VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

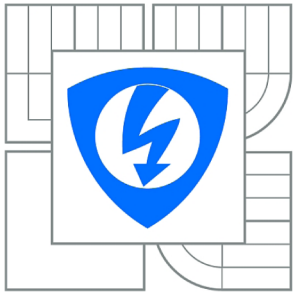
VYUŽITÍ UMĚLÉ INTELIGENCE V KRYPTOGRAFII

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. VOJTĚCH LAVICKÝ
AUTHOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VYUŽITÍ UMĚLÉ INTELIGENCE V KRYPTOGRAFII

THE USE OF ARTIFICIAL INTELLIGENCE IN CRYPTOGRAPHY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

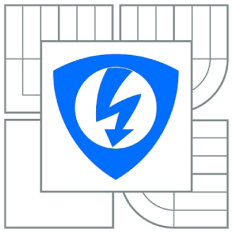
AUTOR PRÁCE — Bc. VOJTĚCH LAVICKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. PATRIK BABNIČ

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Vojtěch Lavický	***ID:***	73017
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Využití umělé inteligence v kryptografii

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou běžně používaných bezpečnostních protokolů v kryptografii. Popište neuronové sítě využitelné v kryptografii. Vytipujte nejvhodnější neuronove sítě, svůj výběr vhodně zdůvodněte. Následně vytvořte koncept nového bezpečnostního protokolu, který využívá vytipovanou neuronovou síť.

DOPORUČENÁ LITERATURA:

[1] MANDIC, DANILO P. Recurrent neural network for prediction, leasing algorithms, architectures and stability. John Wiley, Chichester 2001.
[2] BURDA, K. Bezpečný přenos dat. Bezpečný přenos dat.
Bezpečný přenos dat. Brno: VUT v Brně, 2008. s. 1-77.
[3] ŠNOREK M.,JIRINA M. Neuronové sítě a neuropočítače, CVUT, Praha 1996.
[4] SHIHAB, Khalil. A backpropagation neural network for computer network security. 2006, Journal of Computer Scien-ce 2, 710–715.

Termín zadání:	6.2.2012	***Termín odevzdání:***	24.5.2012

Vedoucí práce: Ing. Patrik Babnič
Konzultanti diplomové práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem práce je se seznámit se s problematikou neuronových sítí a používaných bezpečnostních protokolů v kryptografii. Teoretická práce se zabývá rozborem neuronových sítí s přihlédnutím na výběr typu sítě později využitý v modelu kryptografického systému. V praktické části je vytvořen koncept zcela nového bezpečnostního protokolu, který využívá vytipovanou neuronovou síť.

KLÍČOVÁ SLOVA

neuron, perceptron, neuronová síť, Back-propagation, kryptografie, MATLAB, SIMULINK

ABSTRACT

Goal of this thesis is to get familiar with problematics of neural networks and commonly used security protocols in cryptography. Theoretical part of the thesis is about neural networks theory and chooses best suitable type of neural network to use in cryptographic model. In practical part, a new type of security protocol is created, using chosen neural network.

KEYWORDS

neuron, perceptron, neural network, Back-propagation, cryptography, MATLAB, SIMULINK

LAVICKÝ, Vojtěch *Využití umělé inteligence v kryptografii*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 72 s. Vedoucí práce byl Ing. Patrik Babnič

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Využití umělé inteligence v kryptografii“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Patriku Babničovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Problém jak utajit informace existuje již od doby, kdy se informace začali přenášet. S postupným vývojem technologií a systémů které informace spravují se zvětšuje i problém utajit informace před nechtěným únikem. Masivní rozvoj komunikačních sítí v běžném životě zároveň zvětšuje škodu, kterou může únik informací způsobit. Organizace zvětšující svoji závislost na přenášení informací pomocí počítačových sítí se stávají mnohem více zranitelné na různé druhy útoků. Pokroky v oboru neuronových sítí mohou poskytnout efektivní řešení a obranu proti těmto útokům.

Teoretická část této práce se v první své části zabývá historií výzkumu neuronových sítí a jejími osobnostmi. Je zde popsáno fungování lidského neuronu, snahy o jeho napodobení a vznik formálního neuronu. Dále je popsán základní stavební prvek neuronových sítí - perceptron. Je zde vysvětlen jeho učící algoritmus, stejně tak jako jeho spojování v sítě perceptronů. Podrobně diskutován je učící algoritmus Back-propagation, vybraný k sestavení modelu v praktické části práce a jsou popsány jeho kroky učení a algoritmy.

V praktické části práce jsou nastíněny základy kryptografie a stupně zabezpečení kryptosystémů. Tato část je dále zaměřena na aplikaci neuronové sítě jako koncového šifrátoru a tím vytvoření zcela nového bezpečnostního protokolu založeném na prvcích umělé inteligence. Je popsáno vytvoření neuronové sítě v programu MATLAB, její trénování a simulace. Praktická část se dále zabývá vytvořením funkčního modelu neuronu v prostředí SIMULINK. Tento model je později využit k vytvoření vlastního šifrátoru a dešifrátoru. Poslední část je věnována teoretickému rozboru slabin nově vzniklého protokolu založeného na prvcích umělé inteligence a možnostmi jeho napadení vedoucí k jeho prolomení za účelem získat tajná data.

1 ÚVOD DO NEURONOVÝCH SÍTÍ

1.1 Historie

Základní buňkou nervové soustavy je neuron. Jeho první matematický model vytvořili pánové Warren McCulloch a Walter Pitts v roce 1943 a jejich práce se tímto dá považovat za počátek vzniku oboru neuronových sítí. V tomto modelu byly využity převážně bipolární číselné hodnoty parametrů, tedy množina {-1,0,1}. Bylo dokázáno, že nejjednodušší typy neuronových sítí mohou v principu počítat libovolnou aritmetickou nebo logickou funkci.

V tomto období, přesněji řečeno v roce 1949 vyslovil pan Hebb pravidlo: „Váhové hodnoty na spojení mezi dvěma neurony, které jsou zároveň ve stavu zapnuto budou narůstat a naopak váhové hodnoty na spojení mezi neurony, které jsou zároveň ve stavu vypnuto se budou zmenšovat.“[1]

V roce 1957 pan Frank Rossenblat vynalezl Perceptron. V algoritmu který pro tento model navrhl dokázal, že nezávisle na jeho počátečním nastavení dokáže tento algoritmus při existenci váhového vektoru parametrů nalézt pro daná tréninková data odpovídající váhový vektor po konečném počtu kroků. Nedlouho po objevu perceptronu byl vyvinut další typ výpočetního prvku, který byl nazván ADALINE a to Bernardem Widrowem a jeho studenty [1][2].

Poslední epizodou tohoto období bylo napsání knihy s názvem Perceptrons v roce 1969 pány Marvinem Minskym a Seymourem Papertem. V této knize byl vysloven triviální fakt, že jednovrstvé sítě nedokáží řešit funkci XOR. Vícevrstvé sítě sice tuto funkci vyřešit dokážou, avšak v této době nebyl znám žádný učící algoritmus a pánové Minsky a Papert usoudili, že takový algoritmus díky své složitosti nejspíš ani nebude možné vytvořit [1].

Tímto se zastavilo veškeré financování oboru neuronových sítí. Výzkum se opět rozběhl až v roce 1974 kdy byl objeven algoritmus Back-propagation pro více vrstvé sítě panem Werbosem. Algoritmus Back-propagation je v současné době nejpoužívanější učící algoritmus pro vícevrstvé sítě s dopředným šířením [2].

1.2 Biologie

V průběhu vývoje mnohabuňkových živočichů se hlavně vyvíjely jejich řídící centra od velice jednoduchých po více složité. Vyvinuly se dva systémy: pomalý směrový chemický a rychlý směrový neuronový systém.

V prvním systému jsou do krve uvolňovány hormony a ty jsou následně roznášeny do celého těla, kde na ně reagují jednotlivé buňky. Pokud buňky cév tyto cévy smrští,

vzroste krevní tlak, do krve je vyloučen adrenalin a reakcí buněk srdce je zvýšení počtu tepů.

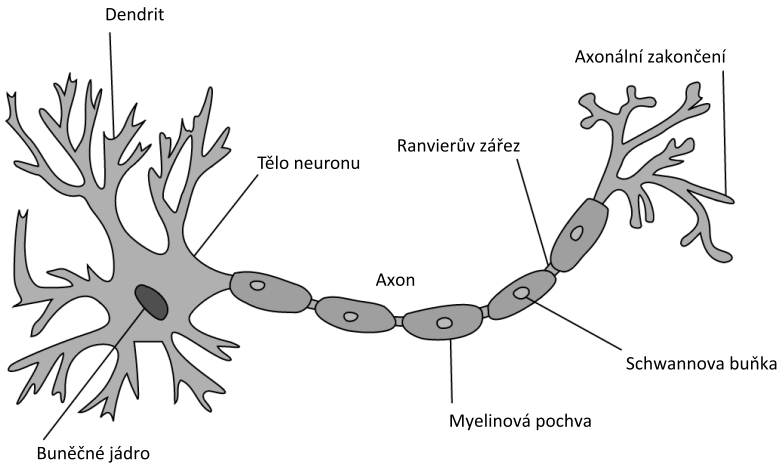
Naproti tomu u rychlého směrového systému buňky reagují na vnější podráždění (například tlakem). Dojde-li k podráždění takové buňky, tato posílá dlouhým vláknem vzruch do svalové buňky a ta se následně smršťí. Protože tato soustava se skládá s osamocených neuronových buněk, nazývá se rozptýlená.

Dalším vývojem živočichů se tyto systémy dále vyvíjely a byly mnohem složitější. Vznikla například žebříková nervová soustava, dále neuronové uzly. Následovala nervová trubice, která se přetransformovala na mozek a míchu [4].

2 NEURONY

2.1 Biologický neuron

Neuron se dá označit jako základní prvek nervové soustavy. Je to buňka, která je používána pro přenos zpracovaní, uchování a vyhodnocení informací. Při vývoji modelu neuronu byla zpočátku snaha pochopit a modelovat způsob jakým myslíme a kterým funguje náš mozek. Nejde o přesnou kopii lidského mozku, ale je jen žádoucí napodobit jeho základní funkce [1]. Struktura neuronu je znázorněna na obr.2.1.[1]



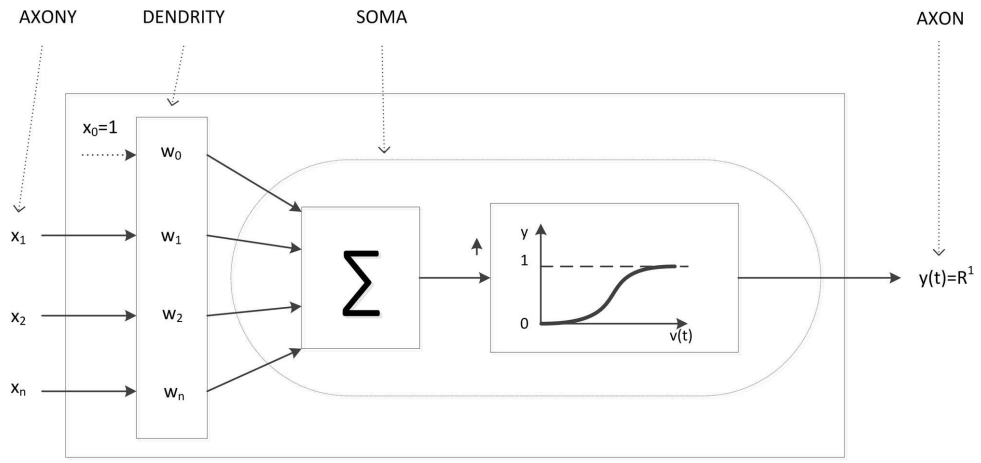
Obr. 2.1: Neuron

Neuron se kromě základní části tzv. Somatu skládá i ze vstupní a výstupní části, resp. dendritu a axonu. Soma je obalena tenkou membránou a vyplněna plazmou. Jak je patrno z obrázku 2.1, z axonu vyrůstá řada výběžků, které jsou zakončeny blanou a navazuji na dendrity ostatních neuronů. Naopak dendrity, které můžeme považovat za vstupy, vyrůstají ze samotného těla buňky. Ke vlastnímu spojení dvou neuronů a tedy k přenosu informace, slouží speciální mezineuronové rozhraní - synapse [1].

Synapse můžeme z funkčního hlediska rozdělit na excitační a inhibiční, tedy jestli šíří vzruch nebo způsobují útlum v nervové soustavě. Synapse mohou za určitých situací generovat elektrické impulzy, které se šíří pomocí axonu synaptickými branami na dendrity jiných neuronů. Další neurony mají určitou mez propustnosti a tím určují intenzitu podráždění dalších neuronů. Při podráždění neuronu dostatečně velkým signálem, větším než jeho hraniční mez, tzv. práh, tento neuron sám generuje

[1]Autor obrázku 2.1 je Quasar Jarosz, licence GNU

impulz a zajišťuje tak šíření dané informace. Synaptická propustnost se může měnit po každém průchodu signálu, není tedy konstantní, ale adaptivní a přizpůsobuje se různým situacím. Tento proces změny synaptických vah se dá přirovnat k vývoji organismu během jeho života a jeho učení se [1]. Z těchto vlastností vzniká model umělého neuronu, ovšem neznamená to, že jsou zcela totožné. Některé modely se od jejich biologických předloh v principu činnosti dost odlišují. Obrázek 2.2 převzat z literatury [4].



Obr. 2.2: Model umělého neuronu

2.2 Formální neuron

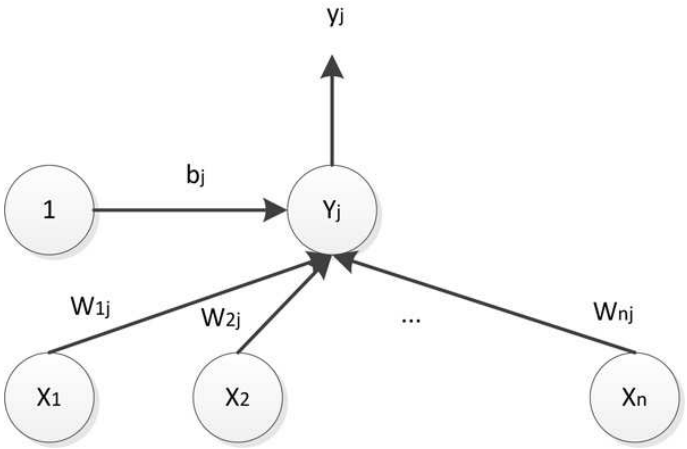
Základní jednotkou neuronových sítí je matematický model neuronu. Mezi nejznámější modely patří McCulloch-Pitsův model neuronu, taktéž nazývaný formální neuron. Jeho struktura je zobrazena na obr. 2.3, převzato z literatury [3]. Modely neuronů se mohou lišit jak matematickou funkcí, tak topologií kterou používají.

- X - vstupy formálního neuron, modelují dendrity, obecně je jich n,
- B - bias neuronu, někdy označován jako koeficient učení.
- W - propustnost, modeluje synaptickou váhu, mohou být I záporné.
- Y - výstupní hodnota neuronu.

Výstupní hodnota neuronu se dá vyjádřit jako vážená suma vstupních hodnot [1].

$$y_j = \sum_{i=1}^{n} w_{ij} x_i \tag{2.1}$$

Kde $f(y_j) = \left\{ \frac{1 jestliže \xi \geq 0}{0 jestliže \xi < 0} \right\}$



Obr. 2.3: Formální neuron

Bias b neuronu, může být do tohoto vztahu začleněn jako jakákoliv jiná váha, jak je uvedeno v následujícím vztahu:

$$y_j = \sum_{i=0}^{n} w_{ij} x_i$$

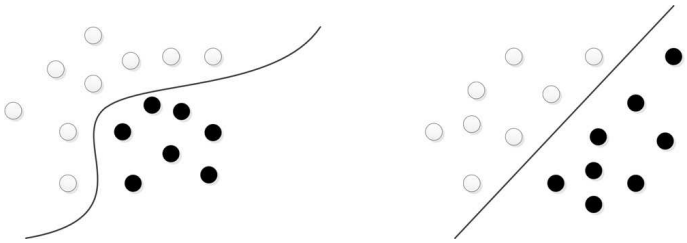
$$= w_{0j} + \sum_{i=1}^{n} w_{ij} x_i$$

$$= b_j + \sum_{i=1}^{n} w_{ij} x_i \tag{2.2}$$

2.3 Perceptron

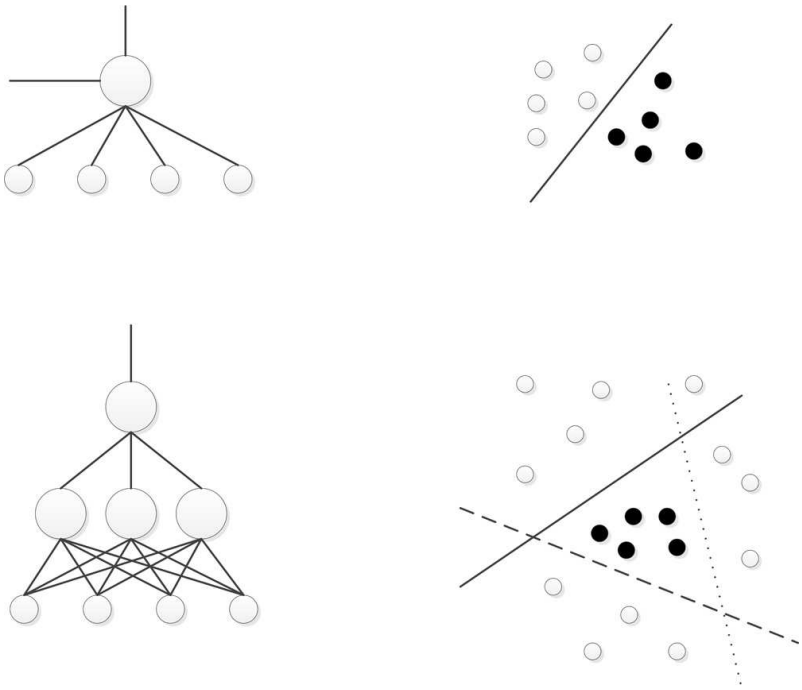
Perceptron sestává většinou z jediného výkonného prvku. Obvykle to bývá McCulloch-Pitsův model neuronu s nastavitelným prahem a váhovými koeficienty. Názvem perceptron může být označen jak jeden prvek, tak i celá síť takových prvků. V roce 1958 Rosenblatt poprvé publikoval algoritmus vhodný k nastavení parametrů perceptronu. O čtyři roky později, tedy v roce 1962 dokázal následující větu: „Máme-li v n-rozměrném prostoru lineárně separabilní třídy objektů, pak lze v konečném počtů kroků učení (iterací optimalizačního algoritmu) nalézt vektor vah W perceptronu, který oddělí jednotlivé třídy bez ohledu na počáteční hodnotu těchto vah.“ [3]

Perceptron s jedním výkonovým prvkem dokáže klasifikovat objekty do maximálně dvou tříd, přidáváním aktivních prvků do topologie se počet tříd do kterých se jím dá klasifikovat zvyšuje. Při větším počtu aktivních prvků než jeden, potom



Obr. 2.4: Separabilnost tříd objektů

tyto třídy nemusí být lineárně separabilní, ale stále musí být separabilní. Na obrázku 2.5 je zobrazena perceptronovská síť, která vychází z fyziologického vzoru [3].

