



**BRNO UNIVERSITY OF TECHNOLOGY**  
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



**FACULTY OF INFORMATION TECHNOLOGY**  
**DEPARTMENT OF COMPUTER GRAPHICS**  
**AND MULTIMEDIA**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**

## **ACTIVITY OF NEURAL NETWORK IN HIDDEN LAYERS – VISUALISATION AND ANALYSIS**

**ZOBRAZENÍ A ANALÝZA AKTIVIT NEURONOVÉ SÍTĚ VE SKRYTÝCH VRSTVÁCH**

**MASTER'S THESIS**

DIPLOMOVÁ PRÁCE

**AUTHOR**

AUTOR PRÁCE

**Bc. MARKO FÁBRY**

**SUPERVISOR**

VEDOUČÍ PRÁCE

**Ing. MARTIN KARAFIÁT, Ph.D.**

BRNO 2016

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií

Akademický rok 2015/2016

**Zadání diplomové práce**

Řešitel: **Fábry Marko, Bc.**

Obor: Počítačová grafika a multimédia

Téma: **Zobrazení a analýza aktivit neuronové sítě ve skrytých vrstvách  
Activity of Neural Network in Hidden Layers - Visualisation and  
Analysis**

Kategorie: Zpracování řeči a přirozeného jazyka

Pokyny:

1. Seznamte se s problematikou neuronových sítí a s jejich použitím v oblasti automatického přepisu řeči.
2. Použijte standardní trénovací skripty pro natrénování systému založeném na neuronové síti.
3. Kód analyzujte a upravte tak aby bylo možné zobrazit jednotlivé aktivace.
4. Aktivace zobrazte a analyzujte pro různá prostředí a typy neuronové sítě.

Literatura:

- Geoffrey Hinton, Li Deng, Dong Yu etc., **Deep Neural Networks for Acoustic Modeling in Speech Recognition**, *IEEE Signal Processing Magazine* (2012)
- L.J.P. van der Maaten and G.E. Hinton, **Visualizing High-Dimensional Data Using t-SNE**, *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008.
- Y. LeCun, Y. Bengio and G.E. Hinton, **Deep Learning**, *Nature* (May 2015): Vol. 521, pp 436-444.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních 2 bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese

<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

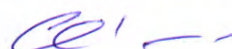
Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Karafiát Martin, Ing., Ph.D.**, UPGM FIT VUT

Datum zadání: 1. listopadu 2015

Datum odevzdání: 25. května 2016

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
Fakulta informačních technologií  
Ústav počítačové grafiky a multimédií  
602 00 Brno, Božetěchova 2



---

doc. Dr. Ing. Jan Černocký  
vedoucí ústavu

## Abstract

Goal of this work was to create system capable of visualisation of activation function values, which were produced by neurons placed in hidden layers of neural networks used for speech recognition. In this work are also described experiments comparing methods for visualisation, visualisations of neural networks with different architectures and neural networks trained with different types of input data. Visualisation system implemented in this work is based on previous work of Mr. Khe Chai Sim and extended with new methods of data normalization. Kaldi toolkit was used for neural network training data preparation. CNTK framework was used for neural network training. Core of this work — the visualisation system was implemented in scripting language Python.

## Abstrakt

Cílem této práce je vytvořit systém schopný zobrazení hodnot aktivačních funkcí neuronů nacházejících se v skrytých vrstvách neuronových sítí použitých na rozpoznávání řeči. Dále byly na tomto systému provedeny experimenty porovnávající vizualizační metody, vizualizace neuronových sítí s různými architekturami a s různými druhy vstupních dat. Vizualizační systém implementovaný v rámci této práce je založen na předchozí práci pana Khe Chai Sim a rozšířen o nové způsoby normalizace vstupních dat. Pro přípravu trénovacích dat neuronových sítí byl použit framework Kaldi. Pro samotné trénování neuronových sítí byl použit nový framework CNTK. Jádro práce — samotný vizualizační systém byl implementován v skriptovacím jazyce Python.

## Keywords

deep neural network, hidden layer, speech recognition, visualisation, analysis, activations, t-SNE, Kaldi, CNTK

## Klíčová slova

hluboká neuronová síť, skrytá vrstva, rozpoznávání řeči, vizualizace, analýza, aktivace, t-SNE, Kaldi, CNTK

## Reference

FÁBRY, Marko. *Activity of Neural Network in Hidden Layers – Visualisation and Analysis*. Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Karafiát Martin.

# Activity of Neural Network in Hidden Layers – Visualisation and Analysis

## Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Martin Karafiát Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....  
Marko Fábry  
25th May 2016

## Acknowledgements

I would like to thank to Ing. Martin Karafiát, Ph.D., for his professional help and especially for encouraging me in difficult times. I would also like to thank to my friends and family, for mentally and materialy supporting me while I was writing this thesis.

I also greatly appreciate access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme „Projects of Projects of Large Research, Development, and Innovations Infrastructures“ (CESNET LM2015042).

© Marko Fábry, 2016.

