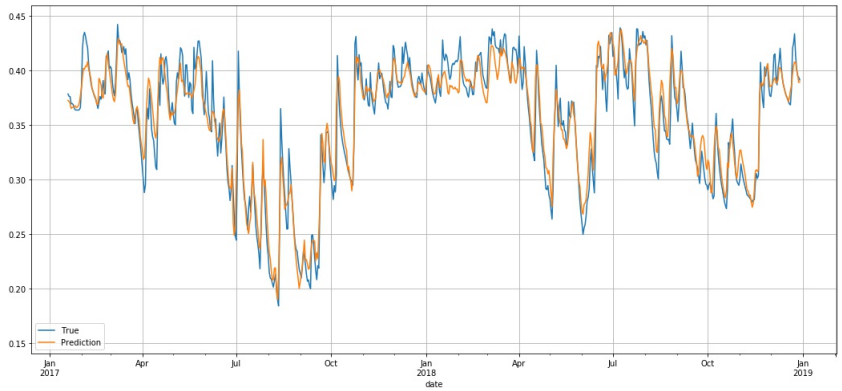


Figure 5.23: Pozega

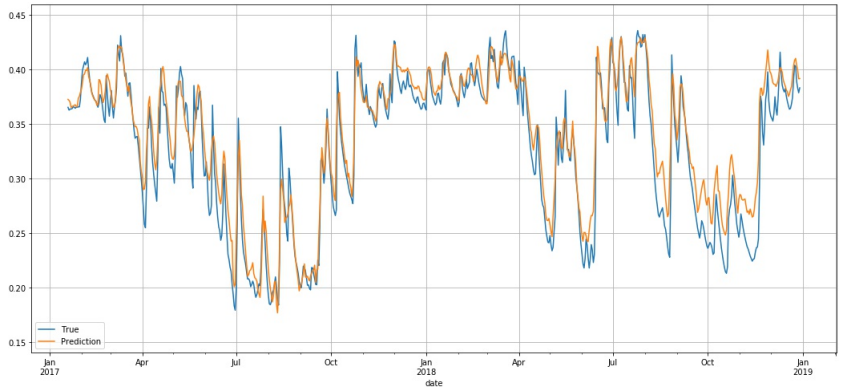


Figure 5.24: Kraljevo

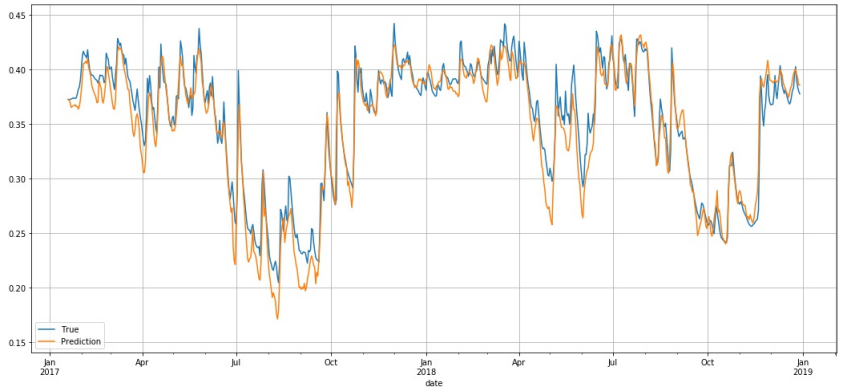


Figure 5.25: Kopaonik

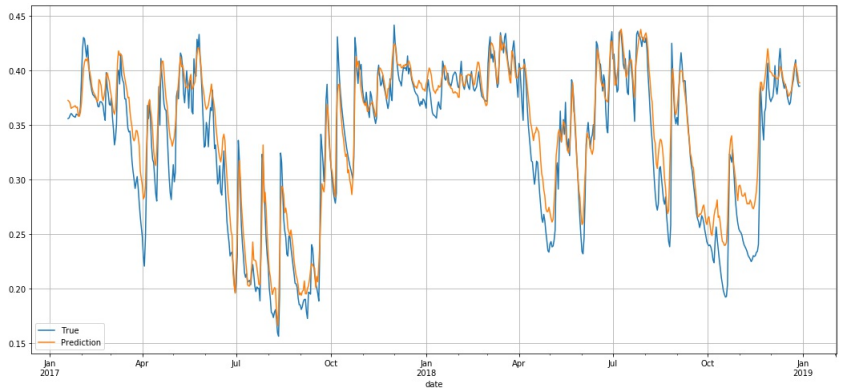


Figure 5.26: Kursumlija

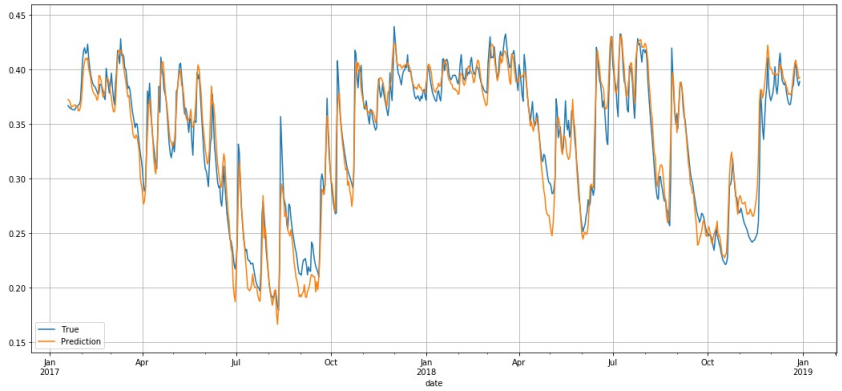


Figure 5.27: Krusevac

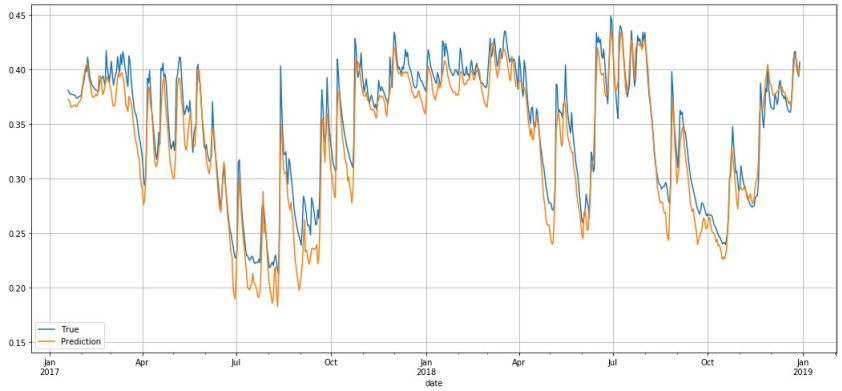


Figure 5.28: Cuprija

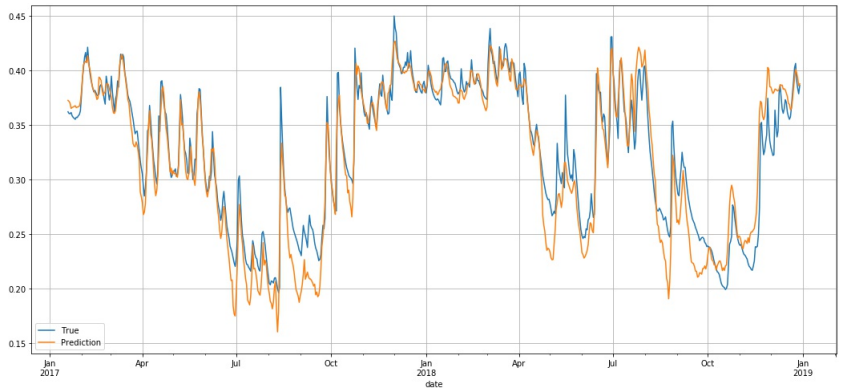


Figure 5.29: Nis

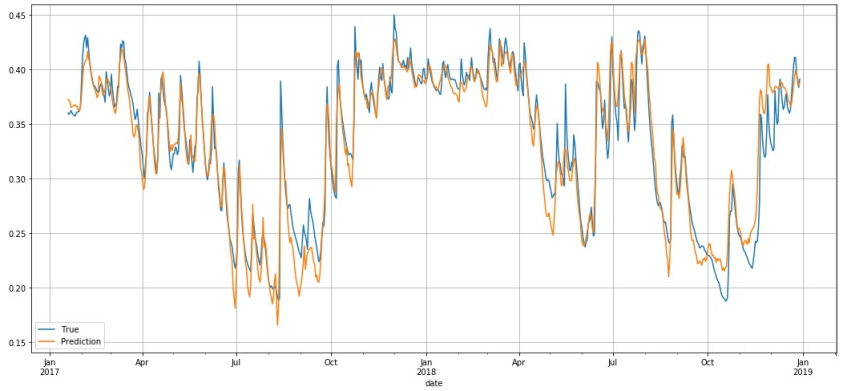


Figure 5.30: Leskovac

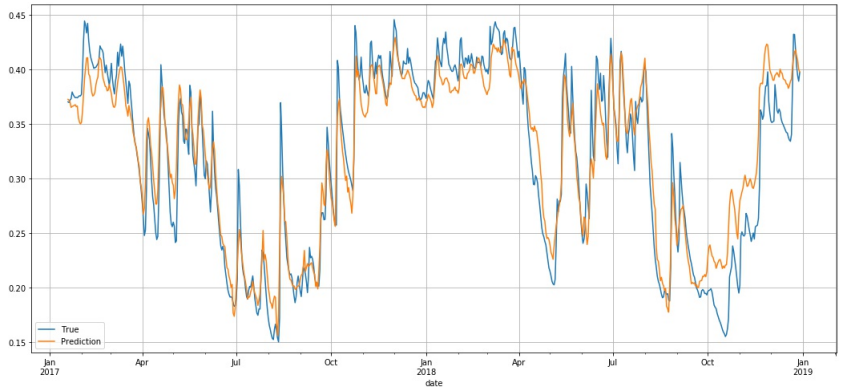


Figure 5.31: Zajecar

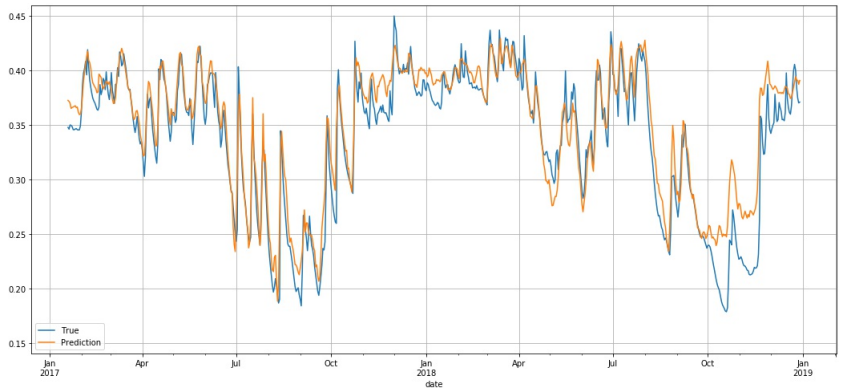


Figure 5.32: Dimitrovgrad

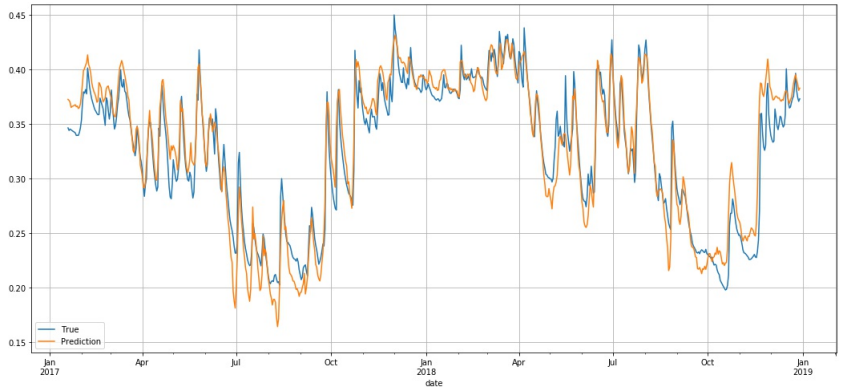


Figure 5.33: Vranje

Chapter 6

Conclusion

The more powerful computers today allow us to develop and use neural networks which are showed as a very powerful tool capable to catch non linear relationship between variables far better then human beings.

In this thesis we considered problem of drought prediction and for that purpose we tried to predict soil moisture content in the first layer which is the most dependent on meteorological situation. Since considered problem belongs to the regression problem and has time series data, the natural choice was the RNN because of it architecture. We chose LSTM network as a very advanced RNN and widely used in practice for time series forecasting, in order to solve that regression problem. Our stacked LSTM model with 4 layers demonstrated very good performance with accuracy 93.9% on the test data. Through experiments we determined that past 50 days period is the best choice with respect to accuracy and time to train network. We made prediction up to 10 days in the future, since that is a period for which we can expect a reliable weather forecast, also it might be possible that for shorter future period we can have a similar or even better results. Our model did not use information about soil type and this should be included into the future work as it could further improve obtained results.

We had also evaluated the influence of the most popular optimization algorithms for neural networks with regard to decreasing validation error. Obtained results align with theory as it turned out that RMSprop and Nadam were the best optimization algorithms for this problem.