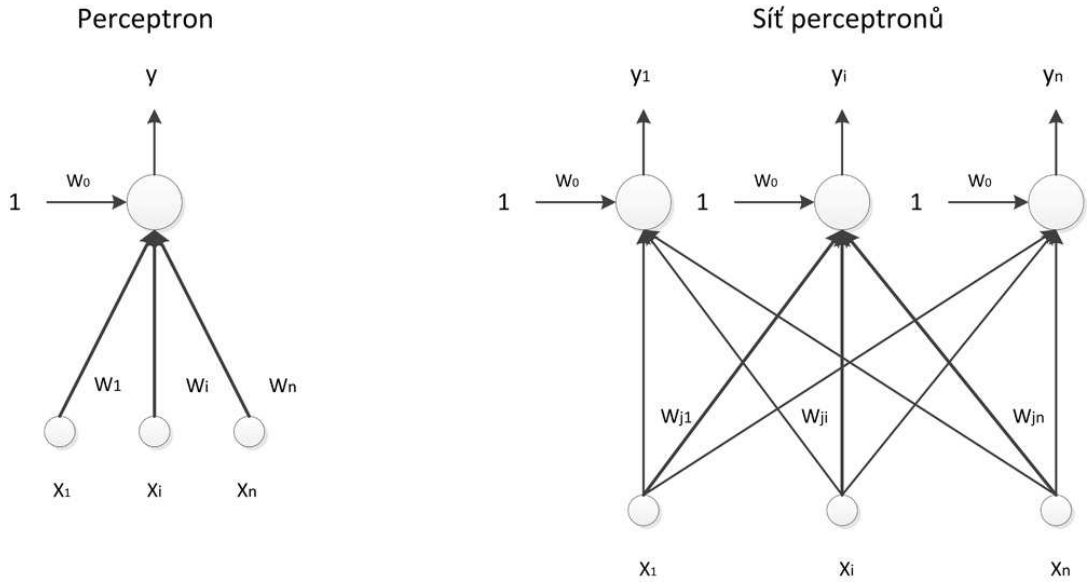


Obr. 2.5: Zvyšování aktivních prvků a jejich vliv na klasifikaci tříd

2.3.1 Topologie a funkce

V topologii uvedené na předchozím obrázku existují celkem tři vrstvy a každá z nich má svůj specifický úkol. Vstupní vrstva perceptronové sítě rozvětvuje (vyrovnává) vstupní vektor (většinou dvouvrstvý) na jednovrstvý. Součástí perceptronu jsou i pevně připojené, tudíž konstantní váhy. Prvky (neurony) ve druhé vrstvě jsou náhodně spojeny s prvky vrstvy druhé, tzv. detektory příznaků. Nejdůležitější v topologii je třetí vrstva, která má za úkol rozpoznat vzory. Obsahuje tzv. rozpoznávače vzorů (pattern recognizer, perceptrons), které naopak od ostatních vrstev nemají

pevně nastavené váhy, tyto se nastavují při procesu učení [3]. Obrázek 2.6 převzat z literatury [4]



Obr. 2.6: Síť perceptronů

K učení takovéto sítě F. Rosenblatt navrhl učící algoritmus se skokovou přenosovou funkcí. Tato funkce je v podstatě transformací vstupního obrazu na výstupní. Funkci můžeme definovat takto:

- $x1, ..., xn \in R^n$ což je skutečná reálná množina proměnných z R^n
- Existuje množina funkcí definovaných na členech této množiny

Jestliže nalezneme množinu takových koeficientů, aby platila rovnice 2.3, je tato rovnice skoková. Rovnice 2.3 je čerpána z literatury [3].

$$\psi(x) = F_n \left(\sum_{i=1}^{M} w_i \phi_i(x) + w_{M+1} \right) \tag{2.3}$$

Tato rovnice je realizována za pomocí modelu neuronu od McCullocha-Pittse s N vstupy a patřičně zvolenými vahami. Neurony ve vrstvě ve které probíhá učení mají zpravidla ještě další vstup, jehož hodnota je nastavena na konstantní hodnotu 1. Jak už bylo řečeno, pro správné naučení musí být předložený vzor (problém) separabilní. Perceptron se jako takový dokáže naučit lineárně separabilní funkci. Pokud má být vyřešen problém který není lineárně separabilní (musí však být separabilní) použije se síť perceptronů. Stejně jak je to zobrazeno na obr. 2.6.

Učící algoritmus perceptronu probíhá po krocích. Jako první se v nultém kroku inicializují váhy w_i ($i = 1$ *až* n) a bias b. Váhy se většinou nastavují blízko nulové

hodnoty. Další krok reprezentuje podmínku a to takovou, že pokud nenastane změna váhových hodnot další kroky se opakují. V kroku 2 probíhá výběr jednotlivých vstupů tréninkové množiny a požadovaných výstupů a provádějí se kroky 3 až 5, kde probíhá aktivace vstupních neuronů, výpočet skutečných hodnot na výstupu a korekce váhových hodnot a biasu pro daný vzor. V posledním kroku 6 se kontroluje jestli nenastává žádná změna váhových hodnot, jinak se pokračuje znovu od kroku 2 [2][3].

Krok 0: Inicializace vah w_i ($i = 1$ *až* n) a biasu b malými náhodnými čísly. Přiřazení inicializační hodnoty koeficientu učení α ($0 < \alpha \leq 1$).

Krok 1: Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 6).

Krok 2: Pro každý tréninkový pár (tj. vstupní vektor a příslušný výstup), provádět kroky (3 až 5).

Krok 3: Aktivace vstupních neuronů.

Krok 4: Výpočet skutečných hodnot na výstupu.

Krok 5: Aktualizace váhových hodnot a biasu pro daný vzor, jestliže:

$Y = 0, t = 1$
w_i(new) = w_i(old) + t $\cdot x_i$ (i = 1 až n).
b(new) = b(old) + t.

Y = 1, t = 0
w_i(new) = w_i(old) - t $\cdot x_i$ (i = 1 až n).
b(new) = b(old) - t.

jinak :
w_i(new) = w_i(old)
b(new) = b(old)

Krok 6: V tomto kroku se nachází podmínka ukončení, jestliže ve druhém kroku již nenastává žádná změna, algoritmus se ukončí, jestli ano, pokračuje se dále.

Lepších výsledků dosáhneme, když do kroku 5 implementujeme tzv. koeficient učení

$\alpha(0 < \alpha \leq 1)$.

Krok 5 pak bude vypadat následovně:

$Y = 0, t = 1$
w_i(new) = w_i(old) + $\alpha \cdot t \cdot x_i$ (i = 1 až n).
b(new) = b(old) + $\alpha \cdot t$.

$Y = 1, t = 0$
w_i(new) = w_i(old) - $\alpha \cdot t \cdot x_i$ (i = 1 až n).
b(new) = b(old) - $\alpha \cdot t$.

jinak:
w_i(new) = w_i(old)
b(new) = b(old)

Aby byly vstupy aktivní musí být jejich hodnota na vstupech nad prahem nenulová. Velikost změny vah závisí na zvolené variantě. Při aktualizaci vah (inkrementaci nebo dekrementaci) se aplikují pevné přírůstky. Konvergenci sítě lze zrychlit změnou velikosti přírůstků v závislosti na velikosti chyby. Toto zrychlení však může mít za následek nestabilitu učení [3].

Matematicky se dá učící algoritmus perceptronu vyjádřit takto:

Perceptron:

$$y_{out} = w^T \cdot x \tag{2.4}$$

Data:

$$(x^1, y_1), (x^2, y_2), \ddots, (x^{N,y_N}) \tag{2.5}$$

Chyba:

$$E(t) = (y(t)_{out} - y_t)^2 = (w(t)^T x^t - y_t)^2 \tag{2.6}$$

Učení:

$$w_i(t+1) = w_i(t) - c \cdot \frac{\partial E(t)}{\partial w_i} = w_i(t) - c \cdot \frac{\partial (w(t)^T x^t - y_t)^2}{\partial w_i} \tag{2.7}$$

$$w_i(t+1) = w_i(t) - c \cdot (w(t)^T x^t - y_t) \cdot x_i^t$$

$$w(t)^T x = \sum_{j=1}^{m} w_j \cdot x_j^i$$

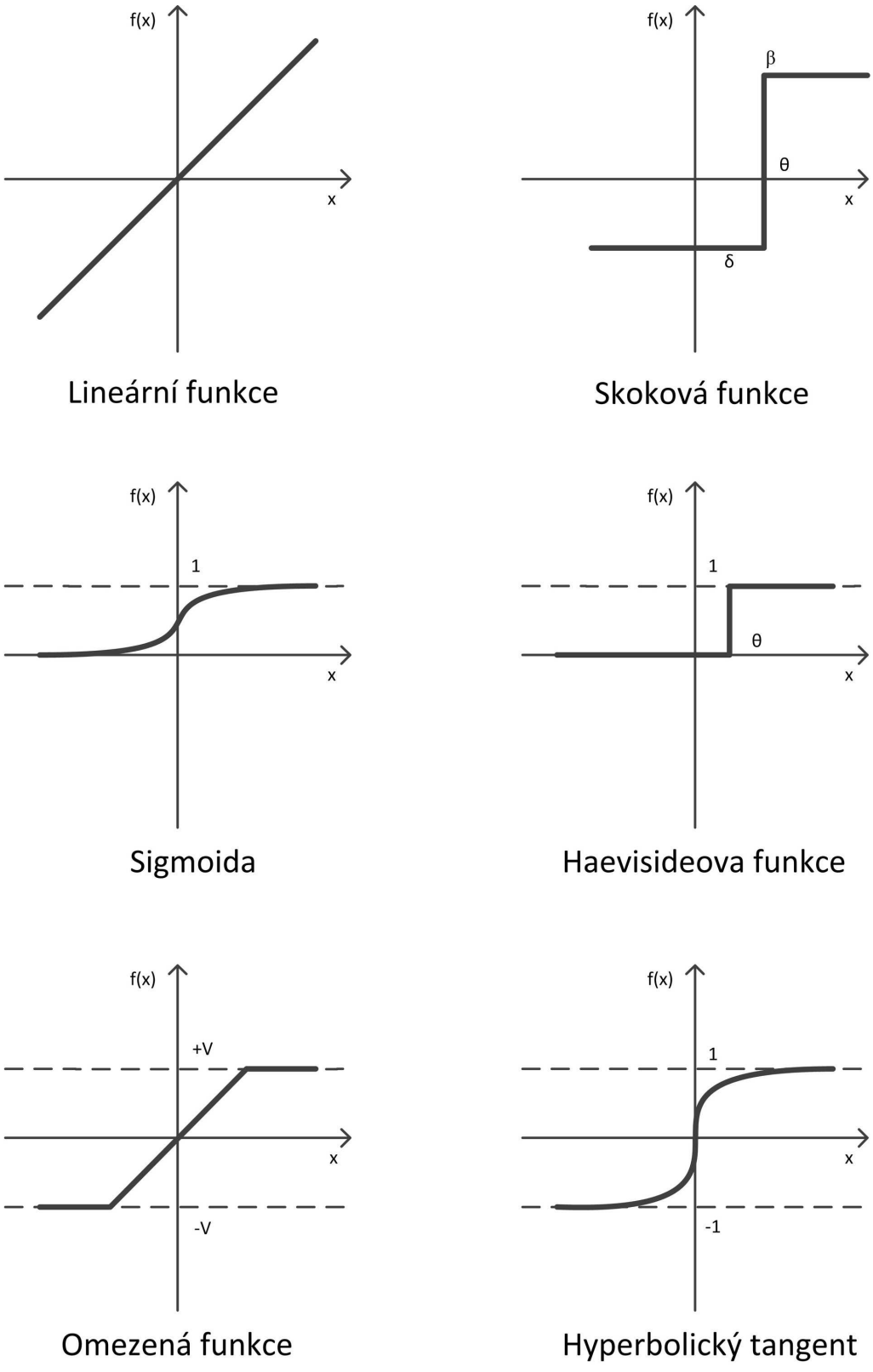
Jak je z matematického zápisu zřejmé, učící algoritmus nám ovlivní chyba $E_{(t)}$, která vychází z nastavení vah na poslední vrstvě perceptronu. Matematické vzorce jsou čerpány z literatury [5].

Z výše uvedených faktů se dá vyvodit následující, perceptron jako takový dokáže realizovat funkci AND a pro realizaci funkce XOR se používá jako síť (spojení perceptronů). Skoková přenosová funkce kterou využíval byla nahrazena (například sigmoida) a rovněž učící algoritmus byl nahrazen lepším a to metodou zpětného šíření chyby (Back-propagation) jak je popsáno v dalších kapitolách [4].

2.3.2 Přenosové funkce

Zatím bylo uvažováno jen o přenosové funkci neuronu danou rovnicí 2.1, ta byla inspirována funkcí biologického neuronu. Další přenosové funkce však vznikly matematickou invencí, nebo jinou, například fyzikální. Běžně se můžeme setkat s následujícími typy přenosových (někdy označovaných jako aktivační) funkcí [1] Obrázek 2.7 převzat z literatury [4].

1. Lineární funkce
2. Skoková funkce
3. Sigmoida
4. Haevisideova funkce
5. Omezená funkce
6. Hyperbolický tangent



Obr. 2.7: Přenosové funkce

3 UMĚLÉ NEURONOVÉ SÍTĚ

Neuronová síť se skládá z formálních neuronů spojených tak, že výstup neuronu je vstupem do obecně více neuronů. Zde je podobnost s biologickými neurony. Síť tvoří paralelní distribuovaný systém výkonových prvků uspořádaných vhodně tak, aby byly schopny zpracovat požadovanou informaci. Zpracování informací a šíření signálu sítí je umožněno změnou stavu neuronů a synaptických vah. Tyto dva parametry známé také jako stav a konfigurace neuronové sítě, spolu s učením, jsou základní a podstatné vlastnosti neuronové sítě [1].

Tzv. paradigma neuronové sítě, určuje celkovou charakteristiku sítě a popisuje následující veličiny:

- Modely neuronů
- Topologie
- Způsob učení

V čase se neuronová síť mění (mění se stav neuronů, adaptují se váhy, atd.), tato změna se dá rozdělit do tří tzv. dynamik [1]:

- Organizační (změna topologie)
- Adaptační (změna stavu, tj. učení a trénování)
- Aktivační (změna konfigurace)

Cílem je neuronovou síť nastavit tak, aby dávala přesné výsledky. Toto „nastavení“ je v biologických neuronech uchovávané v tzv. dendritech. Ekvivalentně se informace v umělých neuronových sítích uchovávají v podobě synaptických vah [1][4].

3.1 Druhy neuronových sítí a topologie

Existuje několik kritérií, podle kterých lze neuronové sítě rozdělit [3].

Podle počtu vrstev:

- Jednovrstvá (Hopfieldova síť, Kohonenova síť, atd.)
- Vícevrstvá (Perceptron, atd.)

Podle algoritmu učení:

- S učitelem
- Bez učitele

Rozdělení podle stylu učení:

- Deterministické
- Stochastické

Deterministický styl učení je zastoupen například v algoritmu Back-propagation kde se váhy nastavují podle chyby sítě. Oproti tomu u stochastického stylu se váhy

nastavují náhodně [3].

Učení s učitelem

Při učení s učitelem se síti předloží tréninkový vzor a stejně jako je tomu u biologických neuronů, použije se zpětná vazba. Síť je nastavena náhodně a aktuální výsledek se porovná s výsledkem požadovaným. Na základě tohoto srovnání se určí chyba. Následně se provede korekce vah či prahů za účelem snížení chyby. Učící proces trvá tak dlouho, dokud není chyba nulová, nebo pod určitým dopředu nastaveným limitem. Po tomto procesu je síť adaptovaná [5].

Učení bez učitele

Při učení bez učitele neznáme výstup sítě a neexistuje zde žádné kritérium správnosti. Ve vstupních datech se hledají vzorky s určitými podobnými vlastnostmi. Tomuto učení se také říká samoorganizace [5].

Aplikace neuronových sítí je velice široká, od zpracování obrazů, přes lékařství, až po vojenství a umělou inteligenci. Jako poměrně nový výpočetní paradigma mají své výhody a nevýhody [3].