*This thesis was created as a school work at the Brno University of Technology, Faculty of Information Technology. The thesis is protected by copyright law and its use without author's explicit consent is illegal, except for cases defined by law.*

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Neural networks</b>	<b>4</b>
2.1	Biological neuron . . . . .	4
2.2	Artificial neuron . . . . .	5
2.2.1	General model . . . . .	5
2.2.2	Types of units . . . . .	5
2.3	Neural Network . . . . .	7
2.3.1	Deep Neural Network . . . . .	8
2.3.2	Softmax output layer . . . . .	8
2.4	Learning . . . . .	8
2.4.1	Error function . . . . .	9
2.4.2	Gradient descent . . . . .	9
2.4.3	Backpropagation . . . . .	10
2.5	Specific types of Neural Networks . . . . .	11
2.5.1	Bottleneck Neural Networks . . . . .	11
2.5.2	Recurrent Neural Networks . . . . .	11
2.5.3	Long Short-Term Memory Neural Networks . . . . .	12
<b>3</b>	<b>Automatic Speech Recognition</b>	<b>13</b>
3.1	Automatic speech recognition system . . . . .	13
3.1.1	Feature extraction . . . . .	13
3.1.2	Acoustic modelling . . . . .	14
3.1.3	Acoustic matching . . . . .	14
3.1.4	Language model . . . . .	14
3.1.5	Speech decoding . . . . .	14
3.2	Deep Neural Networks in ASR . . . . .	15
<b>4</b>	<b>High dimensional data projection</b>	<b>16</b>
4.1	Dimension reduction . . . . .	16
4.2	Stochastic Neighbour Embedding (SNE) . . . . .	16
4.3	Crowding problem . . . . .	17
4.4	t-Distributed Stochastic Neighbour Embedding (t-SNE) . . . . .	17
<b>5</b>	<b>Implementation of visualisation system</b>	<b>19</b>
5.1	Data preparation . . . . .	19
5.2	Neural Network training . . . . .	19
5.2.1	Computational Network Toolkit (CNTK) . . . . .	20

5.3	Activation extraction . . . . .	20
5.4	Activation visualization . . . . .	20
5.4.1	Activity vectors . . . . .	21
5.4.2	Normalised entropy . . . . .	21
5.4.3	Hidden activity space . . . . .	22
5.4.4	Delaunay triangulation . . . . .	23
5.4.5	Ranking vectors . . . . .	24
5.4.6	Interpretable activity regions . . . . .	24
5.4.7	Interpretable regions in 3D . . . . .	26
5.4.8	Activity visualisation . . . . .	26
<b>6</b>	<b>Experiments</b>	<b>28</b>
6.1	Dataset description . . . . .	28
6.2	Compensating dataset statistical characteristics . . . . .	29
6.2.1	Attribute occurrence normalisation . . . . .	29
6.2.2	Silence trimming . . . . .	29
6.3	Experimental set-ups . . . . .	30
6.3.1	Experimental architectures . . . . .	30
6.3.2	Experimental environments . . . . .	30
6.4	Performance of the analysed networks . . . . .	31
6.5	Comparison of different views on experimental data . . . . .	32
6.6	Comparison of visualisation methods . . . . .	35
6.7	Analysis of NNs with different architectures . . . . .	37
6.8	Analysis of NNs trained with different datasets . . . . .	40
6.9	Experiments epilogue . . . . .	41
<b>7</b>	<b>Conclusion and future plans</b>	<b>42</b>
	<b>Bibliography</b>	<b>44</b>
	<b>Appendices</b>	<b>46</b>
	List of Appendices . . . . .	47
<b>A</b>	<b>Contents of DVD</b>	<b>48</b>
<b>B</b>	<b>Example scripts for CNTK</b>	<b>49</b>
B.1	Activation printing script . . . . .	49

# Chapter 1

## Introduction

Current state of the art methods in Automatic Speech Recognition and in many other fields, for example computer vision, stock market prediction or scheduling optimization are using artificial neural networks.

Artificial neural networks are mathematical models inspired by principles of biological nervous systems, in particular human brain. Although neural networks are widely and frequently used and their models and learning algorithms are mathematically well defined, they are used essentially as *black boxes*, without much understanding of their inner workings. Reason behind this is that practically used neural networks have large number of learnable parameters, therefore their structure easily becomes very complex.

This thesis is mainly based on the work of Khe Chai Sim and it aims to recreate and extend experiments described in his recent article [16]. Motivation for this work is in visualisation of artificial neuron activations in similar fashion to methods used in human brain research, like positron emission tomography (PET) scan, which shows how the human brain and its parts are working.

First part of this text provides basic theoretical knowledge and overview of technologies used in current research of neural networks and automatic speech recognition. This part of the thesis was done in the term project. The second part describes implementation of system capable of visualising different neural network architectures. Subsequently, experiments performed using this system on selected architectures are described and analysed.

This work is segmented in chapters as follows. Chapter **2 — Neural networks** describes basic structural units and learning principles of neural networks. There are also described particular types of neural networks used later in experimental part of this work. Chapter **3 — Automatic Speech Recognition** deals with the usage of neural networks in automatic speech recognition. In chapter **4 — High dimensional data projection** are provided basic informations about the Stochastic Neighbour Embedding, method selected for high-dimensional data visualization. Chapter **5 — Implementation of visualisation system** describes implemented system for experiments on neural networks and their usage in speech recognition. This system covers conversion of sound files to acoustic features, training neural network on these features, printing the activation values of hidden layers of the neural network and — the main concern of this work — visualizing the result. Consequently, multiple experiments conducted with different types of neural networks are described in chapter **6 — Experiments**. Finally, chapter **7 — Conclusion and future plans** contains overall review of this diploma thesis, results of conducted experiments and suggestions for possible future work.