For the further work on the proposed model, it might be useful to optimize irrigation process like in [26], and thus minimize quantity of water which is used for irrigation, but more importantly we can also minimize energy needed

to run the irrigation system.

Bibliography

[1] American Meteorological Society. "Heatwaves, Droughts and Floods among Recent Weather Extremes Linked to Climate Change". In: *American Meteorological Society* (2018).

[2] Donald A. Wilhite and Michael H. Glantz. "Understanding the Drought Phenomenon: The Role of Definitions". In: *Water International* (1985).

[3] Károly Fiala et al. "Drought Severity an its Effect on Agricultural Production in the Hungarian-Serbian Cross-Border Area". In: *Journal of Environmental Geography* (2015).

[4] Jovana Kos et al. "Occurrence and estimation of aflatoxin M1 exposure in milk in Serbia". In: *Food Control* (2013).

[5] Food and Agricultural Organization of the United Nations. "Drought risk management guidelines Western Balkan region". In: *Food and Agricultural Organization of the United Nations* (2018).

[6] Republic Hydrometeorological Service of Serbia. "Seasonal Bulletin for Serbia Summer 2015". In: *Republic Hydrometeorological Service of Serbia* (2015).

[7] Republic Hydrometeorological Service of Serbia. "Seasonal Bulletin for Serbia Summer 2017". In: *Republic Hydrometeorological Service of Serbia* (2017).

[8] United Nations Serbia. "Serbia Floods 2014". In: *United Nations Serbia* (2014).

[9] *The differences between Artificial and Biological Neural Networks*. URL: `https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7`.

[10] Vladimir Crnojevic. *Prepoznavanje oblika za inzenjere (Serbian), Pattern Recognition for engineers*. University of Novi Sad, Faculty of Technical Sciences, 2014.

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

[12] *A Simple Multilayer Perceptron with TensorFlow*. URL: https://medium.com/pankajmathur/a-simple-multilayer-perceptron-with-tensorflow-3effe7bf3466.

[13] Andriy Burkov's. *The Hundred Page Machine Learning Book*. 2019.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[15] Gang Chen. "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation". In: (2016). https://arxiv.org/abs/1610.02583.

[16] *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[17] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[18] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[19] Sebastian Ruder. "An Overview of Gradient Descent Optimization Algorithms". In: (2016). https://arxiv.org/abs/1609.04747.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning*. Cambridge University Press, 2014.

[21] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: (2012). URL: https://arxiv.org/abs/1212.5701.

[22] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: (2014). URL: https://arxiv.org/abs/1412.6980.

[23] Jasminka Smailagic et al. "Climatological Analysis of Summer 2012 for Serbia". In: *Republic Hydrometeorological Service of Serbia Department of National Center for Climate Changes* (2012).

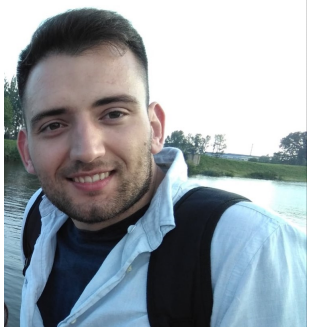
[24] *ERA5 hourly data on single levels from 1979 to present*. URL: https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=overview.

[25] *Climate Reanalysis*. URL: https://climate.copernicus.eu/climate-reanalysis.

[26] Olutobi Adeyemi et al. "Dynamic Neural Network Modelling of Soil Moisture Content for Predictive Irrigation Scheduling". In: *Sensors (Basel)* (2018).

[27] *Classification and Loss Evaluation - Softmax and Cross Entropy Loss*. URL: https://deepnotes.io/softmax-crossentropy#derivative-of-cross-entropy-loss-with-softmax.

Biography



Nemanja Filipović was born on 21th of November 1995 in Šabac. In 2014 he started his bachelor studies of Applied Mathematics at Faculty of Sciences, University of Novi Sad and finished in 2017 with GPA 8.85, same year he continued with master studies of Data Science at the same faculty and passed all exams in 2019 with GPA 9.71. He attended ECMI Mathematical Modelling Week in summer of 2018 where he was included on project "Modelling Human Interactions across a City with Graphs". During his studies he has been financially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia. Nemanja currently works as Junior Researcher at BioSense Insitute.