Výhody:

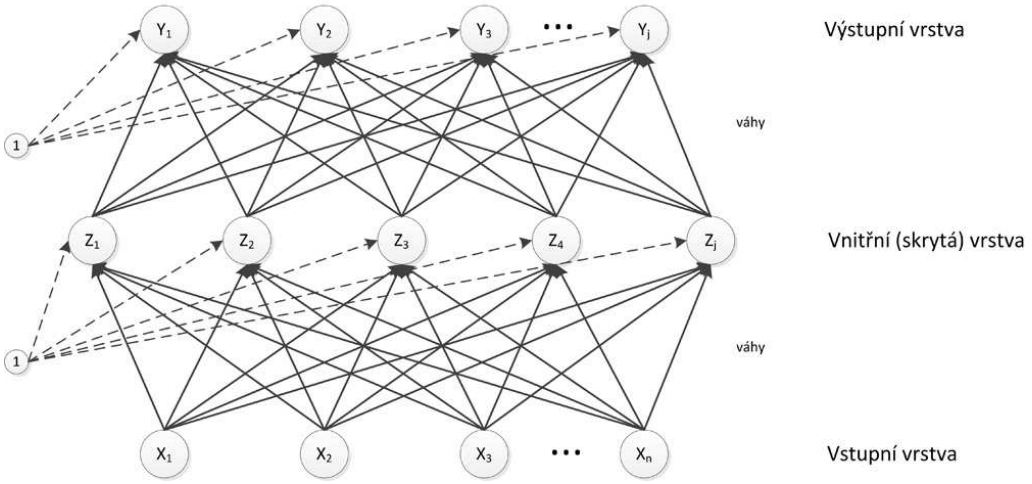
- Učení
- Návrh se může přizpůsobit požadavkům
- Generalizace

Nevýhody:

- Je obtížné odhadnout výkonnost budoucí sítě
- Trénování sítě může být v některých případech velice obtížné eventuálně nemožné
- Neexistují přesná pravidla pro základní návrhy sítě
- Neexistuje obecná přístupová cesta k vnitřním operacím sítě

3.2 Back-propagation

Učící algoritmus Back-propagation (metoda zpětného šíření) je v současnosti jeden z nejužívanějších algoritmů pro trénování a učení neuronových sítí. Díky své jednoduchosti a flexibilitě je používán přibližně v 80% všech aplikací a dokáže vyhovět velice širokému spektru použití. Jedná se o matematický model vícevrstvé neuronové sítě vycházející ze základní neuronové sítě a to perceptronu [3].



Obr. 3.1: Model vícevrstvé neuronové sítě

Na obrázku 3.1 je zobrazena vícevrstvá síť s jednou skrytou (vnitřní) vrstvou. Převzato z literatury [2]. Jak je patrno, vícevrstvá síť je tvořena minimálně třemi vrstvami neuronů: vstupní, výstupní a alespoň jednou vnitřní vrstvou [2].

Spojení mezi sousedními vrstvami je provedeno tak, že každý neuron nižší vrstvy je spojen s každým neuronem vrstvy vyšší.

3.2.1 Algoritmus učení

Algoritmus učení se skládá ze tří etap. V první etapě se šíří sítí vstupní signál tréninkového vzoru (tzv. feedforward), ve druhé probíhá zpětné šíření chyby sítí od vyšších vrstev k nižším a v poslední etapě se aktualizují váhové hodnoty na spojeních mezi neurony. Během adaptace sítě jsou pro každý neuron ve výstupní vrstvě srovnávány jeho výstupní hodnoty s definovanými hodnotami pro dané vstupní vzory.

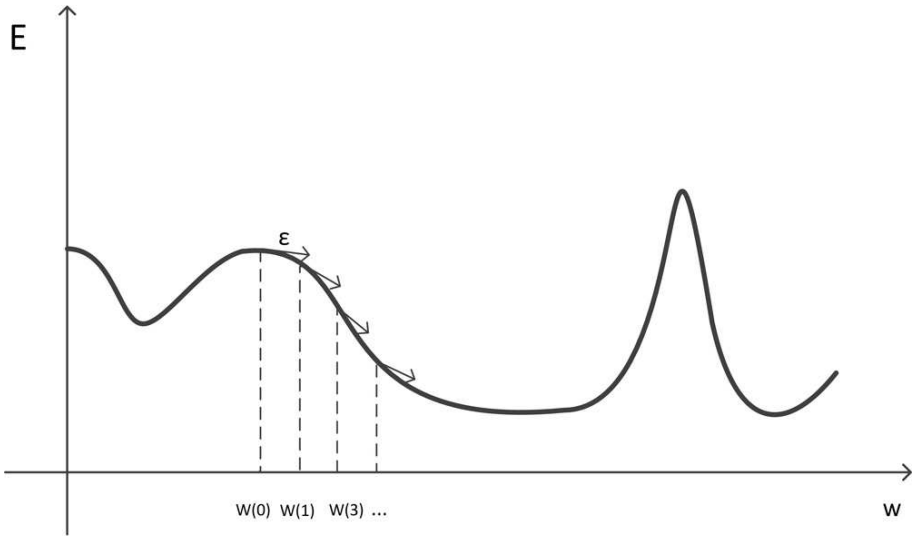
Cílem algoritmu Back-propagation je minimalizace energie neuronové sítě [3]. Pokud bude energie co nejmenší, bude síť podávat nejlepší výsledky v závislosti na tréninkové množině. Energií sítě je myšlena chyba výstupního neuronu (míra naučenosti sítě) pro daný tréninkový vzor. Označuje se jako E a vyjadřuje odchylku mezi skutečnou hodnotou a hodnotou na získanou na výstupu sítě pro určitý vzor [1][3][5].

$$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - d_i)^2 \tag{3.1}$$

Kde:

- n -počet neuronů výstupní sítě,
- y_i - i-tý výstup,
- d_i - i-tý požadovaný výstup.

Pro dané hodnoty na vstupu neuronové sítě, bude síť vykazovat určitou energii. Je třeba docílit zmenšení této energie (chyby) a to změnou jednotlivých váhových koeficientů. K tomuto účelu je možné použít například tzv. Gradientní metodu. Při použití této metody je třeba vypočítat gradient chybové funkce v závislosti na váhových koeficientech. Gradient určí jak správně upravit váhové koeficienty (o kolik a s jakým znaménkem), jestliže bude daná váha upravena o Δw. Gradientní metoda pracuje s energetickou funkcí, ve které považuje vstupy za neměnné a hodnoty prahů a vah jako proměnné. Toto si lze představit jako zakřivenou plochu v hyperprostoru jak ukazuje obrázek 3.2. Bodem na této ploše je pak okamžitá hodnota vah a prahů. Cílem je minimalizace energie, tedy pohyb po ploše podle největšího gradientu [1][3][5]. Obrázek 3.2 převzat z literatury [2].



Obr. 3.2: Proces minimalizace energie neuronové sítě

Vzorec pro chybu výstupní vrstvy je (3.1). Algoritmus Back-propagation šíří tuto výstupní chybu zpět přes všechny skryté vrstvy k vrstvě vstupní a následně se přepočítávají váhové koeficienty [6].

Pro výpočet vah platí následující vzorec [2]:

$$w_{ji}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ji}^{(t)} \tag{3.2}$$

Gradient váhy w_{ji} je spočten jako parciální derivace podle váhy w_{ji} [1]:

$$\Delta w_{ji}^{(t)} = \frac{\partial E}{\partial w_{ji}}(w^{(t-1)}) \tag{3.3}$$

Dále bude vypočítána tzv. chyba j-tého neuronu. Ta je potřeba k tomu, aby bylo možné upravit váhové koeficienty ve skryté vrstvě (nebo vrstvách, záleží na topologii) během zpětného šíření chyby [2]:

$$\frac{\partial E_K}{\partial y_j} = \delta_j \tag{3.4}$$

Jestliže se chyba výstupních neuronů vypočítá jako rozdíl očekávané hodnoty (dle tréninkového vzoru) a skutečným výstupem neuronu, pak pro skrytou vrstvu (vrstvy) se zpětně určí chyba j-tého neuronu jako vážená suma chyb neuronů z vrstvy vyšší [3][5].

Pro úpravu vah platí následující vztah [1]:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \delta_j \frac{\partial f_j(\xi_j)}{\partial \xi_j} y_j \tag{3.5}$$

Kde δ_j je chyba j-tého neuronu a $\frac{\partial f_j(\xi_j)}{\partial \xi_j}$ je parciální derivace obecné přenosové funkce s potenciálem ξ_j podle potenciálu daného neuronu.

Zavádí se koeficient α, který určuje rychlost učení. Váhy se vždy upravují α násobkem gradientu. Tento koeficient v podstatě udává rychlost, jakou se síť učí. Je možné jej v průběhu učení měnit, jestli chyba klesá, koeficient se zvětšuje a naopak [3][5].

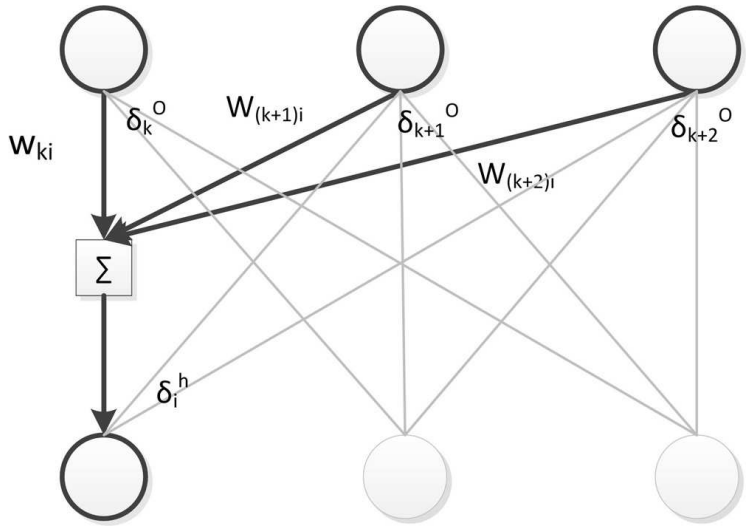
Pokud je koeficient učení α náhodně nastaven, tato metoda vždy konverguje k nějakému lokálnímu minimu. Tento výpočet může být časově velice náročný. Hlavním problémem gradientní metody je, že pokud nalezne určité minimum energie E (nulový gradient), adaptace se v tomto bodě zastaví a ve výpočtu již nepokračuje, tudíž ani chyba se nezmenšuje. Problémem je to především proto, že nalezené lokální minimum nemusí být tzv. globální jak je vidět na obrázku 3.2 [2].

3.2.2 Výpočet parametrů ve skryté vrstvě

Parametry ve skryté vrstvě (váhy, prahy, chyby) se počítají na základě stejných úvah jako parametry výstupní vrstvy. Pro každou váhu ve skryté vrstvě je spočítána chyba dle:

$$\frac{\partial E_K}{\partial y_j} = \delta_j \tag{3.6}$$

Pro přenesení chyby na skrytou (nižší) vrstvu, musí být chyba vyšší vrstvy vynásobena hodnotou příslušného váhového koeficientu. Jelikož je ve výstupní vrstvě většinou více než jeden neuron a neuron v nižší vrstvě je spojen se všemi neurony vyšší vrstvy, je výsledná chyba dána součtem příspěvků od všech neuronů vyšší vrstvy [3]. Toto je znázorněno na obrázku 3.3, lterý je převzat z literatury [3].



Obr. 3.3: Přenesení chyby na nižší vrstvy

Obecně lze toto zapsat:

$$\delta_i^h = y_i^h(1 - y_i^h) \sum_{k=1}^{n} w_k i^o \delta_k^o \tag{3.7}$$

Vzorec (3.7) převzat z literatury [3].

Váhy jsou adaptovány podle vztahu (3.7) pro všechny neurony ve všech vrstvách. Dále následuje předložení dalšího tréninkového vzoru a celý proces se opakuje. Proces předložení všech tréninkových vzorů se nazývá iterace. Většinou bývá během procesu učení vykonáno několik stovek iterací. Učící proces končí buď vykonáním maximálního počtu iterací, nebo dosáhnutí maximální dovolené chyby, kterou může síť dosáhnout [5].

3.2.3 Kroky učení algoritmus Back-propagation

Cílem učícího algoritmu je snižování energie neuronové sítě. Jak již bylo zmíněno algoritmus se může zastavit v tzv. lokálním minimu. Takováto síť není naučena s dostatečně malou chybou. Tomu to jevu lze předejít například [3]:

- Vhodnou volbou parametru alfa
- Vhodným výběrem vzorů pro trénovací množinu
- Vhodným počátečním nastavením vah

Váhové hodnoty se volí dostatečně malé a generují se náhodně. Parametr α je z množiny $\langle 0, 1 \rangle$. Přesné hodnoty parametrů se liší síť od sítě a nejde stanovit přesné universální hodnoty. Zjištění optimálních hodnot je tak otázkou experimentu.

Kroky učení:

Krok 0: inicializace váhových hodnot a biasů malými náhodnými čísly.

Krok 1: Dokud není splněna podmínka ukončení výpočtu, opakovat kroky 2 až 9.

Feedforward

Krok 2: Pro každý tréninkový vzor provádět kroky 3 až 8.

Krok 3: Aktivace vstupní vrstvy.

Krok 4: Výpočet vstupních a výstupních hodnot hodnot vnitřních neuronů.

Krok 5: Stanovení skutečných hodnot na výstupu neuronové sítě.

Back-propagation

Krok 6: Srovnání hodnot výstupní vrstvy s tréninkovým vzorem, je vypočtena δ_k která slouží ke korekci vahových koeficientů a biasu.

Krok 7: Výpočet δ_k pro všechny neurony vnitřní vrstvy.

Korekce váhových koeficientů a biasů

Krok 8: Každý neuron aktualizuje váhové koeficienty na svých spojeních a svůj bias. Toto platí pro výstupní a všechny skryté vrstvy.

Krok 9: Podmínka ukončení. Algoritmus se zastaví pokud již nedochází ke změně vah a nebo pokud již bylo vykonáno maximálně definované množství změn.

4 VYUŽITÍ UMĚLÉ INTELIGENCE V KRYPTOGRAFII

4.1 Úvod do kryptografie

Kryptografie je věda, zabývající se konstrukcí kryptografických ochran, tedy metoda utajení zprávy ve smyslu převádění zprávy do podoby, kdy je čitelná jen se znalostí speciálního druhu (např. vyřešení matematického problému) [7].

Problém jak utajit informace existuje již od doby, kdy se informace začali přenášet. S postupným vývojem technologií a systémů které informace spravují se zvětšuje i problém utajit informace před nechtěným únikem. Masivní rozvoj komunikačních sítí v běžném životě zároveň zvětšuje škodu, kterou může únik informací způsobit. Organizace zvětšující svoji závislost na přenášení informací pomocí počítačových sítí se stávají mnohem více zranitelné na různé druhy útoků. Pokroky v oboru neuronových sítí mohou poskytnout efektivní řešení a obranu proti těmto útokům [8].

Hojné využití kryptografických metod v informačních technologiích je dáno jejich snadnou implementací, možností aplikace a relativně vysokou odolností proti útokům [7].

Základní pojmy

- Zpráva Z - informace kterou chceme přenést po nezabezpečeném kanále, v podobě tzv. otevřeného textu (plain text).
- Klíče K_E a K_D - šifrovací a dešifrovací klíč, mohou být stejné, nebo různé.
- Kryptogram C - zašifrovaný text

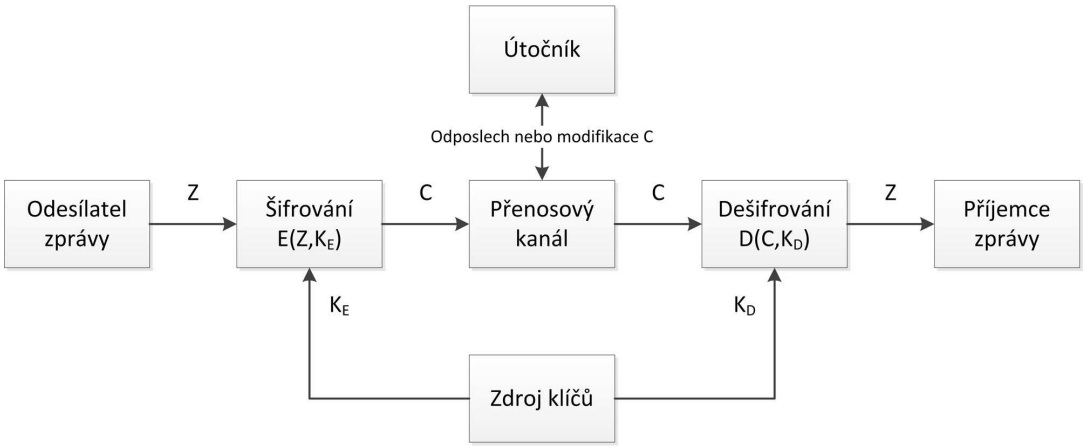
Proces transformace otevřeného textu na zašifrovaný text se nazývá šifrování a inverzní proces je nazýván dešifrování. Při šifrování dochází k prostému zobrazení z množiny Z do množiny C podle parametru K_E [7]:

$$C = E(Z, K_E)$$

Dešifrování potom následovně:

$$Z = D(C, K_D)$$

Komunikace přes přenosový kanál je znázorněna na obrázku 4.1. Obrázek je převzat z literatury [7].

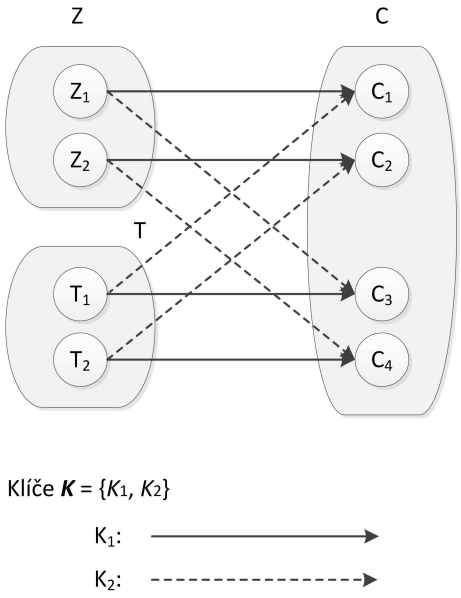


Obr. 4.1: Kryptografický systém

Mezi další a velice důležitý parametr kryptografických systémů patří stupeň utajení vůči kryptoanalýze. Stupně jsou tři [7]:

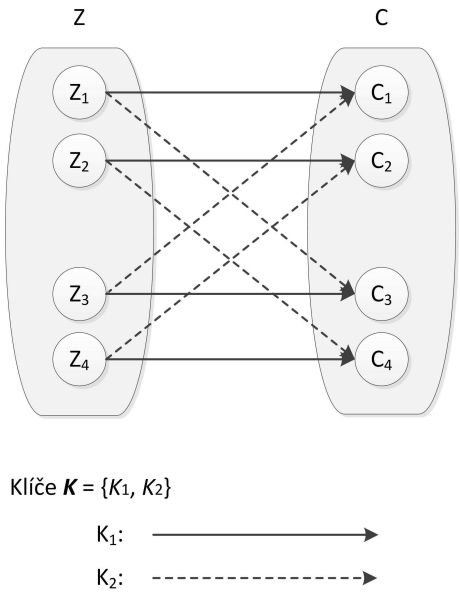
- Dokonalé utajení.
- Ideální utajení.
- Reálné utajení.