## Chapter 2

# Neural networks

This chapter explains origins and basic principles of neurons, neural networks and learning. Content of this chapter is mainly based on publication [2], unless otherwise stated.

### 2.1 Biological neuron

Biological neuron (see figure 2.1) consists of main body named *soma*, which has many “input” endings called *dendrites* and one very long projection – *axon*, which splits into multiple *synaptic terminals*. Synaptic terminals are “output” endings of neuron connected to dendrites of other neurons. Neuron “fires”, when there is sufficient amount of electrical impulses from neuron’s dendrites that is greater than threshold. It means that the neuron propagates electrical signal through his axon and synaptic terminals to following connected neurons. The neuron becomes numb for a short period of time after propagation.

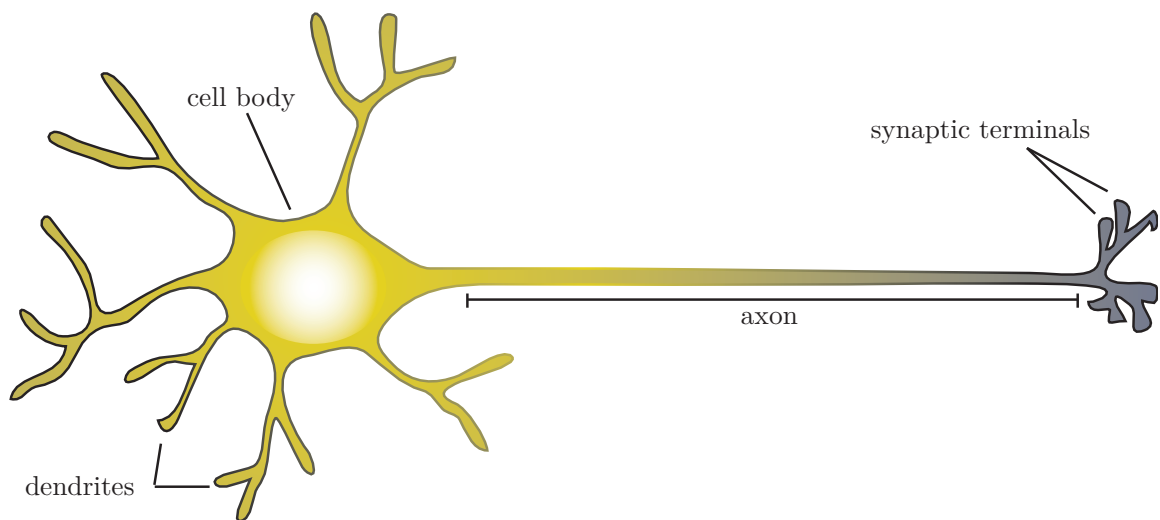


Figure 2.1: Biological neuron<sup>1</sup>.

---

<sup>1</sup>The picture is based on <http://bio3520.nicerweb.com/Locked/chap/ch03/neuron.html>



## 2.2 Artificial neuron

### 2.2.1 General model

An artificial neuron or *neural network unit* is basic building block of neural networks. It transforms vector of inputs  $\mathbf{x} = (x_1, x_2, \dots, x_d)$  to single output  $\mathbf{y}$  using composition of two functions:

1. A *net value function*  $\xi$ , which computes *net value*  $v$  using neurons inputs  $\mathbf{x}$  and their respective weights  $\mathbf{w}$ :

$$v = \xi(x, w) \tag{2.1}$$

Mostly weighted sum or some kind of a vector distance function is used.

2. An *activation function*  $\phi$ , which computes neurons output  $\mathbf{y}$  using *net value*  $v$ :

$$y = \phi(v) \tag{2.2}$$

Resulting output value is then copied to all following neurons. There are many types of activation functions. The most used, with regards to speech recognition, will be described in following sections.

Artificial neuron was inspired by the biological neuron. The similarity between artificial and biological neurons can be seen in previous function definitions. The net value function aggregates weighted input values, like biological neuron aggregates electrical impulses. Consequently, according to this net value, activation function decides intensity, with which the neuron will fire (or inhibit input signals).

### 2.2.2 Types of units

Because of many possible combination of suitable net value and activation functions, there are multiple types of neural network units. In following sections are briefly described important currently used units. Content of this section is inspired by the book [2].

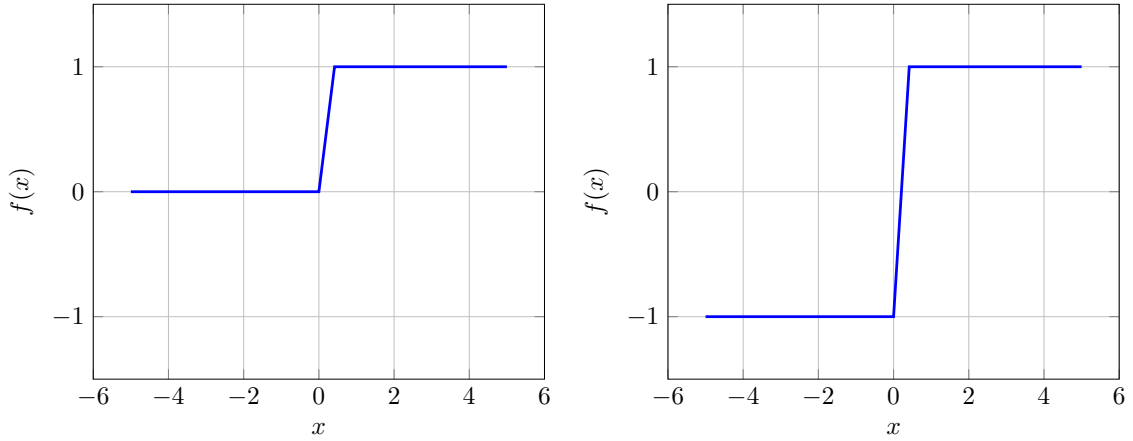
#### Linear threshold unit

Linear threshold unit is sometimes incorrectly called *perceptron*, after single layer neural network in which it was originally used. It is considered to be first artificial neuron. Perceptron was designed by Frank Rosenblatt in 1957 [13], based on mathematical model provided by McCulloch and Pitts [10].