Appendix A

Neural Networks

A.1 RNN

A.1.1 Activation and Loss Functions

Hyperbolic tangent is defined as $tanh(x) : \mathbb{R} \to [-1, 1]$

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{A.1}$$

and softmax function $softmax(\mathbf{x}) : \mathbb{R}^n \to [0, 1]^n$ as follows:

$$softmax(\mathbf{x}) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{A.2}$$

Cross entropy function between two discrete probability distributions $\mathbf{x}$ and $\mathbf{y}$:

$$H(\mathbf{x}, \mathbf{y}) = -\sum_{k} y_k \log z_k \tag{A.3}$$

A.1.2 Derivative of Softmax and Cross Entropy Function

First, we are going to calculate derivative of softmax activation function. [27]

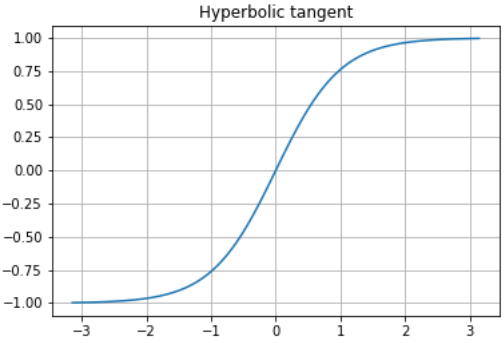


Figure A.1: Activation function

$$z_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \tag{A.4}$$

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}}{\partial x_j} \tag{A.5}$$

if $i = j$

$$= \frac{e^{x_i} \sum_{k=1}^{n} e^{x_k} - e^{x_i} e^{x_j}}{\left(\sum_{k=1}^{n} e^{x_k}\right)^2} = \frac{e^{x_i}\left(\sum_{k=1}^{n} e^{x_k} - e^{x_j}\right)}{\left(\sum_{k=1}^{n} e^{x_k}\right)^2} \tag{A.6}$$

$$= \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \frac{\sum_{k=1}^{n} e^{x_k} - e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} = z_i(1 - z_j) \tag{A.7}$$

for $i \neq j$

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}}{\partial x_j} = \frac{0 - e^{x_i} e^{x_j}}{\left(\sum_{k=1}^{n} e^{x_k}\right)^2} = -\frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} = -z_i z_j \tag{A.8}$$

$$\frac{\partial z_i}{\partial x_j} = \begin{cases} z_i(1-z_j), & i=j \\ -z_i z_j, & i \neq j \end{cases} \tag{A.9}$$

Using Kronecker delta function $\delta_{ij} := \begin{cases} 1, & i=j \\ 0, & i \neq j \end{cases}$ we can derive derivative of softmax activation function as:

$$\frac{\partial z_i}{\partial x_j} = z_i(\delta_{ij} - z_j) \tag{A.10}$$

Now, we are going to calculate derivative of cross entropy loss function

$$H(\mathbf{y}, \mathbf{z}) = -\sum_k y_k \log z_k \tag{A.11}$$

In our case $H(\mathbf{y}, \mathbf{z})$ is loss at time t, therefore denote $L_t = H(\mathbf{y}, \mathbf{z})$. Now we can calculate derivative of L_t with respect to α_t and $z_t = softmax(\alpha_t)$.

$$\frac{\partial L_t}{\partial \alpha_t} = -\sum_k y_k \frac{\partial \log z_k}{\partial z_k} \frac{\partial z_k}{\partial \alpha_t} = -\sum_k y_k \frac{1}{z_k} \frac{\partial z_k}{\partial \alpha_t} \tag{A.12}$$

now we use what is derivative of softmax activation function

$$= -y_t \frac{1}{z_t} z_t(1-z_t) - \sum_{k \neq t} y_k \frac{1}{z_k}(-z_k z_t) = -y_t(1-z_t) + \sum_{k \neq t} y_k z_t \tag{A.13}$$

$$= -y_t + y_t z_t + \sum_{k \neq t} y_k z_t = z_t \sum_k y_k - y_t \tag{A.14}$$

sum of all entries of vector $\mathbf{y}$ is one, therefore:

$$\frac{\partial L_t}{\partial \alpha_t} = z_t - y_t \tag{A.15}$$

A.1.3 Derivative of Hyperbolic Tangent Function

$$\frac{\partial tanh(x)}{\partial x} = \frac{\partial \frac{sinh(x)}{cosh(x)}}{\partial x} = \frac{\frac{\partial sinh(x)}{\partial x} cosh(x) - sinh(x) \frac{\partial cosh(x)}{\partial x}}{(cosh(x))^2} \tag{A.16}$$

$$= \frac{(cosh(x))^2 - (sinh(x))^2}{(cosh(x))^2} = 1 - (tanh(x))^2 \tag{A.17}$$