Při použití reálného utajení lze kryptogram rozluštit jednoznačně a to i bez použití klíče (např. hrubou silou). Všechny prakticky používané kryptosystémy zaručují pouze reálné utajení. Poskytuje nejnižší míru zabezpečení. Větší bezpečnosti můžeme dosáhnout velkým počtem klíčů [8]. Obrázek převzat z literatury [7].



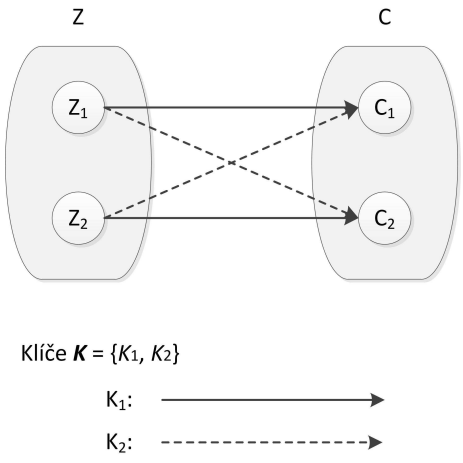
Obr. 4.2: Reálné utajení

Při použití ideálního utajení nelze kryptogram rozluštit jednoznačně. Lze však vyloučit zprávy jejichž zašifrováním daný kryptogram vzniknout nemohl [8]. Obrázek převzat z literatury [7].



Obr. 4.3: Ideální utajení

Při použití dokonalého utajení nelze kryptogram rozluštit, bez znalosti klíče, vůbec. Kryptogram může vzniknout zašifrováním libovolné ze všech možných zpráv [8]. Obrázek převzat z literatury [7].



Obr. 4.4: Dokonalé utajení

4.2 Aplikace neuronové síťě

Jako další, velice vhodnou alternativou k již existujícím kryptosystémům, může být s úspěchem použita neuronová síť. Toto použití se nabízí, neuronová síť v podstatě transformuje vstupní informace na její výstup jako to dělá například bloková šifra.

Vstup do neuronové sítě lze chápat jako zprávy Z, klíč K lze zaměnit s váhovými koeficienty a výstup z neuronové sítě lze označit jako kryptogram. Při použití vhodné topologie, lze použít vhodný typ neuronové sítě jako koncový šifrátor. Volba topologie závisí na dané aplikaci sítě a na dalších faktorech, jako například zvolená abeceda zprávy [6]. Použitím neuronové sítě jako šifrátoru se tak vytvoří zcela nový zabezpečovací protokol.

Aplikací neuronové sítě jako šifrátoru nebo dešifrátoru je vytvořen tzv. kryptografický systém s tajným klíčem (neboli symetrický kryptosystém). To se sebou ovšem přináší určité nevýhody. Šifrovací klíč musí komunikující strany držet v tajnosti. Velkým problémem je také distribuce klíčů od zdroje klíčů k odesílateli a příjemci. Naproti tomu jsou tyto systémy velice rychlé [7].

Použití neuronové sítě nabídne reálnou míru utajení, protože stupeň utajení závisí na počtu zpráv, které vzniknou dešifrováním kryptogramu všemi možnými klíči [7]. Toto vyjadřuje tzv. valence kryptogramu a pro její výpočet platí rovnice:

$$V = \frac{|Z| \cdot |K|}{|C|} \tag{4.1}$$

Kde $|Z|$ je počet všech zpráv, $|K|$ je počet všech možných klíčů a $|C|$ je počet všech možných kryptogramů [8]. Výsledná míra utajení se pak určí:

- Dokonalé utajení, když V = $|Z|$.
- Ideální utajení, když 1< V < $|Z|$.
- Reálné utajení, když V = 1.

Jelikož se jedná o neuronovou síť, váhy sítě tvoří v podstatě jeden klíč. Počet zpráv (vstupů sítě) je stejný jako počet získaných kryptogramů (výstupů sítě). Výsledná valence kryptogramu V se tedy rovná jedné - reálné utajení.

4.2.1 Neuronová síť jako šifrátor a dešifrátor

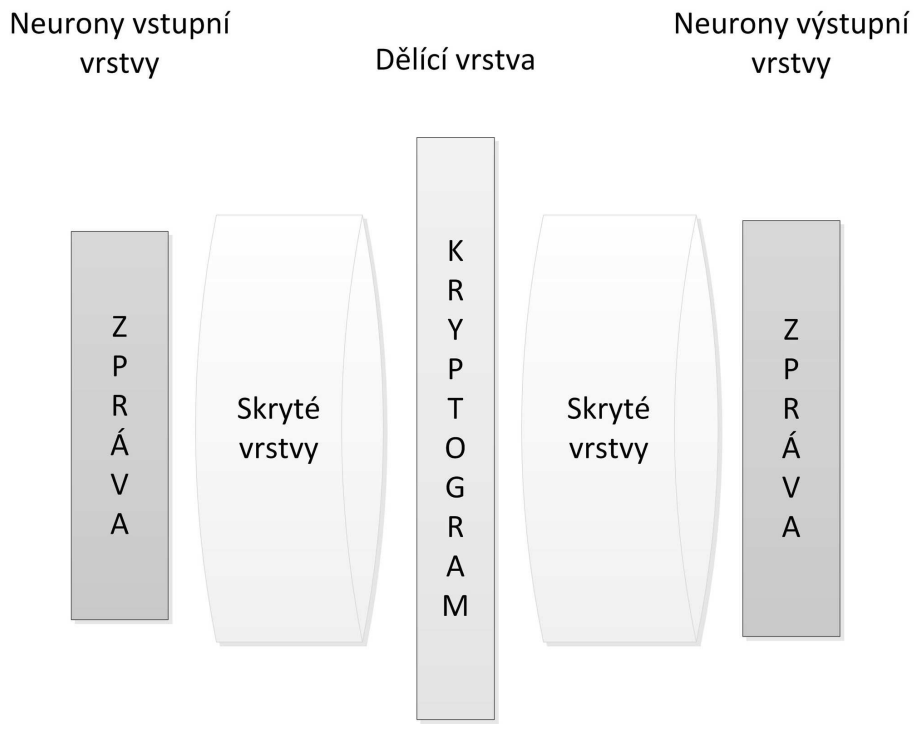
Jak již bylo zmíněno v předchozích kapitolách, neuronová síť v podstatě transformuje vstupní data na libovolná data výstupní. Logicky se nabízí použít takovou síť jako šifrátor a data zpět dešifrovat pomocí inverzně naučené sítě.

Toto ovšem není příliš vhodné řešení. Mnohem vhodnější je vzít do úvahy fakt, že data se transformují uvnitř sítě pomocí vah a biasů. Tento fakt, dělá situaci do jisté míry snadnější. Naučenou neuronovou síť je třeba na vhodném místě rozdělit a části

použít odděleně jako šifrátor a dešifrátor. Kryptogram poté tvoří data z výstupu neuronů dělící vrstvy.

Při použití neuronové sítě jako šifrátoru a dešifrátoru v jednom, je tedy potřeba docílit toho, aby hodnoty vstupující do sítě ve stejné podobě ze sítě i vystupovali. Tohoto se dá snadno docílit vytvořením sítě typu back-propagation, která byla popsána v teoretickém úvodu této práce.

Jako tréninkovou množinu je nutné zvolit námi vybranou abecedu a síť na ní naučit, tzn. vstupní množina a tréninková data se musí rovnat.



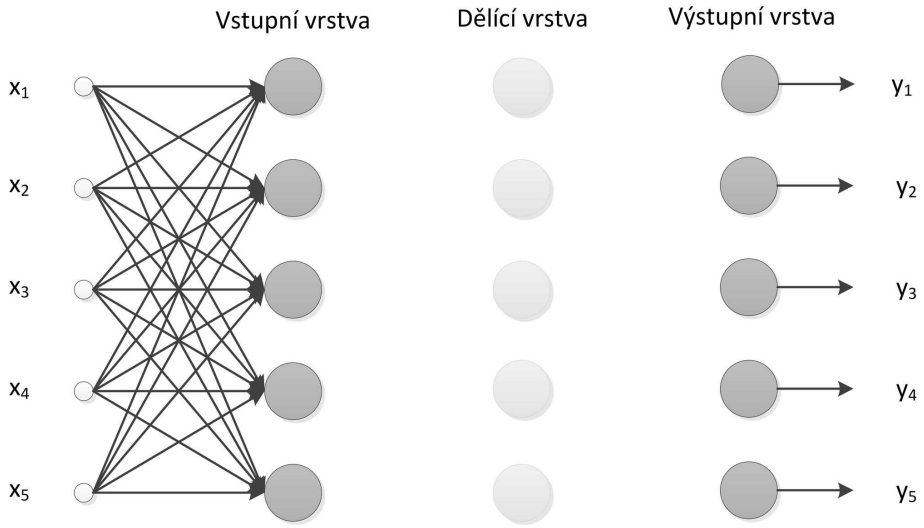
Obr. 4.5: Možnost využití neuronové sítě jako koncového šifrátoru

4.2.2 Abeceda

Nejdříve je třeba si uvědomit jaká data budou do sítě vstupovat a na jaká se bude síť učit. Je třeba vytvořit si tzv. abecedu, tj. množinu všech znaků, které se budou šifrovat. Tyto znaky se poté nahradí vhodnými daty. Možnosti jsou rozsáhlé a záleží jen na fantazii jejich tvůrce avšak neuronová síť pracuje s čísly a proto je třeba písmena nahradit určitými číselnými kombinacemi (binární kombinace, ASCII kód apod.). Velice dobrou volbou je použít binární kombinace.

Běžná abeceda, bez zvláštních znaků, číslic a písmen s háčky a čárkami, má 26 znaků. Abychom obsáhli celou běžnou abecedu, je třeba N bitů. Nejmenším možným

počtem bitů (z důvodu šetření adresním prostorem) je N=5, protože $2^5 = 32$. Každé písmeno tedy musí být popsáno pěti bity. Při počtu znaků 26 potom zbývá 6 nevyužitých kombinací, které můžeme dále využít. Obrázek 4.6 je převzat z literatury [6].



Obr. 4.6: Topologie neuronové sítě jako koncového šifrátoru

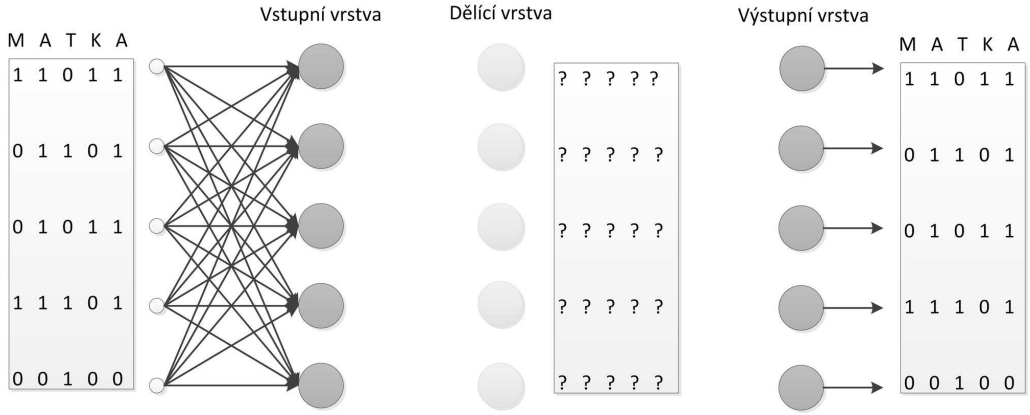
Vhodné zvolení vstupní abecedy je také důležité pro zvolení topologie sítě. Při takto zvolené abecedě, kdy každé písmeno je popsáno pěti bity, bude mít vstupní vrstva pět neuronů. Taktéž vrstva výstupní bude tvořena pěti neurony. Následná topologie sítě je libovolná. Důležité však je, aby se síť správně adaptovala na množinu.

Tab. 4.1: Zvolená abeceda

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	z
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Je třeba zajistit, aby se síť naučila na všech 26 binárních kombinací, tzn. co je zadáno na vstupu, bude i na výstupu. Pokud je tedy zvolena topologie jako na obr. 4.6, v jedné síti je obsáhnut jak šifrátor, tak dešifrátor. Příklad zvolené abecedy je uveden v tabulce 4.1.

Příklad funkce takovéto sítě, ukazuje obr. 4.7. Na vstup sítě je přivedeno slovo „MATKA“. Do sítě vstupuje písmeno po písmenu. Síť je rozdělena na libovolné dvě části (kterákoliv vrstva může být dělící) a výstup z této vrstvy tvoří zašifrovanou zprávu, neboli kryptogram. Stejná kombinace bitů - kryptogram, potom vstupuje do druhé části sítě a je již naučenou sítí zpět transformována na vstupní slovo. Obrázek 4.7 je převzat z literatury [6].



Obr. 4.7: Příklad šifrování slova MATKA

4.3 Vytvoření neuronové sítě v prostředí MATLAB

Tato kapitola se zabývá možnostmi vytvoření vlastní neuronové sítě, jejím učením a simulací.

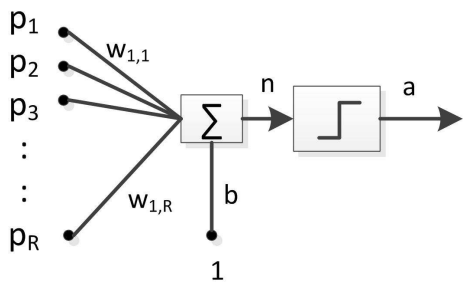
Práce se zaobírá vytvořením dopředné neuronové sítě s využitím algoritmu zpětného šíření chyby (back-propagation), který byl popsán v teoretickém úvodu. Pro vytvoření je využito funkcí z knihovny Neural Network Toolbox, který musí být nainstalován spolu s programem MATLAB a umožňuje uživatelsky velice jednoduše a efektivně definovat neuronové sítě různých typů.

4.3.1 Zobrazování neuronových sítích v programu MATLAB

Na obrázku 4.8, je uveden model neuronu tak, jak ho definuje prostředí MATLAB. Obrázek 4.8, převzat z literatury [9].

Vstupní vektor p je tvořen množinou p_1 až p_R. Tyto vstupy jsou váhované maticí vah W. Bias b je od matice vah oddělený a tvoří ho samostatný vektor pro každou vrstvu sítě a v průběhu trénování se oba vektory mění.

Výstup sítě tvoří obyčejně více neuronů, výstupní veličina a tedy bude tvořena vektorem o stejné velikosti jako je počet výstupních neuronů.



Obr. 4.8: Model neuronu v programu MATLAB

Jako přenosovou funkci dovoluje MATLAB zvolit jakoukoliv dříve popsanou funkci. Kromě předem definovaných funkcí, dovoluje uživatelovi definovat i funkci vlastní. Přenosové funkce jsou například tyto [9]:

- hardlim - prahová funkce (tvrdá limita),
- purelin - lineární funkce,
- tansig - sigmoidní funkce,
- logsig - logická sigmoida.

4.3.2 Vytvoření sítě

Dopředné neuronové sítě většinou využívají výše uvedených přenosových funkcí, volba funkce je víceméně na volbě uživatele. Protože síť v této práci pracuje s binárními vektory jako vstupy a výstupy sítě, je použita přenosová funkce logická sigmoida, která pracuje s hodnotami v rozsahu $\langle 0, 1 \rangle$. Přenosové funkce se dají definovat pro každou vrstvu zvlášť.

Nová síť se vytváří příkazem `newff`, který má následující parametry:

```
net = newff(minmax(P),[5,20,30,20,5],'logsig','logsig','logsig',
'logsig','logsig','traingd');
```

První parametr vyjadřuje rozsah hodnot vstupního vektoru (v tomto případě $\langle 0, 1 \rangle$), následuje počet neuronů v jednotlivých vrstvách. Dále je definována přenosová funkce pro každou vrstvu zvlášť a nakonec je uveden typ algoritmu trénování sítě (traingd - gradient descent training). Hodnoty vah a biasů jsou v této fázi nastaveny náhodně.

Topologie sítě v této práci byla zvolena náhodně a s ohledem na co nejlepší adaptaci sítě na zvolenou abecedu. Vstupní a stejně tak i výstupní vrstva má 5 neuronů protože každé slovo ze zvolené abecedy je reprezentováno pěti bity. Skryté vrstvy potom obsahují 20, 30 a 20 neuronů. Jako přenosová funkce je zvolena logická sigmoida (logsig) pro každou z pěti vrstev. Propojení sítě je definováno tak, že každý

neuron ve vrstvě předcházející je spojen s každým neuronem na následující vrstvě.

V této fázi je vytvořená síť tzv. nenaučená a po přivedení jakéhokoli dříve definovaného vstupního vektoru na vstup sítě, by na výstupu sítě byli jen nesmyslné hodnoty (závislé na počátečním náhodném nastavení vah a biasů). Síť je tedy nejprve potřeba naučit na tréninkový vzor.

4.3.3 Trénování sítě

Trénování sítě je další krok po jejím vytvoření. Základní trénovací algoritmus použitý v této práci je Back-propagation popsaný dříve. Tento algoritmus může být vykonáván v inkrementálním nebo dávkovém módu. V inkrementálním módu se výpočet gradientu a úprava vah vykoná pro každý vstupní vektor. Naopak v dávkovém módu se váhy upraví až po aplikaci všech vstupních vektorů na vstup sítě, gradienty vypočítané pro konkrétní vstupní vektory se sečtou a výsledná hodnota je použita pro výpočet vah na konci každého cyklu.