Linear threshold unit uses linear net value function:

$$v = \sum_i^m (w_i * x_i) \tag{2.3}$$

The step function (see figure 2.2a) or the sign function (see figure 2.2b) is used as the activation function. Output of the unit depends on its property  $b$  called *bias*. Bias is negative threshold independent of any input variables. When weighted sum of inputs is greater than  $|b|$ , linear threshold unit produces value 1 on its output - the neuron fires, otherwise it produces 0 (for step function) or  $-1$  (for sign function) - it inhibits inputs.



$$f(x) = \begin{cases} 1 & \text{if } v \geq b \\ 0 & \text{otherwise} \end{cases}$$

(a) Step activation function (for bias  $b = 0$ ).

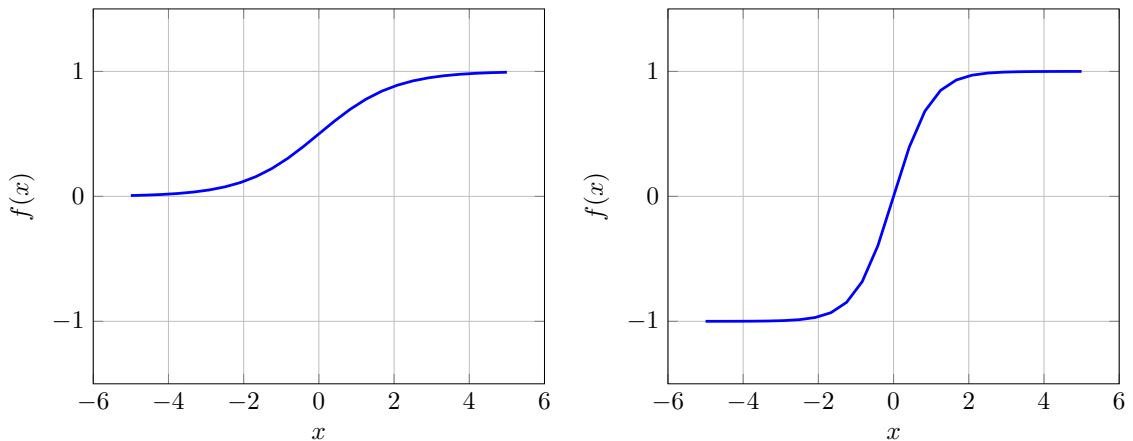
$$f(x) = \begin{cases} 1 & \text{if } v \geq b \\ -1 & \text{otherwise} \end{cases}$$

(b) Sign activation function (for bias  $b = 0$ ).

Figure 2.2: Activation functions of linear threshold unit.

### Sigmoid unit

*Sigmoid neuron* uses same net value function as the linear threshold unit - weighted sum of inputs. Sigmoid function (sometimes called *logistic function*) is used as the activation function (see figure 2.3a). This function has the output in range (0; 1), which is suitable for representing probabilities of classified samples to belong to certain class. Other alternative is hyperbolic tangent function (see figure 2.3b). Both functions have continuous output range. It enables them to be used in neural networks that are using some variation of gradient descent learning described later in text.



$$f(x) = \frac{1}{(1 + e^{-x})}$$

(a) Sigmoid function.

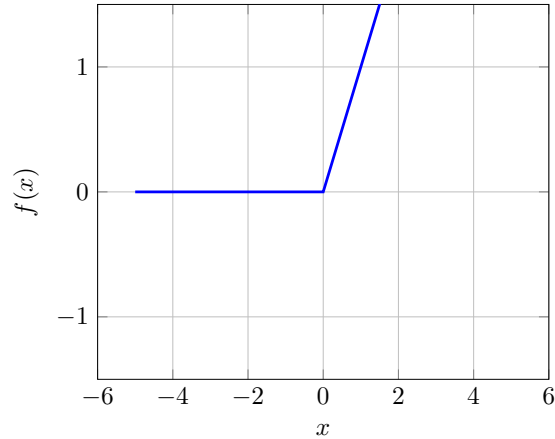
$$f(x) = \tanh(x)$$

(b) Hyperbolic tangent function.

Figure 2.3: Activation functions of sigmoid unit.

## Rectified linear unit (ReLU)

*Rectified linear unit (ReLU)* uses linear net value function similarly to functions described above. Rectified linear function is used as the activation function (see figure 2.4). These units are becoming more popular than the sigmoid ones, because they have far lower computational complexity - practically only instruction needed comparison with zero.



$$f(x) = \max(0, x)$$

Figure 2.4: Activation function of rectified linear unit.

## 2.3 Neural Network

*Neural Networks (NNs)* are considered as universal functions approximators from the mathematical point of view. Given the sufficient number of parameters, neural networks can approximate any given vector space mapping, therefore NNs should be capable of performing any classification task. The most basic type of neural network is *feed-forward* neural network. It is acyclic oriented graph with one *output layer* and at least one *hidden layer*. Nodes in these layers are neurons and edges are outputs of neurons of the previous layer and at the same time inputs to neurons of the next layer [8]. Every edge has assigned its respective weight. There is convention, that inputs are also considered as layer, even though this *input layer* contains no neurons. Example of feed-forward neural network is depicted in figure 2.5.

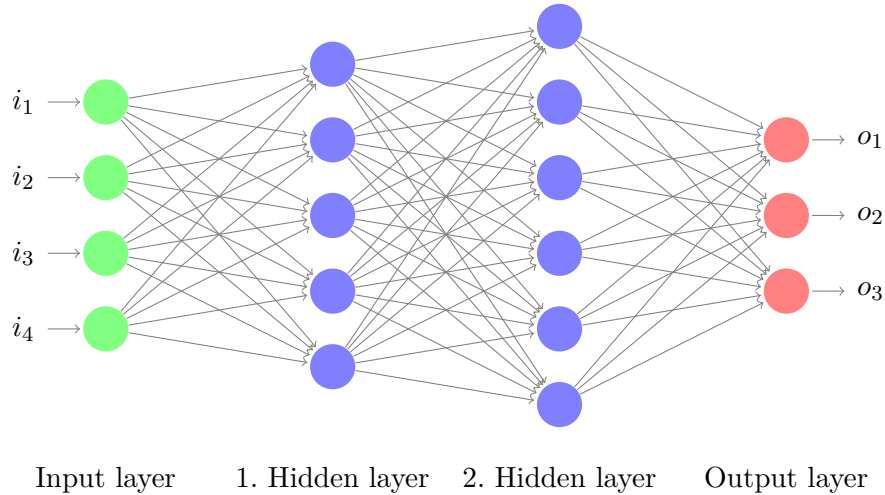


Figure 2.5: Neural network example.

[Inspired by <http://www.texample.net/tikz/examples/neural-network/>.]

### 2.3.1 Deep Neural Network

*Deep Neural Networks* (DNNs) is designation for neural network that has several hidden layers (often containing multiple different types of units), as opposed to *shallow neural network*, which has usually only one hidden layer [5].

### 2.3.2 Softmax output layer

For multi-class classification tasks, it is usually desirable to normalize neural network classification results. *Softmax* function (see equation (2.4)) is usually used for this purpose. It normalizes the  $j$ -th neuron's output, with respect to all outputs of neurons  $x_0, x_1, \dots, x_k$  in output layer. Softmax output layer is different from previously mentioned feed forward layers, because its neurons are fully interconnected.

$$p_j = \frac{e^{x_j}}{\sum_k (e^{x_k})} \quad (2.4)$$

## 2.4 Learning

Learning in biological neural networks is based on strengthening of the synapses which are connections between neurons. In artificial neural networks, these connections are represented with weights of unit inputs. Because learning can be defined as making less mistakes in given task, the amount of weights which are strengthened (or weakened) is based on number of correctly (or incorrectly) classified samples. Content of this section originates in publications [2, 8, 5]. Content of section 2.4.3 is mainly inspired by publications [17, 11].

### 2.4.1 Error function

*Error function*  $\mathcal{E}(\mathbf{w})$  is function that represents quality of neural network. It is used in learning with teacher to determine ratio of incorrect outputs. The main goal of learning is to minimize this objective function. It is defined in equation (2.5), as expectation of error metric  $E(\mathbf{w}; \mathbf{x})$ , between desired output vector  $\mathbf{t} = (t_1, t_2, \dots, t_n)$  and real neural network output vector  $\mathbf{o} = (o_1, o_2, \dots, o_n)$  for given input vector  $\mathbf{x} = (x_1, x_2, \dots, x_m)$ .

$$\mathcal{E}(\mathbf{w}) = \frac{1}{N} \sum_{\mathbf{x} \in X} E(\mathbf{x}; \mathbf{w}) \quad (2.5)$$

Currently, two of the most used examples of error metrics are *Mean Square Error* (MSE) (see equation (2.6)) and *Cross-Entropy* (CE) (see equation (2.7)). The MSE metric is mainly used in pattern learning applications, whereas the CE metric is mostly used in classification tasks.

$$E_{MSE}(\mathbf{x}; \mathbf{w}) = \sum_{k=0}^K (o_k - t_k)^2 \quad (2.6)$$

$$E_{CE}(\mathbf{x}; \mathbf{w}) = - \sum_{k=1}^K t_k \ln(o_k) \quad (2.7)$$

### 2.4.2 Gradient descent

*Gradient descent* is an algorithm for minimization of functions with multiple input variables. The main idea behind gradient descent is to initialize input variables to random values at the start and keep changing them until we reach point where difference between two consequent output values are acceptably small. Values are changed in respect to negative *gradient* which is the vector of first derivations of the minimized function (see equation (2.8)). Condition to gradient descent method is, that it requires function to be defined and differentiable for all possible input variables in order to find the derivatives.

$$\nabla f(x) = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]^T \quad (2.8)$$

One step of the gradient descent algorithm can be represented with equation (2.9):

$$x(t+1) = x(t) - \mu \nabla f(x(t)), \quad (2.9)$$

where  $x(t)$  is the vector of inputs in current step  $t$  of the algorithm,  $\nabla f(x(t))$  is gradient of function  $f$  in point  $x(t)$ ,  $\mu$  is *learning coefficient* and  $x(t+1)$  is the resulting vector of inputs.