Program MATLAB umožňuje využít více variací algoritmu Back-propagation. Tyto varianty se liší především rychlostí konvergence, složitostí a stabilitou [9].

- Batch Gradient Descent (traingd).
- Batch Gradient Descent with Momentum (traingdm).
- Gradient Descent with Variable Learning Rate (traingdx).

Před samotným trénováním sítě musíme definovat tzv. tréninkový vzor. To je vzor na který bude síť naučena. V případě této práce je to dříve zvolená abeceda, tedy posloupnost pěti bitů pro každé písmeno základní abecedy. Tyto proměnné se zadávají v maticovém tvaru, tedy matice vstupních vektorů P a matice cílových vektorů T. Protože je nutné síť naučit tak, aby to co vstupuje do sítě z ní i vystupovalo, budou se oba dva vektory P a T rovnat.

Každé slovo ze zvolené abecedy reprezentuje sloupcový vektor o rozměrech 5×1. Například písmeno „a“:

```
> > a
a =
     1
     1
     1
     1
     0
```

Celý vektor obsahující všechny znaky abecedy má rozměr 5×26 a je, jak bylo uvedeno schodný s vektorem cílových hodnot.

```
   > > P
P =
```

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
```

Pro úspěšnost trénování je dále nutno nastavit parametry spojené se zvoleným algoritmem trénování. Algortimus `traingd` byl zvolen již při vytváření sítě a další jeho parametry jsou:

`epochs` - udává maximální počet cyklů po kterých se trénování zastaví
`goal` - maximální přípustná kvadratická chyba
`time` - maximální čas trénování v sekundách
`min_grad` - minimální velikost gradientu, při kterém se trénování zastaví
`lr` - míra učení

K těmto parametrům sítě přistupujeme jako ke vlastnostem objektu, kterým je proměnná představující síť (`net`) a jsou nastaveny takto:

```
net.trainParam.epochs = 2000;
net.trainParam.goal = 1e-4;
net.trainParam.lr = 0.3;
```

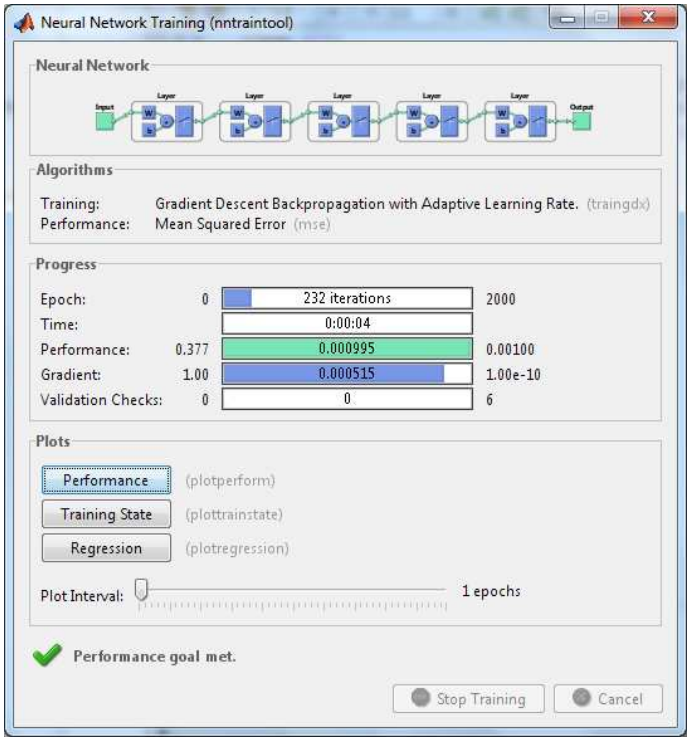
Nastavení těchto parametrů bylo zvoleno s cílem nejlépe adaptovat vytvořenou síť. Vzhledem k citlivosti algoritmu gradient descent na počáteční stav vah je možné, že nebude dosáhnuto požadované maximální kvadratické chyby a učení skončí po dosáhnutí maximálního počtu epoch.

Vlastní trénování sítě spustíme příkazem `train`. Jako parametry tohoto příkazu je třeba zadat trénovanou síť, vektory vstupních hodnot a hodnot očekávaných na výstupu.

```
[net]=train(net,P,T);
```

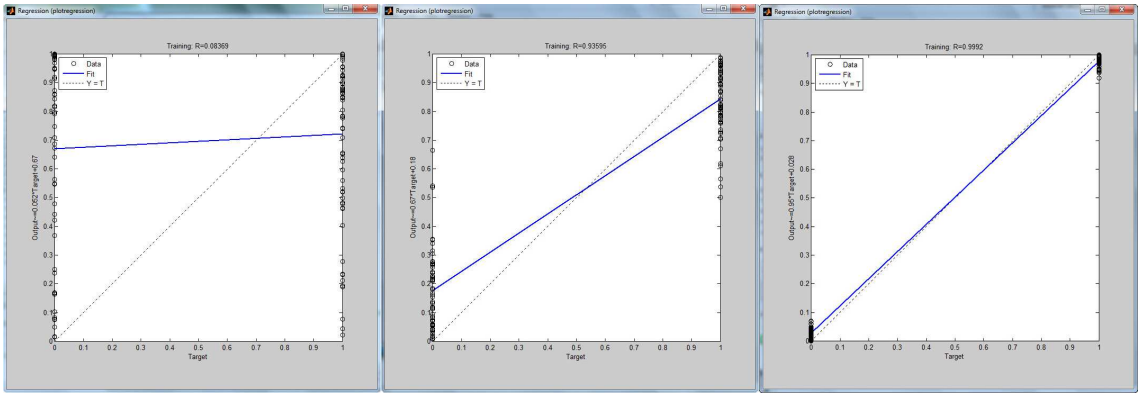
Po zadání příkazu se automaticky spouští dialogové okno nástroje `nntraintool`, které zobrazuje průběh trénování sítě a umožňuje také zobrazit grafy popisující výkon sítě na konci učení i v jeho průběhu.

V horní části je zobrazena topologie učené sítě. V druhé části je uveden zvolený trénovací algoritmus, poté jednotlivé parametry jako je počet iterací, čas, minimální



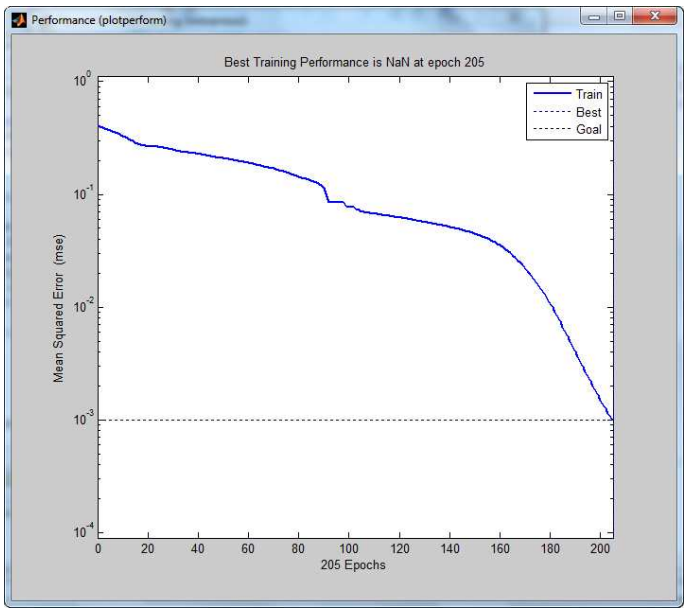
Obr. 4.9: Okno nástroje nntraintool

dosáhnutá kvadratická chyba a gradient. Jak se síť adaptuje můžeme sledovat v průběhu učení. Toto je ukázáno na obr. 4.10.



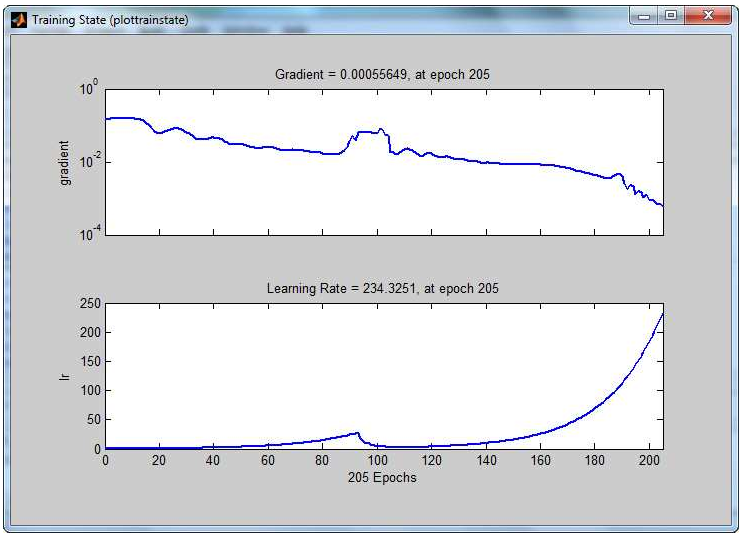
Obr. 4.10: Průběh učení

V tomto nástroji je také možné zobrazit průběh snižování kvadratické chyby v průběhu trénování, tak jak je uvedeno na obr. 4.11.



Obr. 4.11: Průběh zmenšování kvadratické chyby

Zajímavý je taky obrázek 4.12 zobrazující změny gradientu a míry učení v průběhu trénování sítě.



Obr. 4.12: Změna gradientu a míry učení v průběhu trénování

Tabulka 4.2 udává počty iterací za kterých bylo dosáhnuto požadované maximální kvadratické chyby (zde například 0,003) s použitím algoritmu `traingd`. Je zde patrno, že v některých případech požadovaného výsledku dosaženo nebylo. Maximální dosažený počet iterací je dán počátečním nastavením vah příkazem `init`.

Tab. 4.2: Počet iterací s algoritmem `traingd`

Pokus	1	2	3	4	5	6	7
Počet cyklů	3430	3190	3500	2868	2933	3500	2747
Kvadratická chyba	0,003	0,003	0,00863	0,003	0,003	0,00458	0,003

Pro představu o rychlosti konvergence sítě s použitím algoritmu `traingdx` s proměnným parametrem lr - learning rate (míra učení), jsou v tabulce 4.3 uvedeny také počty iterací potřebných k dosažení stejné kvadratické chyby s tímto algoritmem. Použitím tohoto učícího algoritmu je dosáhnuto markantního snížení počtu iterací při zachování stejné kvadratické chyby. To nám dovolí vytvořenou síť naučit ještě mnohem přesněji a je dále v této práci využit.

Tab. 4.3: Počet iterací s algoritmem `traingdx`

Pokus	1	2	3	4	5	6	7
Počet cyklů	252	188	342	196	127	222	129
Kvadratická chyba	0,003	0,003	0,003	0,003	0,003	0,003	0,003

4.3.4 Simulace sítě

V průběhu simulace sítě, přivádíme na vstup sítě hodnoty pro síť neznámé a pozorujeme korektnost jejího výstupu. Jelikož je zvolená trénovací množina složena ze stejných vektorů, očekáváme pro naučenou síť nulovou odchylku vstupu od výstupu. Vlastní simulace se spouští příkazem `sim`:

```
> > A = sim (net,I)
```

Kde I je vstupní sloupcový vektor [1; 1; 1; 0; 0];

MATLAB ukládá do výstupního vektoru A data ze simulace sítě. Jelikož vektory vah a biasů jsou vždy náhodně zvolená čísla a v průběhu trénování sítě se mění, na výstupu sítě budou nebudou čísla binární jako na vstupu, ale jejich přibližné hodnoty. Tyto hodnoty je nutné zaokrouhlit k docílení stejných hodnot které do sítě vstupovali.

```
> > A
```

```
A =

    0.8786
    0.9471
    0.9945
    0.0479
    0.0163
```

Po zaokrouhlení dosáhneme shody vstupu s výstupem.