Learning coefficient  $\mu$  represents *step size* of the algorithm. If it is too small, the learning is too slow. On the other hand, if it is too big, it is possible, that the algorithm misses the desired minimum. The fact that learning coefficient can vary between steps is often used in methods determining suitable learning coefficient dynamically [5].

Disadvantage of gradient descent is, that for non-convex functions the algorithm can get stuck in local minimums. Another disadvantage is that the resulting minimum is dependent on selection of starting points.

For large training sets, it is usually more convenient to generate so called *mini-batches* — batches of randomly selected training vectors, which represent whole training set. During learning phase, weights are not updated after every training vector but after every mini-batch and average gradient computed of the mini-batch is used for the update. This method speeds up the learning process and it is called *stochastic gradient descent* (SGD) [5].

### 2.4.3 Backpropagation

Learning in DNNs consists of two phases:

1. *Forward phase*: The neural network inputs are in this phase propagated forward, as they would in the case of classification. Output values of the classification are compared to expected values and an **error value** is computed based on chosen error metric.
2. *Backpropagation phase*: Error values are propagated in backward direction from outputs to inputs throughout the network layers. The gradients at the previous layers are learned as a function of the errors and weights in the layer ahead.

Every weight of nodes in neural network is updated according to equation (2.10), which is clearly based on gradient descent method described in section 2.4.2. If the minimized function is error metric  $E$  and its input is vector  $\mathbf{w}_{ji}$ , it can be deduced:

$$\begin{aligned}\Delta^{(l)}w_{ji} &= -\mu \frac{\partial E}{\partial^{(l)}w_{ji}} \\ \Delta^{(l)}w_{ji} &= -\mu \frac{\partial E}{\partial^{(l)}v_j} \frac{\partial^{(l)}v_j}{\partial^{(l)}w_{ji}} \\ \Delta^{(l)}w_{ji} &= -\mu \frac{\partial E}{\partial^{(l)}y_j} \frac{\partial^{(l)}y_j}{\partial^{(l)}v_j} \frac{\partial^{(l)}v_j}{\partial^{(l)}w_{ji}} \\ \Delta^{(l)}w_{ji} &= \mu^{(l)}\delta_j^{(l)}x_i\end{aligned}\tag{2.10}$$

where:

$$^{(l)}\delta_j = -\frac{\partial E}{\partial^{(l)}y_j} \frac{\partial^{(l)}y_j}{\partial^{(l)}v_j}\tag{2.11}$$

and:

$$^{(l)}x_i = \frac{\partial^{(l)}v_j}{\partial^{(l)}w_{ji}} \quad (\text{note: see equation (2.1)})\tag{2.12}$$

where  $\Delta^{(l)}w_{ji}$  is the *error derivative* - value by which the  $i$  - th input of the  $j$  - th neuron in layer  $l$  will be updated,  $^{(l)}\mathbf{x}$  is the input vector of layer  $l$ ,  $^{(l)}\mathbf{y}$  is the output vector of layer  $l$ ,  $\mu$  is the learning rate,  $^{(l)}v_j$  is the net value of the  $j$  - th neuron in layer  $l$  and  $^{(l)}\delta_j$  which is the derivation of error metric and activation function and it is typically called the *credit* of the neuron.

Backpropagation starts in the **output layer**  $L$ , where the error derivative  $\Delta^{(L)}w_{ji}$  is computed by differentiating **the error metric function  $E$  with respect to output**

$y$  ( $\frac{\partial E}{\partial^{(L)}y_j}$ ) and **the output of the activation function  $y$  with respect to output of the net value function  $v$**  ( $\frac{\partial^{(L)}y_j}{\partial^{(L)}v_j}$ ), multiplied by inputs  $^{(L)}\mathbf{x}_i$  and learning rate  $\mu$  (see equation (2.13)).

$$\Delta^{(L)}w_{ji} = \mu \frac{\partial E}{\partial^{(L)}y_j} \frac{\partial^{(L)}y_j}{\partial^{(L)}v_j} x_i \quad (2.13)$$

In every **hidden layer** is the error derivative computed as **weighted sum of credits** of the following layer with respect to the total inputs to the units in layer above  $^{(l+1)}\delta_j$ , multiplied by **the output of the activation function  $y$  with respect to output of the net value function  $v$**  of the current layer ( $\frac{\partial^{(l)}y_j}{\partial^{(l)}v_j}$ ), multiplied by inputs  $^{(l)}\mathbf{x}_i$  and learning rate  $\mu$  (see equation (2.14)).

$$\Delta^{(l)}w_{ji} = \mu \sum_{k=1}^n (^{(l+1)}w_{kj} \ ^{(l+1)}\delta_j) \frac{\partial^{(l)}y_j}{\partial^{(l)}v_j} x_i \quad (2.14)$$

## 2.5 Specific types of Neural Networks

This section introduces some of the many specific architectures of neural networks, that are currently researched and used for their particular abilities. Content of this section is inspired by articles [8, 7, 4].