```
> > Y=round(A);
Y=

     1
     1
     1
     0
     0
```

4.3.5 Export matice vah a biasů

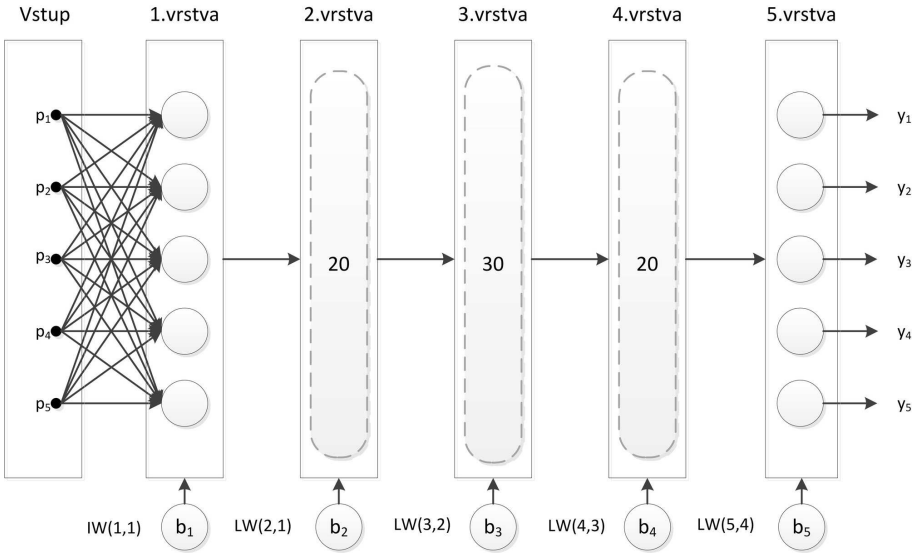
Příkazem newff je vytvořena nová síť typu feed-forward (dopředné šíření signálu). Síť je vytvořena jako objekt a jako s takovým s ní můžeme zacházet. To znamená, že veškeré jeho parametry (s výjimkou parametrů jen pro čtení) můžeme upravovat a přistupovat k nim přes tzv. tečkovou notaci.

Tohoto bylo využito již v sekci vytváření sítě, při nastavování maximálního počtu cyklů učení, minimální kvadratické chyby sítě atd. Stejného principu lze využít i při exportu matic vahových koeficientů a matic biasů. Ve vytvořeném objektu sítě jsou jednotlivé vektory dostupné následovně:

- `net.IW` - matice vstupních vah
- `net.LW` - matice vah jednotlivých vrstev
- `net.b` - matice biasů

Matici vstupních vah vytvořené sítě je získána příkazem `net.IW`$\{1, 1\}$, matice vah jednotlivých vrstev potom `net.LW`$\{M, N\}$, kde koeficienty M a N označují vrstvu následující a vrstvu předcházející. Matice biasů je získána pomocí příkazu `net.b`$\{i\}$, kde číslo i reprezentuje číslo vrstvy.

Rozměr matice závisí na počtu neuronů ve vrstvách a je o velikosti $M \times N$. Rozměr matice pro spojení první vrstvy s vrstvou druhou `net.LW`$\{2, 1\}$ bude mít například rozměr 20×5, pro pět vstupních neuronů a 20 neuronů ve skryté vrstvě.



Obr. 4.13: Export matic a biasů ze sítě

4.3.6 Použitá topologie

Propojení neuronů může být libovolné avšak v praxi je využíváno především sítí vícevrstvých. V těchto jsou neurony uspořádány v několika vrstvách nad sebou.

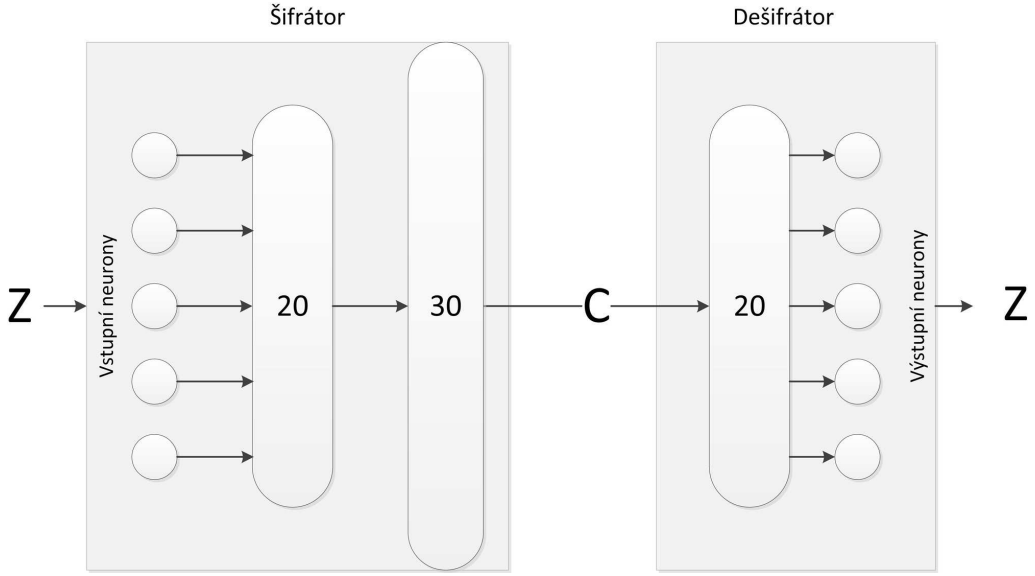
Jakmile byla určena konečná podoba trénovací množiny, mohla být vytvořena podoba modelu neuronové sítě vhodné pro řešení daného problému. Vzhledem ke dříve zvolené abecedě znaků, kde jsou jednotlivá písmena popsána vždy pěti bity, musí mít vytvořená síť pět neuronů na vstupní i na výstupní vrstvě.

Počet neuronů ve skrytých vrstvách záleží více méně na tvůrci sítě. Zvětšuje li se počet skrytých vrstev, narůstá také počet vah a to s násobkem počtu neuronů ve vrstvě. To přispívá k větší náročnosti na prolomení (uhodnutí klíčů) kryptografického systému. Podobného efektu je docíleno i zvyšováním počtu neuronů v již existujících vrstvách. Zvyšování počtu neuronů či skrytých vrstev má ale i své zápory. Při velké složitosti systému může nastat problém již při učení sítě. Síť se jednoduše nemusí naučit správně, nebo proces učení může trvat nepřijatelně dlouhou dobu.

Dalším problémem je velikost dat, které musíme přenést po rozdělení sítě do jejich částí (šifrátoru a dešifrátoru). Daty jsou v tomto případě myšleny vektory váhových koeficientů a vektory biasů.

Neuronová síť v této práci využívá kromě vstupní a výstupní ještě třech skrytých vrstev. Počty neuronů v těchto vrstvách byly zvoleny jako kompromis složitosti a velikosti. První skrytá vrstva obsahuje 20 neuronů, druhá 30 neuronů a poslední opět 20 neuronů. Jako přenosovou funkci všechny neurony využívají logickou sigmoidu. Propojen je každý neuron s každým na následující vrstvě.

Síť je rozdělena na dvě části za druhou skrytou vrstvou. Jako výstup ze šifrátoru



Obr. 4.14: Výsledná topologie neuronové sítě

(kryptogram) je tedy použito 30 neuronů druhé skryté vrstvy. Jiný počet vstupních a výstupních neuronů a tím pádem různý počet bitů na vstupu a výstupu šifrátoru má ještě více znesnadnit útok a dešifrování kryptogramu.

Část dešifrátoru potom tvoří dvě vrstvy a to 20 neuronů a výstupních 5 neuronů, které dešifrují kryptogram zpět na zprávu.

4.4 Vytvoření neuronové sítě v prostředí SIMULINK

Tato část se zabývá vytvořením modelu šifrátoru a dešifrátoru s využitím neuronové sítě v prostředí MATLAB SIMULINK. Tento způsob má svoje výhody i nevýhody.
Výhody:

- Názornost,
- možnost zobrazení dílčích výsledků,
- modelování vlastní sítě.

Nevýhody:

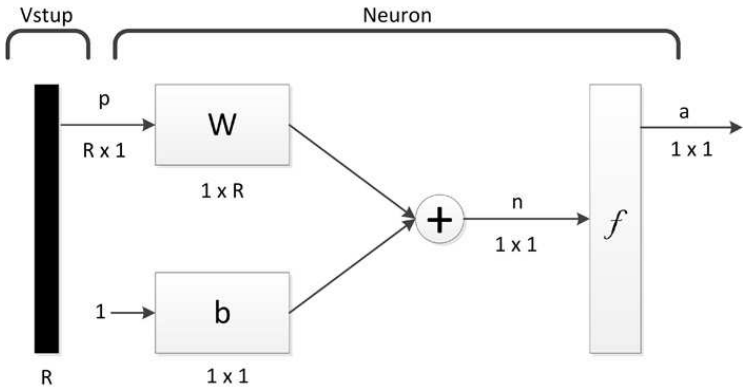
- Složitost,
- časová náročnost,
- možnost udělat snadno chybu.

Nespornou výhodou při vytváření neuronové sítě v prostředí SIMULINK je bezpochyby přehlednost a fakt, že je celá síť tzv. řečeno „pod kontrolou“. Topologie a propojení neuronů je názornější a dá se snadno jakkoli měnit. Nevýhodou může být

značná složitost a zdlouhavost vytváření modelu. Ve velkém modelu sítě lze také mnohem snáze udělat chybu. Toto vše je závislé na použité topologii. Proto zvolená topologie v této práci brala tento fakt v úvahu.

Pro vytváření neuronových sítí je v programu SIMULINK tzv. toolbox (knihovna) jménem „Neural network toolbox“, který obsahuje všechny potřebné bloky pro sestavení modelu. Použité bloky budou blíže popsány v dalších kapitolách.

Obrázek 4.15 zobrazuje zjednodušený model neuronu v prostředí SIMULINK [9], který je základem k pozdějšímu vytváření celé sítě neuronů.



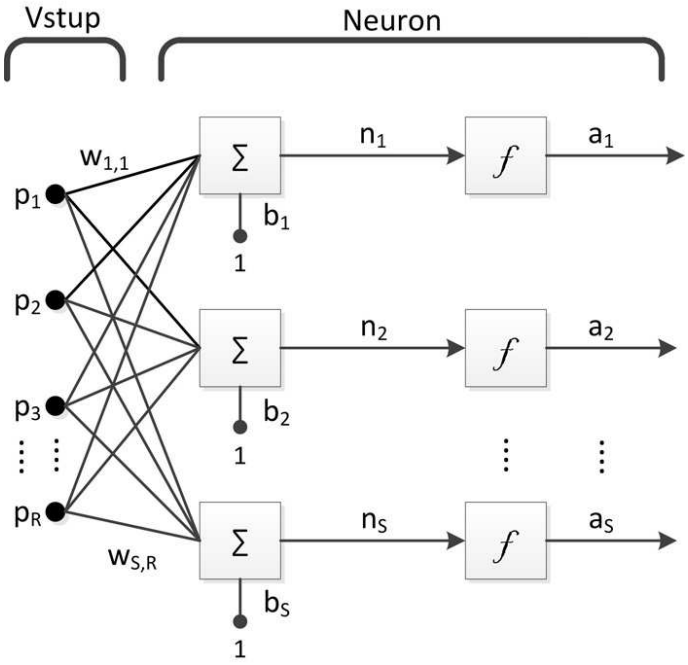
Obr. 4.15: Model neuronu v programu SIMULINK

Vlevo se nachází vstupní vektor p o velikosti R, ten se násobí s maticí vah a biasem b. Skalární vektor n reprezentuje vstup váhovaných vstupů a biasů do přenosové funkce f. Výstupní hodnota a je skalár. V případě více neuronů, bude výstupní hodnota vektor [9].

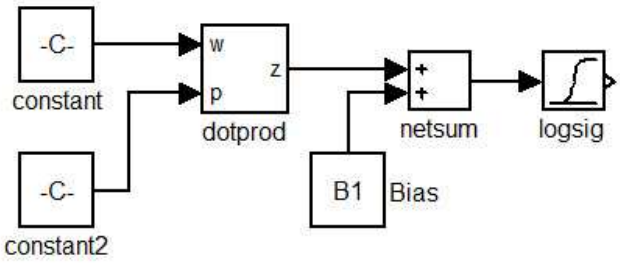
Dva a více modelů neuronů lze spojit do vrstvy, kterých může mít síť libovolné množství (obr. 4.16, převzato z literatury [9]). R reprezentuje počet prvků ve vstupním vektoru a S počet neuronů ve vrstvě. V této síti je každý prvek vstupního vektoru p spojen ke každému neuronu přes matici vah. Každý neuron má svůj bias a svoji vlastní přenosovou funkci. Výstup neuronu je poté tvořen sloupcovým vektorem a. Je vhodné poznamenat, že je běžné, když se liší počet vstupů do vrstvy a počet neuronů ve vrstvě.

4.4.1 Popis funkčních bloků

Následující podkapitola se zabývá popisem funkčních bloků, ze kterých je výše uvedený model sestaven a jejich popisem. K jeho sestavení je použito běžně používaných bloků a bloků speciálních, z knihovny Neural Network Toolbox. Bloky, ze kterých je model neuronu postaven, jsou patrné z obr. 4.17.



Obr. 4.16: Síť neuronů v programu SIMULINK

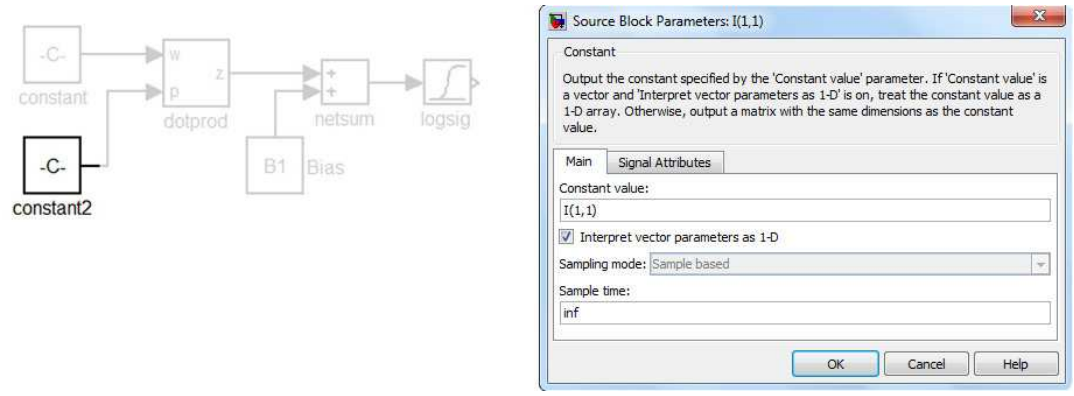


Obr. 4.17: Neuron sestaven z funkčních bloků

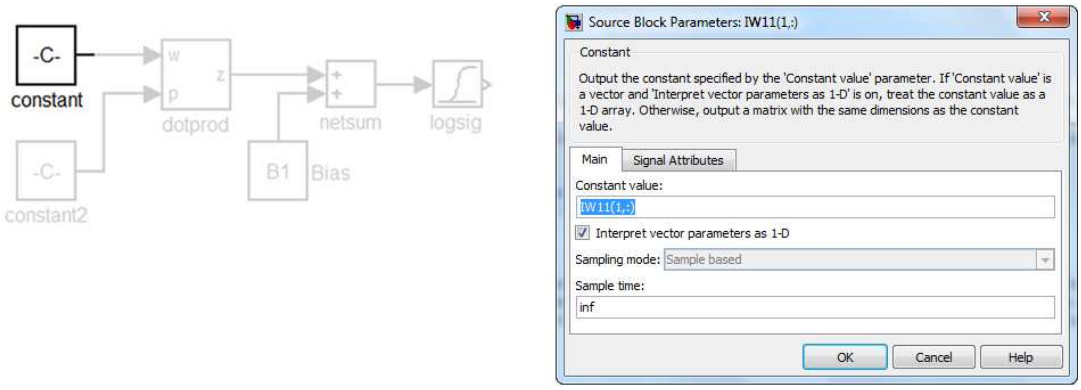
Blok Constant

Blok Constant produkuje konstantní signál definovaný buď v samotném okně nastavení bloku, nebo signál definovaný dříve a uložený do patřičné proměnné. Tak je tomu i v této práci. Jednotlivé bloky konstant čtou příslušné hodnoty vektorů dříve exportovaných z naučené sítě. Například vstupní blok Constant čte první řádek vstupního sloupcového vektoru.

Stejně jako hodnoty ze vstupu i hodnoty jednotlivých vah jsou postupně vyčítány z uložených hodnot pomocí bloku Constant. Jako příklad je uvedena první (vstupní) vrstva. Jak je vidět z obrázku 4.19, výstup bloku je první řádek a všechny sloupce z vektoru IW. To je reprezentováno zápisem ve tvaru IW11(1,:), kde vektor IW11 je vektorem vah mezi vstupy sítě a první vrstvou.



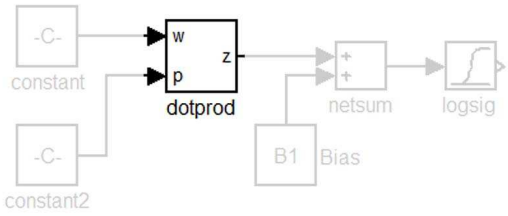
Obr. 4.18: Block Constant jako vstup do sítě



Obr. 4.19: Block Constant jako vektor vah

Blok Dotprod

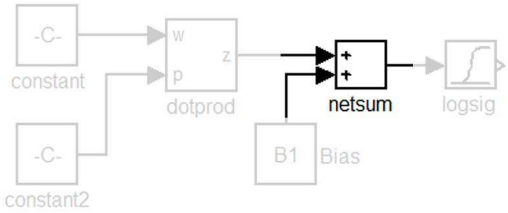
Blok Dotprod spojuje bloky Constant, kde jeden je vstup do neuronu a druhý vyjadřuje váhu. Je to váhovací blok, v podstatě zajišťuje aby na jeho výstupu byla váhovaná hodnota přivedena na vstup p vektorem vah ze vstupu w. Blok Dotprod nemá žádné možnosti nastavení.



Obr. 4.20: Blok dotprod

Blok Netsum

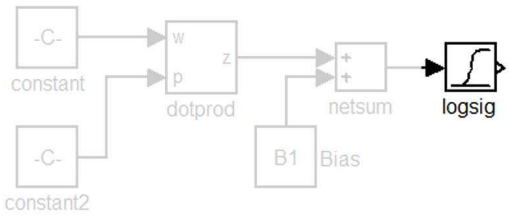
Blok Netsum je blok sumační. Jeho vstupy jsou váhované vektory vstupu a bias. Výstup tvoří kombinaci těchto vstupů a tento je dále přiveden do bloku přenosové funkce.



Obr. 4.21: Blok netsum

Blok Logsig

Blok Logsig tvoří přenosovou funkci neuronu. Tento blok nemá žádné možnosti nastavení. Pro změnu přenosové funkce je třeba ho nahradit blokem jiným. Například z knihovny Transfer functions bloky purelin (lineární funkce), tansig (hyperblický tangent). Výstup z tohoto bloku je zároveň výstupem neuronu (neuronů).



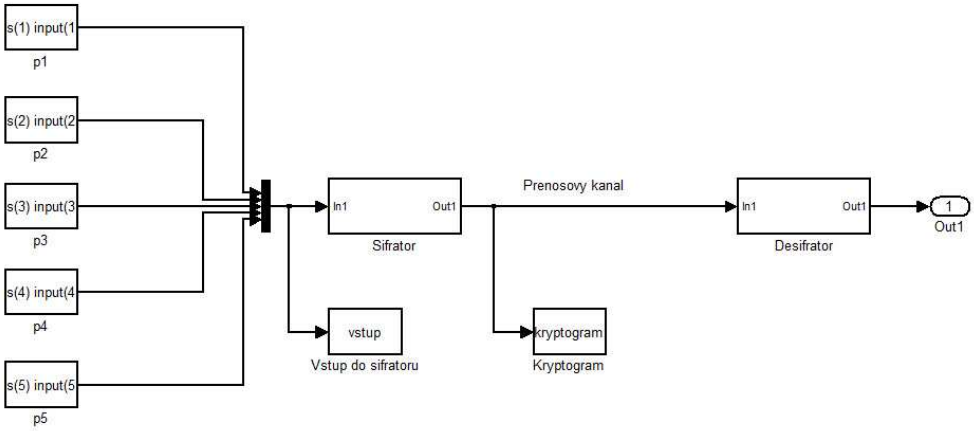
Obr. 4.22: Blok přenosové funkce logsig

Všechny tyto bloky tvoří jeden neuron, tak jak je zobrazen na obr. 4.17. Takto vytvořené neurony můžeme dále spojit do jednotlivých vrstev.

4.4.2 Vytvoření šifrátoru a dešifrátoru

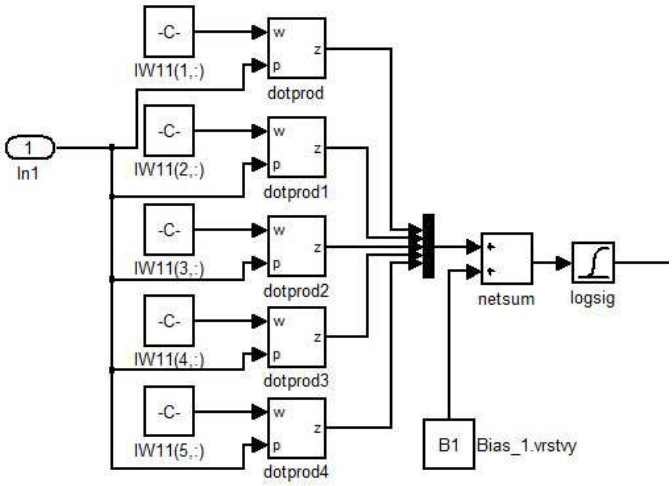
Z výše popsaných funkčních bloků jsou sestaveny jednotlivé neurony ve vrstvách neuronové sítě. Na základní úrovni se nachází dva subsystémy. Tyto subsystémy tvoří blok šifrátoru a blok dešifrátoru. Do šifrátoru vstupuje pět bloků Signal from

workspace, které reprezentují pět vstupů do systému, ty jsou dále multiplexovány do jednoho vstupního vektoru. Toto je zobrazeno na obr. 4.23.



Obr. 4.23: Model šifrátoru a dešifrátoru

Spojení neuronů do vrstvy je ukázáno na obr. 4.24. Jako příklad je uvedena první vrstva. Tato vrstva obsahuje pět neuronů. Do této vrstvy je přiveden vstupní sloupcový vektor o velikosti 5×1.



Obr. 4.24: Zapojení první vrstvy šifrátoru

Spojení je provedeno následovně, vstupní vektor je přiveden na vstup p všech pěti bloků dotprod. Do vstupů označených w vstupuje signál z bloku constant, na jehož výstupu je signál reprezentující příslušnou část z vektoru vah dané vrstvy. V bloku dotprod se provede váhování vstupních hodnot p příslušnou částí vektoru vah.

Takto získaný signál je multiplexován se signály na výstupu ostatních bloků dotprod. Vektor těchto hodnot je následně sečten s vektorem biasů příslušné vrstvy v bloku netstum.

Vektor na výstupu bloku netsum je dále přiveden do bloku přenosové funkce, v tomto případě blok logsig. Výstup z bloku přenosové funkce tvoří výstupní vektor z první vrstvy. Takto je vytvořena jedna vrstva sítě, tyto vrstvy se dále spojují tak, že výstup vrstvy je přiveden na vstupy p všech neuronů vrstvy následující.

Existují více způsobů jak spojit libovolný počet neuronů v jednu vrstvu. Lze spojit neurony z obrázku 4.17, kde má každý neuron svůj sumační blok s hodnotou příslušného biasu a přenosovou funkci, nebo to udělat tak, jak je uvedeno na obrázku 4.24 reprezentující první vrstvu. U první varianty vyčítáme z matice biasů jednotlivých vrstev vždy hodnotu pro daný neuron a můžeme samostatně pro každý neuron ve vrstvě nastavit odlišnou přenosovou funkci, což může být v některých případech výhodné. U varianty z obrázku 4.24 reprezentuje daný počet neuronů ve vrstvě počet vstupů a bloků dotprod. Ty jsou následně multiplexovány a přivedeny do jednoho společného sumačního bloku s celým vektorem biasů pro danou vrstvu a jedním typem přenosové funkce. Výhody jsou následující, v praxi se běžně nenastavují různé přenosové funkce pro jednotlivé neurony ve vrstvě a tak je možné celý model udělat více přehledným, vynecháním těchto bloků a nastavením jedné přenosové funkce pro všechny neurony najednou. Rovněž vektor biasů je možné přivést do jediného sumačního bloku jako sloupcový vektor reprezentující všechny biasy pro danou vrstvu a tím ušetřit další bloky.

4.5 Útoky na neuronovou síť

Využití neuronové sítě jako koncového šifrátoru a dešifrátoru vytváří tzv. symetrický šifrovací systém. Ačkoli se nepoužívá stejný klíč k šifrování i dešifrování, dvojice šifrátor a dešifrátor používá v podstatě jeden klíč složený z vah a biasů jednotlivých vrstev, rozdělený na dvě části. Délka tohoto klíče je potom dána součtem všech dílčích vah a biasů v neuronové síti. Součástí klíče, tedy tajnou informací která se přenáší oběma částem sítě, je i informace o topologii s jasně daným místem rozdělení sítě na tyto části.

Jelikož jednotlivé váhy neuronů a jejich biasy nejsou celá čísla, ale čísla reálná o určité náhodné velikosti, počet kombinací a tím pádem složitosti nalezení správného klíče rapidně roste. Délka klíče se může pro danou topologii lišit v závislosti na velikosti vah a biasů.

V případě topologie použité v této práci se v celé síti nachází celkem 1425 jednotlivých vah a 80 biasů. Každá z vah je určena minimálně jedním číslem, přičemž

maximální počet závisí na na procesu učení a na programu MATLAB samotném. Průměrně se jednotlivé váhy skládají ze 4 čísel, což je při využití čísel 0 - 9 deset tisíc kombinací na jednu váhu. Takto obrovské číslo kombinací všech vah je útokem hrubou silou (brute force) prakticky nerozluštitelné. V praxi není nutné uhádnout všechny čtyři číslice váhy se stoprocentní přesností, ale je třeba se jim alespoň dostatečně přiblížit, protože výsledné hodnoty vystupující ze sítě nejsou úplně přesně ty hodnoty, které do ní vstupovali, ale hodnoty zaokrouhlené.

Jak už bylo zmíněno, daný systém využívá jeden klíč rozdělený na dvě části. Tímto vyvstává problém distribuce klíčů a jejich zabezpečení. Tímto problémem se zabývá problematika klíčového hospodářství. Je zapotřebí dodržovat následující požadavky [7]:

- klíče musí být voleny z množiny všech možných klíčů náhodně,
- klíče musí být chráněny minimálně takovým způsobem jako jsou chráněny jimi šifrované informace,
- klíče by se měly často měnit (optimálně pro každou přenášenou zprávu nový klíč),
- pro každou komunikující dvojici šifrátorů musí existovat jiný klíč.

Požadavek o náhodné volbě klíče splňuje samotná síť během procesu učení. Další požadavky již musíme zajistit sami.

V každém útoku se má za to, že útočník E může odposlouchat libovolný počet zpráv mezi komunikujícími stranami A a B, ale nemůže je měnit.

Pokud by byl kryptogram změněn, bez potřebné znalosti k provedení této změny ve snaze modifikovat zprávu a zmást tak příjemce, dešifrátor změněný kryptogram vstupující do jeho sítě nerozluští správně. Je tak zajištěna i jistá autentizace vysílané strany.

Jestliže jsou za tajné informace pokládány klíč (tedy vektory vah a biasů) a topologie, zachycení jedné z těchto informací nepovede k úspěšnému rozluštění kryptogramu. Při zachycení celého klíče, tedy všech vektorů vah a biasů, není schopen útočník kryptogram rozluštit. Není mu totiž známa topologie sítě a tudíž místa použití jednotlivých částí klíče.

Další možností je zachycení klíče i topologie sítě. Ani zachycení obou těchto informací nemusí nutně vést k prolomení protokolu. Velice důležitá je informace o části sítě, kde dochází k jejímu dělení, bez této informace nelze žádný kryptogram rozluštit. Z předchozího vyplývá, že tyto tři informace je vhodné držet v utajení (obzvláště informace o topologii a rozdělení sítě), zabezpečit je proti úniku a distribuovat k jednotlivým částem systému šifrovaně.

Poslední částí tohoto kryptosystému, která může být potenciální slabinou je samotná abeceda zvolená uživatelem, tedy posloupnost bitů zastupující jednotlivá písmena abecedy. Při nezatajení zvolené abecedy, nebo jejím vyzrazení či úniku,

vystavuje se uživatel značnému riziku prolomení daného kryptosystému. Většinou se abeceda během přenosu nemění a zůstává stejná jako na začátku přenosu. Jelikož se nemění ani složení klíče kterým se daná abeceda šifruje (tedy váhy, biasy a topologie sítě), nemění se ani tvar a obsah kryptogramů. Útočník tedy může z dostatečného množství zachycených kryptogramů za určitých podmínek (například na základě četnosti výskytu jednotlivých znaků) sestavit uživatelem danou abecedu a tím šifru prolomit. Řešením je měnit abecedu dostatečně často, stejně jako klíč. To ovšem sebou nese značné nároky na režii při přenosu těchto tajných informací a také nároky adresní (po vyčerpání kombinací je zapotřebí zvýšit počet bitů v abecedě).

4.6 Spuštění a ovládání programu

Po zapnutí programu MATLAB je třeba otevřít program *start.m*. Spuštěním tohoto souboru se rozběhne hlavní program obsahující veškeré příkazy.

Jednotlivé instrukce programu se vypisují do konzole programu MATLAB. V prvním kroku se vytvoří síť a začne její trénování, což je znázorněno dialogovým oknem nástroje nntraintool. Zde lze také zobrazit grafy z průběhu učení sítě.

Dále je uživatel vyzván k zadání slova které chce programem šifrovat. Po zadání slova je spuštěn model *sif_desif.mdl*, toto může trvat delší dobu vzhledem ke spouštění programu SIMULINK ve kterém je model vytvořen. Je vytvořena matice vstupních dat pro simulaci a uživatel je konzolí vyzván aby zahájil simulaci.

V posledním kroku program zobrazí v konzoli programu výsledná data jako vstupní slovo, jeho matice hodnot, kryptogram a výslednou matici hodnot spolu se zpětně získaným slovem.

Program a simulace byly vytvořeny v programu MATLAB verze 7.6.0 (R2008a), proto nemusí v jiných verzích fungovat zcela bezchybně. Zároveň je třeba mít nainstalované prostředí SIMULINK a knihovnu Neural network toolbox.

5 ZÁVĚR

Tato práce se věnovala především popisu možností využití umělé inteligence v kryptografii. V teoretickém rozboru byla popsána historie vývoje neuronových sítí, slavné osobnosti. Dále byla popsána snaha o napodobení biologického neuronu a vzniku neuronu formálního. Následoval popis hlavního stavebního prvku neuronových sítí - perceptronu, jeho učícího algoritmu a druhů přenosových funkcí, které může využívat.

V kapitole o neuronových sítích byl popsán způsob jakým se perceptrony spojují a vytvářejí tak síť, která dokáže klasifikovat problém do více tříd než samotný prvek. Bylo zde uvedeno, jaké druhy sítí existují a způsoby učení: s učitelem a bez učitele. Vybrán byl vhodný algoritmus pro pozdější použití v této práci a to algoritmus Back-propagation. Tento byl podrobně rozebrán včetně kroků učení a použitých algoritmů.

Po menším úvodu do kryptografie v kapitole o využití umělé inteligence v kryptografii byly nastíněny možnosti využití prvků umělé inteligence v tomto vědním oboru. Jelikož jsou neuronové sítě poměrně mladé výpočetní paradigma a obor kryptografie je otevřen novým možnostem výpočtů, použití umělé neuronové sítě se zde logicky nabízí. V úvodu této práce popsaný algoritmus Back-propagation byl využit k vytvoření vlastní neuronové sítě s uživatelem zvolenou topologií. Tato síť byla naučena na tréninkový vzor který tvoří zvolená abeceda. Abeceda byla zvolena tak, aby každé písmeno reprezentovala binární kombinace o velikosti 5 bitů. Pro názornost byl v programu SIMULINK vytvořen model sítě, který využíval neurony dříve sestavené z funkčních bloků. Tato síť byla rozdělena do částí šifrátor a dešifrátor. Jako kryptogram byl zvolen výstup ze třetí vrstvy modelu, tento poté reprezentuje šifrované slovo přenášené přenosovým kanálem.

Tímto způsobem byla ověřena vhodnost použití prvků umělé inteligence v kryptografii a byl vytvořen zcela nový bezpečnostní protokol využívající sítě typu Back-propagation. V podstatě se jedná o symetrický kryptosystém, který využívá jako klíč váhové koeficienty sítě a především informaci o její topologii. Výhodou je obrovská velikost klíče, díky které se stává útok hrubou silou takřka nepoužitelný. Nevýhodami k bezpečnému fungování tohoto kryptosystému, zmíněnými v závěru této práce, je potřeba držet v tajnosti velké množství informací a jako u každého symetrického protokolu, potřeba měnit poměrně často šifrovací klíč.

Využití umělé inteligence a neuronových sítí v kryptografii může mít nespočet podob, které se budou do budoucna vyvíjet, v této práci byl nastíněn jen malý zlomek možností jejich použití.

LITERATURA

[1] NERUDA, R., ŠÍMA, J. *Teoretické otázky neuronových sítí*. Praha: MATFYZPRESS, 1996. 390 s.

[2] VOLNÁ, E. *Neuronové sítě 1*. Ostrava: Ostrvská universita, Přírodovědecká fakulta, 2002. 85 s.

[3] JIRSÍK, V., HRÁČEK, P. *Umělá inteligence. Elektronický text. AMT008*. Brno: VUT FEKT, 2002. 107 s.

[4] BABNIČ, P. *Možnosti využití neuronových sítí v síťových prvcích.*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 75s.

[5] NOVÁK, M. a kolektív *Umělé neuronové sítě: teorie a aplikace. 1. vydání.*. Praha: C. H. Beck, 1998. 382 s. ISBN 80-7179-132-6.

[6] BABNIČ, P., ROSENBERG, M. *Využitie viacvrstvovej neurónovej siete v kryptografii* [online]. Elektrorevue - Internetový časopis, roč. 2012, č. 31, ISSN: 1213- 1539. Poslední aktualizace 09.05.2012 [cit. 13.5. 2012]. Dostupné z URL: *http://www.elektrorevue.cz/*

[7] BURDA, K. *Bezpečnost informačních systémů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2005. 104 s.