### 2.5.1 Bottleneck Neural Networks

*Bottleneck NNs* have at least one hidden layer called *bottleneck (BN)* with significantly smaller number of units than hidden layers before and after this layer. This results in compression of the information flowing through the bottleneck layer. Since the information is forced to be compressed, the neural network learns to discard unnecessary information and compress useful information. Systems based on bottleneck NNs are generally slightly more resilient against noised data. They are also less prone to the over-fitting problem, which in simple terms means, that the neural network is unable to generalize - to classify different data than it was trained on. Bottleneck NNs are often used for noise removal and compression in image or audio precessing tasks and feature extraction.

### 2.5.2 Recurrent Neural Networks

*Recurrent Neural Networks (RNNs)* are different than standard feed-forward DNNs described in section 2.3, because they allow creating cycles in their topology. This fact allows RNNs to efficiently model sequential processes. The cycle creates feedback connection in neural network, which provides for the RNNs the ability to predict following input values from the previous input values of the sequence. This ability is especially useful in tasks where output does not depend solely on current input value but on arbitrary number of previous values like speech or language. For simplicity, the RNN can be viewed as standard feed forward NN, with added memory cells, which store the internal state of the neural network units. When plain RNNs are trained using modified version of back-propagation algorithm (*Back Propagation Through Time (BPTT)*), they are very prone to vanishing / exploding gradient problem. This problem is caused by the

fact that when trained, one layer of RNN units can be unfolded as  $t$  layers of standard feed-forward DNN, where  $t$  is count of all inputs  $x_{t'}$  up to current input  $x_t$ . In such extremely deep neural network the gradient can easily fade away or start exponentially growing. Currently there exist many modifications of RNN neural networks, but this work concerns only with the most popular type - the Long Short Term Memory NN, which does not suffer from vanishing - exploding gradient problem.

### 2.5.3 Long Short-Term Memory Neural Networks

*Long Short-Term Memory (LSTM)* neural network is a subtype of Recurrent Neural Networks suggested by Hochreiter and Schmidhuber in [7]. LSTM neural networks are composed of complex neural units called *memory blocks*. These blocks with feedback loops are able to store network state information in small memory called *memory cell*. Information flow of memory block is controlled by three standard sigmoid neural units called *gates*. The *input gate* controls amount of information able to enter the LSTM unit. Similarly, the *output gate* controls amount of information proceeding to the next layer of the LSTM network. The last gate – *forget gate* enables the memory block to discard information stored in memory cell, which enables the LSTM network to dynamically adapt to continuous input streams.

As stated above, all these gates are sigmoid units, which have their own weight matrices capable of learning and activation functions determining the amount of information to input, forget or output. Action of the gate is implemented as point-wise multiplication of the gate value and the respective input which gate controls. This effectively solves vanishing / exploding gradient problem from which suffer the RNNs, because the activation functions of the gates, especially the forget gate, prevent the gradient from decreasing / increasing uncontrollably.

It is necessary to note that the input to the LSTM unit before it reaches the input gate is squashed into range  $(-1;1)$  with the *tanh* function. The output of the memory before reaching the output gate is also adjusted with the *tanh* function to range  $(-1;1)$ .

Some practical implementations of LSTM NNs are employing several times more memory cells while maintaining the computational complexity by using special projection layers. These projection layers are reducing the number of recurrent outputs of memory cells. In this way there are more units for data processing, but less recurrent outputs which reduces learning time. This approach is described in more detail in article [15].

This section provided only very brief introduction to the currently popular LSTM neural networks. For more detailed informations about LSTM NNs and their abilities, please see articles [7, 4, 14].



## Chapter 3

# Automatic Speech Recognition

An *Automatic Speech Recognition (ASR)* is a system that converts the speech signal into words. Such systems are used to control digital devices with simple commands or data entering in mobile devices. Current state-of-the-art ASR systems are providing great results on simple tasks, such as recognition of isolated words or read speech. Some practical applications are already usable, but there is still more research needed to improve their performance. Contents of this chapter are inspired by publications [18, 5].

### 3.1 Automatic speech recognition system

Basic scheme of Automatic Speech Recognition system is depicted in figure 3.1. Functions of individual blocks will be briefly described in following sections.

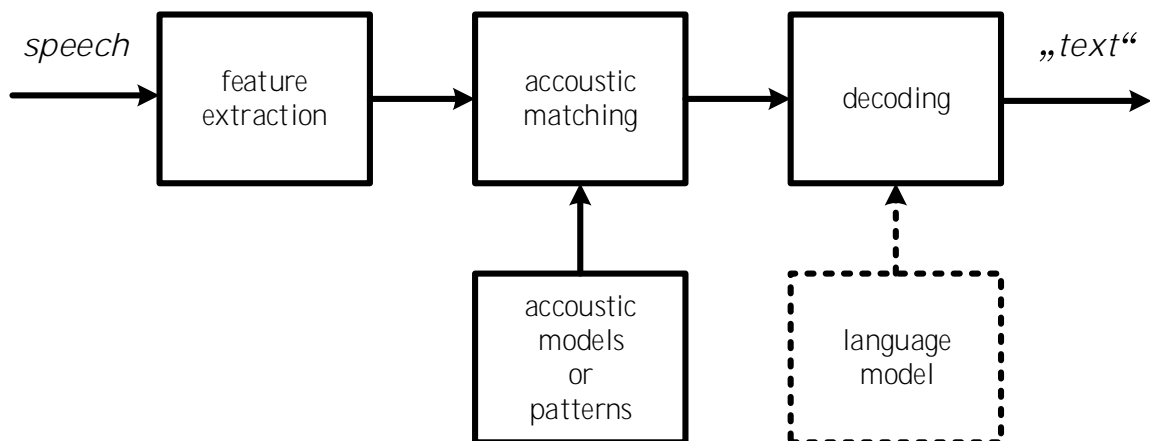


Figure 3.1: Automatic speech recognition system [18].