[8] SHIHAB, K. *A backpropagation neural network for computer network security* [online]. J. Comput. Sci., 2: 710-715, 2006, poslední aktualizace 2006 [cit. 13.5. 2012]. Dostupné z URL: *http://thescipub.com/abstract/10.3844/jcssp.2006.710.715*

[9] BEALE, M. H., HAGAN, M. T., DEMUTH, H. B. *Neural Network Toolbox, User's Guide* [online]. 2012, poslední aktualizace Březen 2012 [cit. 13.5. 2012]. Dostupné z URL: *http://www.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf*

SEZNAM PŘÍLOH

A SEZNAM SOUBORU NA PŘILOŽENÉM CD

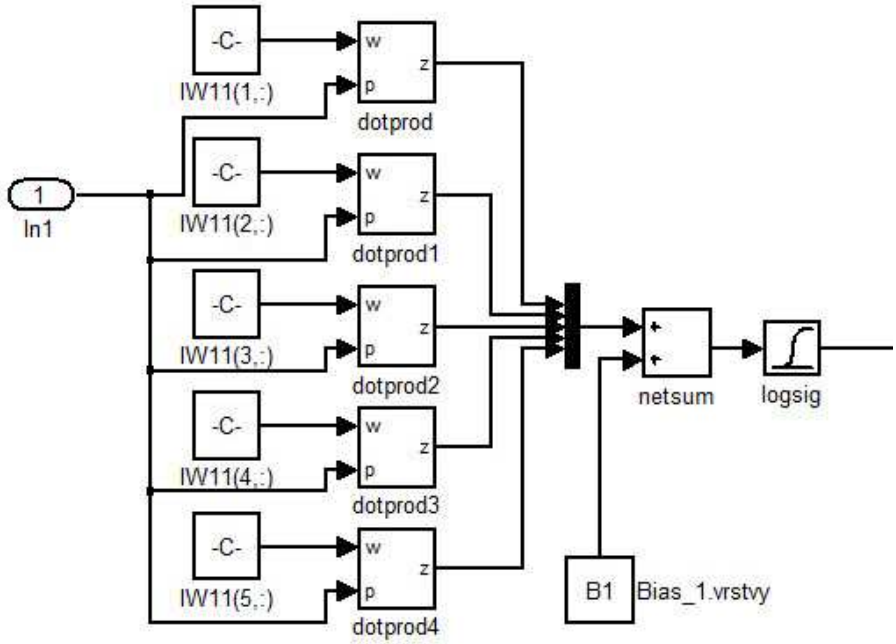
- Elektronická verze práce
- soubor *start.m*
- model *sif_desif.mdl*

B ZÁKLADNÍ POJMY K NEURONOVÝM SÍTÍM

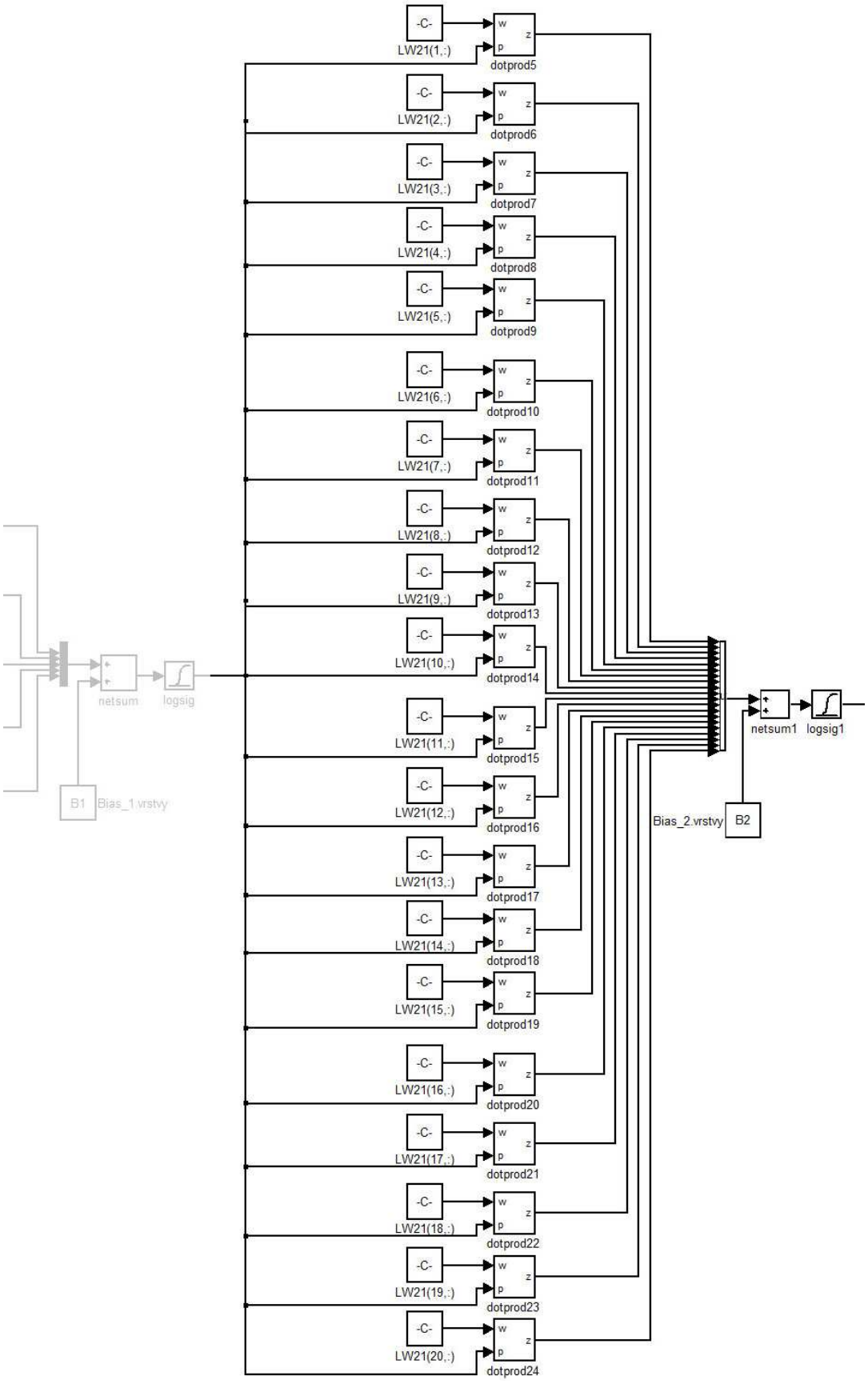
V příloze jsou uvedeny základní pojmy z oblasti neuronových sítí. Text je převzat z literatury [3].

- adaptace - schopnost umělé neuronové sítě k samoorganizaci. Realizuje se obvykle změnami vah během učení,
- architektura - struktura sítě výkonných prvků, jejich vzájemné propojení,
- algoritmus - metoda nebo procedura pro získání cíle nebo řešení,
- excitace - takové působení neuronu, že při něm v připojených neuronech dochází k růstu jejich vnitřního potenciálu,
- energetická funkce - energie je mírou naučenosti, tedy odchylky mezi skutečnými a požadovanými hodnotami vstupů neuronové sítě pro danou trénovací množinu,
- klasifikace - schopnost přiřadit vstupní pozorovaný vzor nebo znak do určité kategorie,
- znak nebo vlastnost - něco co charakterizuje vlastnost nějakého objektu nebo situace,
- heuristika - metoda používaná podle zkušeností, ale nemusí být garantované řešení problému,
- přeučování - učící proces, ve kterém se maže jistý počet vah. V kontrastu k normálnímu učení už při přeučování sítě jistým objemem dat obsahovala,
- samoorganizace - schopnost neuronové sítě učením přizpůsobit své chování k vyřešení daného problému,
- síť - spojení vzájemně spojených entit,
- topologie - popisuje druh a počet výkonných prvků sítě a strukturu (graf) jejich propojení,
- jednotka - elementární objekt umělé neuronové sítě.

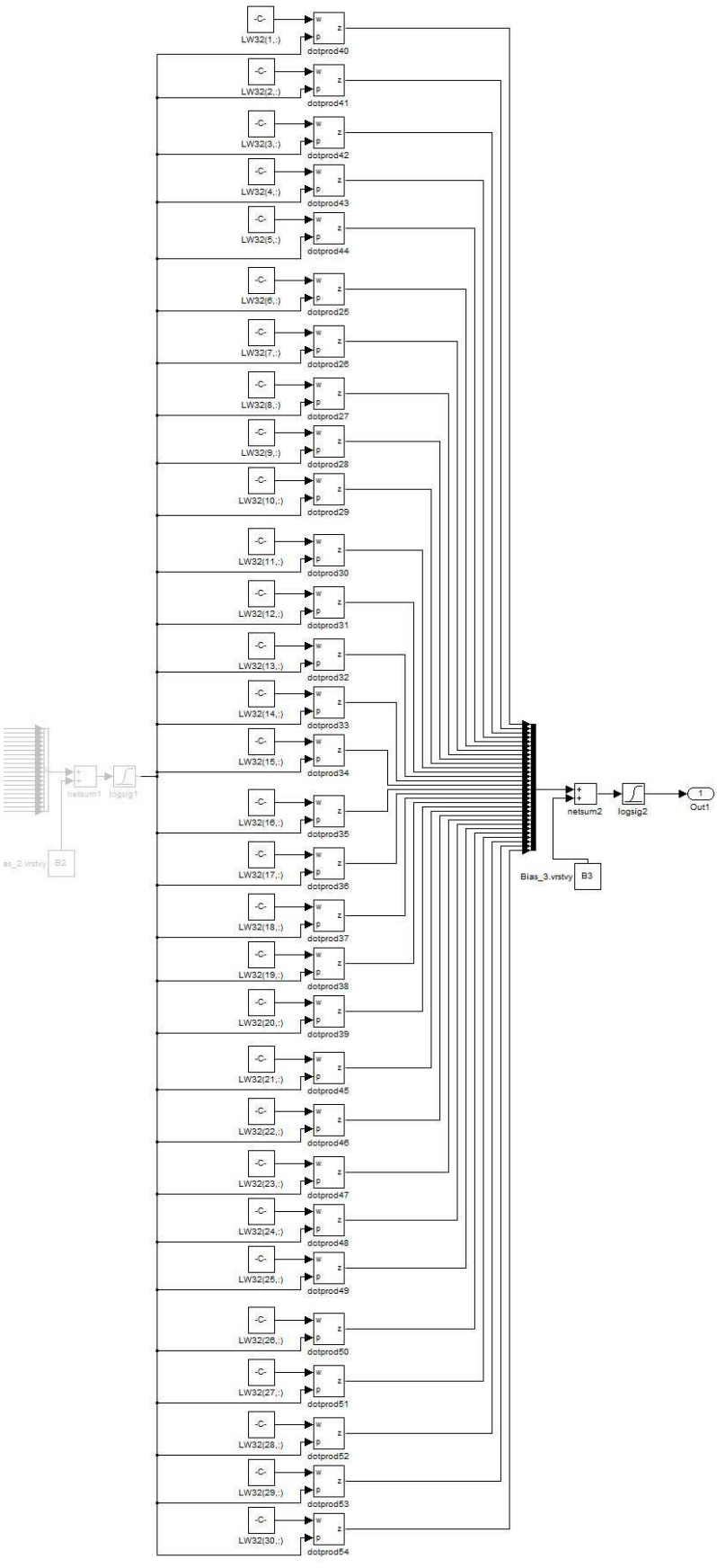
C VNITŘNÍ ZAPOJENÍ BLOKU ŠIFRÁTOR



Obr. C.1: 1.vrstva šifrátoru

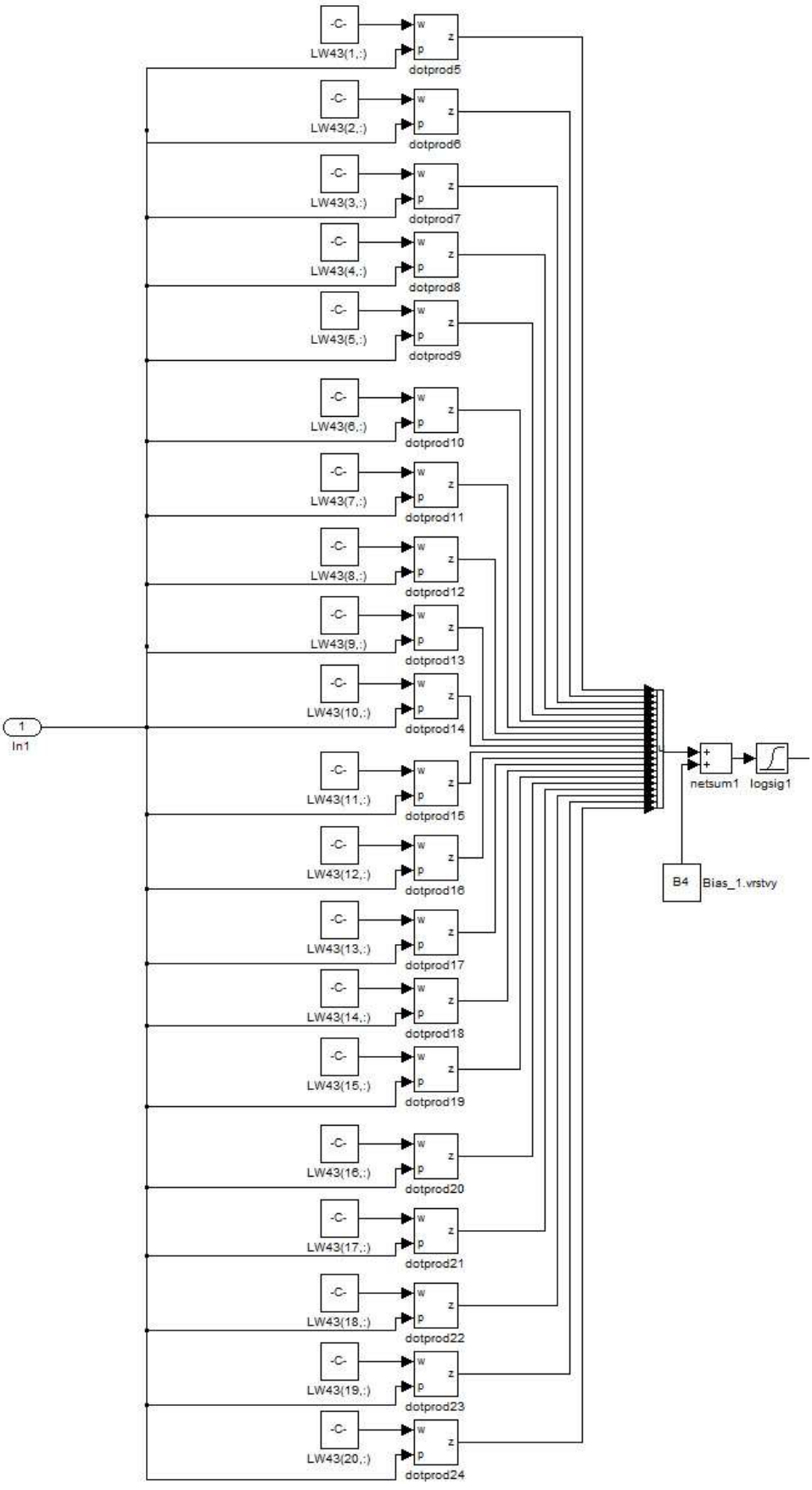


Obr. C.2: 2.vrstva šifrátoru

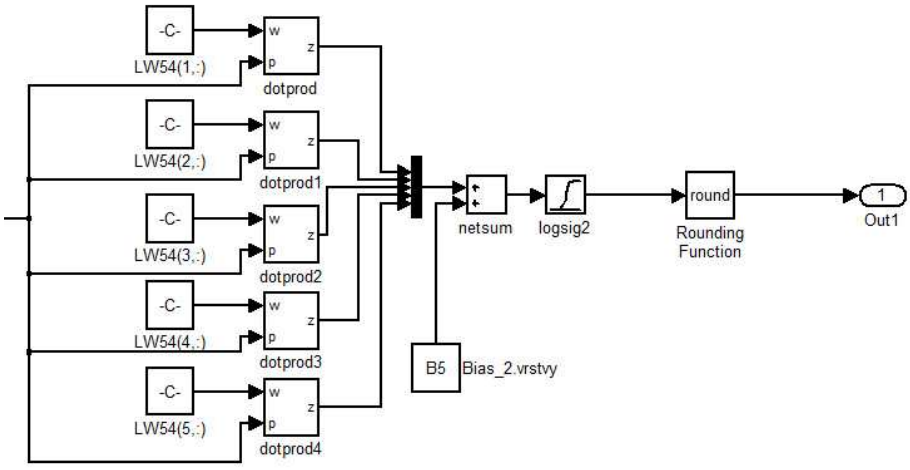


Obr. C.3: 3.vrstva šifrátoru

D VNITŘNÍ ZAPOJENÍ BLOKU DEŠIFRÁTOR



Obr. D.1: 1.vrstva dešifrátoru



Obr. D.2: 2.vrstva dešifrátoru

E ZDROJOVÝ KÓD PROGRAMU START.M

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Diplomova prace
% Program pro sifrovani a desifrovani textu pomoci neuronove site
% Vytvořil:
% Bc. Lavicky Vojtech
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
clc;
%===========================================
%Treninkova data
P = [   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
        1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0;
        1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1;
        1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1;
        0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0];
%
T = [   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0;
        1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0;
        1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1;
        1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1;
        0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0];
%
%===========================================
%Vytvoreni nove site
net = newff(minmax(P),[5,20,30,20,5],{'logsig','logsig','logsig',
'logsig','logsig'},'traingdx');
%===========================================
%
clc;
display('-----------------------------------------')
display('Probiha uceni site')
display('-----------------------------------------')
%===========================================
%Parametry uceni
net.trainParam.epochs = 2000;
```

```
net.trainParam.goal = 1e-4;
net.trainParam.lr = 0.3;
%=========================================
%
%=========================================
%Inicializace vahovych koeficientu a biasu site
net = init(net);
%=========================================
%
%=========================================
%Trenovani site
[net]=train(net,P,T);
%=========================================
%
%=========================================
%Export vektoru vah a biasu vrstev
IW11=net.IW{1,1};
LW21=net.LW{2,1};
LW32=net.LW{3,2};
LW43=net.LW{4,3};
LW54=net.LW{5,4};
B1=net.b{1};
B2=net.b{2};
B3=net.b{3};
B4=net.b{4};
B5=net.b{5};
%=========================================
%
%=========================================
clc;
display('-----------------------------------------')
display('Pro spusteni modelu sifratoru stikni ENTER')
display('-----------------------------------------')
pause
%=========================================
%Spusteni modelu
sif_desif
%=========================================
%
```

```
%==========================================
%Cast programu pro zadani pozadovaneho slova
clc;
clearvars -except v1 v2 net P T IW11 LW21 LW32 LW43 LW54 B1 B2 B3 B4 B5
%
%==========================================
%Nacteni slova z konzole MATLABU
display('----------------------------------------')
text= input('Zadej text a stiskni ENTER: ','s');
display('----------------------------------------')
display('Vstupni slovo prijato!')
display(text)
pause(2)
%==========================================
%
%==========================================
%Cyklus pro sestaveni vstupni matice
textlength = length(text);
input=zeros(5,textlength);
cas=0:1:100;
for ii=1:textlength;
   if text(ii)=='a'
       a=[1 1 1 1 0];
       input(:,ii)=a;
   elseif text(ii)=='b'
       b=[1 1 1 0 1];
       input(:,ii)=b;
   elseif text(ii)=='c'
       c=[1 1 1 0 0];
       input(:,ii)=c;
   elseif text(ii)=='d'
       d=[1 1 0 1 1];
       input(:,ii)=d;
   elseif text(ii)=='e'
       e=[1 1 0 1 0];
       input(:,ii)=e;
   elseif text(ii)=='f'
       f=[1 1 0 0 1];
       input(:,ii)=f;
```

```
elseif text(ii)=='g'
    g=[1 1 0 0 0];
    input(:,ii)=g;
elseif text(ii)=='h'
    h=[1 0 1 1 1];
    input(:,ii)=h;
elseif text(ii)=='i'
    i=[1 0 1 1 0];
    input(:,ii)=i;
elseif text(ii)=='j'
    j=[1 0 1 0 1];
    input(:,ii)=j;
elseif text(ii)=='k'
    k=[1 0 1 0 0];
    input(:,ii)=k;
elseif text(ii)=='l'
    l=[1 0 0 1 1];
    input(:,ii)=l;
elseif text(ii)=='m'
    m=[1 0 0 1 0];
    input(:,ii)=m;
elseif text(ii)=='n'
    n=[1 0 0 0 1];
    input(:,ii)=n;
elseif text(ii)=='o'
    o=[1 0 0 0 0];
    input(:,ii)=o;
elseif text(ii)=='p'
    p=[0 1 1 1 1];
    input(:,ii)=p;
elseif text(ii)=='q'
    q=[0 1 1 1 0];
    input(:,ii)=q;
elseif text(ii)=='r'
    r=[0 1 1 0 1];
    input(:,ii)=r;
elseif text(ii)=='s'
    s=[0 1 1 0 0];
    input(:,ii)=s;
```

```
   elseif text(ii)=='t'
       t=[0 1 0 1 1];
       input(:,ii)=t;
   elseif text(ii)=='u'
       u=[0 1 0 1 0];
       input(:,ii)=u;
   elseif text(ii)=='v'
       v=[0 1 0 0 1];
       input(:,ii)=v;
   elseif text(ii)=='w'
       w=[0 1 0 0 0];
       input(:,ii)=w;
   elseif text(ii)=='x'
       x=[0 0 1 1 1];
       input(:,ii)=x;
   elseif text(ii)=='y'
       y=[0 0 1 1 0];
       input(:,ii)=y;
   elseif text(ii)=='z'
       z=[0 0 1 0 1];
       input(:,ii)=z;
   end
end
%=========================================
%
%=========================================
%Cast spusteni simulace
clc;
display('-----------------------------------------')
display('Pro spusteni simulace stikni ENTER')
display('-----------------------------------------')
pause
sim sif_desif
%=========================================
%
%=========================================
%Vypsani vstupnich dat a kryptogramu do konzole
clc;
display('-----------------------------------------')
```

```
display('Vstupni vektor je:')
display('----------------------------------------')
text
input
display('----------------------------------------')
display('Kryptogram je:')
display('----------------------------------------')
kryptogram(1,:)=[ ];
kryptogram
yout(1,:)=[ ];
%==========================================
%
%==========================================
%Prevedeni vystupu site zpet na text
[rows,columns]=size(yout);
for jj=1:rows;
   if yout(jj,:)==[1 1 1 1 0];
       textout(jj)='a';
   elseif yout(jj,:)==[1 1 1 0 1];
       textout(jj)='b';
   elseif yout(jj,:)==[1 1 1 0 0];
       textout(jj)='c';
   elseif yout(jj,:)==[1 1 0 1 1];
       textout(jj)='d';
   elseif yout(jj,:)==[1 1 0 1 0];
       textout(jj)='e';
   elseif yout(jj,:)==[1 1 0 0 1];
       textout(jj)='f';
   elseif yout(jj,:)==[1 1 0 0 0];
       textout(jj)='g';
   elseif yout(jj,:)==[1 0 1 1 1];
       textout(jj)='h';
   elseif yout(jj,:)==[1 0 1 1 0];
       textout(jj)='i';
   elseif yout(jj,:)==[1 0 1 0 1];
       textout(jj)='j';
   elseif yout(jj,:)==[1 0 1 0 0];
       textout(jj)='k';
   elseif yout(jj,:)==[1 0 0 1 1];
```

```
        textout(jj)='l';
    elseif yout(jj,:)==[1 0 0 1 0];
        textout(jj)='m';
    elseif yout(jj,:)==[1 0 0 0 1];
        textout(jj)='n';
    elseif yout(jj,:)==[1 0 0 0 0];
        textout(jj)='o';
    elseif yout(jj,:)==[0 1 1 1 1];
        textout(jj)='p';
    elseif yout(jj,:)==[0 1 1 0 1];
        textout(jj)='r';
    elseif yout(jj,:)==[0 1 1 0 0];
        textout(jj)='s';
    elseif yout(jj,:)==[0 1 0 1 1];
        textout(jj)='t';
    elseif yout(jj,:)==[0 1 0 1 0];
        textout(jj)='u';
    elseif yout(jj,:)==[0 1 0 0 1];
        textout(jj)='v';
    elseif yout(jj,:)==[0 1 0 0 0];
        textout(jj)='w';
    elseif yout(jj,:)==[0 0 1 1 1];
        textout(jj)='x';
    elseif yout(jj,:)==[0 0 1 1 0];
        textout(jj)='y';
    elseif yout(jj,:)==[0 0 1 0 1];
        textout(jj)='z';
    end
end
%=========================================
%
%=========================================
%Zobrazeni desifrovane zpravy do konzole
display('-----------------------------------------')
display('Desifrovana zprava je:')
display('-----------------------------------------')
yout'
textout
%=========================================
```