#### 3.1.1 Feature extraction

The first thing that is needed in order to create speech recognition system, is to extract speech coefficients, sometimes also called *features*, from the input speech recordings. This is done by filtering out unnecessary components of speech signal like DC offset, mean value or pitch to unify the speech and to limit the data to reasonable size. Consequently, the speech is split into smaller overlapping parts called *frames*, in which the speech signal

should be relatively stationary. Frames are commonly processed by the *spectral* or *linear prediction analysis* to compute the *Mel Frequency Cepstral Coefficient (MFCC)* or the *Linear Predictive Coefficients (LPC)* respectively. Vectors of these coefficients are the previously mentioned features. DNNs used for speech recognition are usually trained using *filter-bank* (sometimes called “*f-bank*”) features. These features are created from frames by *Discrete Fourier Transformation (DCT)* and by using bank of filters with usually 41 coefficients (including energy) distributed on a *log mel-scale* [18].

### 3.1.2 Acoustic modelling

Acoustic modelling in speech recognition systems often utilized *Gaussian Mixture Models (GMM)* which are predicting posterior probabilities of *Hidden Markov Model (HMM)* states. In other words, in every state of HMM is statistical distribution represented by mixture of diagonal covariance Gaussians (the GMM), which returns a likelihood for a vector of features. Acoustic model consists of many individual HMMs, each representing one basic acoustic unit – *phoneme* (sometimes also called *phone*). Since phonemes are context dependent, current systems usually use triplets of phonemes - *triphones*. In the triphone, every individual phoneme is in triplet with preceding and following phoneme. This extends number of possible output classes, but it significantly improves overall speech recognition results.

### 3.1.3 Acoustic matching

*Acoustic matching* is the process of scoring incoming feature vectors with individual HMM models. Problem of acoustic matching is spatial and temporal variability. Spatial variability is caused by inability to say same thing twice, exactly the same way - this variability is resolved by using GMMs mentioned in acoustic modelling, which are mapping similar utterances to the same class, respectively to the same state of the HMM. Temporal variability is caused by inability to say same thing twice with exactly same speed. This is resolved by using HMMs, which allow for one state to accept variable number of feature vectors. Outputs of acoustic matching are *alignments*, probabilities with which sequences of input vectors correspond to HMM states representing phonemes.

### 3.1.4 Language model

*Language model (LM)* is probability model created from transcriptions of recordings. It incorporates vocabulary of words and their occurrence probabilities based on their frequencies or possibly some grammar rules. LM is typically word based only. It gives probability of word sequence  $P(W)$ , where probability of the current word  $P(W_i)$  depends on probabilities of  $n$  previous words  $P(W_{i-(n-1)}), P(W_{i-(n-2)}), \dots, P(W_{i-1})$ . Such models are called *n-gram* models [18].

### 3.1.5 Speech decoding

In speech decoding block of the ASR system, phoneme sequences are produced by acoustic matching transformed to symbols, respectively words with respect to language model which can influence output of similarly sounding phrases. For example, phrases “recognize speech” and “wreck a nice beach” have similar phoneme sequence, because they sound very similar, but based on language model, which provides context, correct phrase is selected. Usual

methods for decoding are Viterbi algorithm, A\*-search, best-first decoding, Finite State Transducers and others [18].

## 3.2 Deep Neural Networks in ASR

Still more popular trend in ASR research is using DNNs, mainly because their ability to generalize is much better than with GMM / HMM based systems. In order to generate DNN modelled ASR system, it is necessary to extract feature vectors as described in section 3.1.1 and to compute their alignments to phonemes as described in section 3.1.2 and section 3.1.3. For this purpose the classical GMM / HMM system is used. Subsequently, feature vectors are used as DNN inputs and alignments are used as DNN outputs in training. After the DNN is trained, its outputs are probabilities in the form:

$$P(S|X) \tag{3.1}$$

This represents probability of HMM state  $S$  to be selected when feature vector  $X$  is observed by the HMM. For following processing, it is required to convert this probability to form of the likelihood:

$$P(X|S) \tag{3.2}$$

This is done by using Bayes theorem (see equation (3.3)). Posterior probabilities  $P(S|X)$ , usually returned from softmax output layer of the NN, are converted to scaled likelihoods by dividing them by the  $P(S)$ . The factor  $P(S)$  is determined from frequencies of HMM states in forced alignment. All likelihoods should be then multiplied by factor  $P(X)$ , but this is simply omitted, since it has no effect on the alignment of the feature vectors and HMM states [5].

$$P(X|S) = \frac{P(S|X) P(X)}{P(S)} \tag{3.3}$$

## Chapter 4

# High dimensional data projection

This chapter briefly describes *Stochastic Neighbour Embedding* (SNE) and *t-distribution Stochastic Neighbour Embedding* (t-SNE) methods which are used for high-dimensional data visualization. Content of this chapter is mainly inspired by articles [6, 9].

### 4.1 Dimension reduction

Dimensionality reduction methods are mathematical functions or algorithms, which convert high-dimensional data set  $X = \{x_1, x_2, \dots, x_n\}$  into low-dimensional data set  $\mathcal{Y} = \{y_1, y_2, \dots, y_n\}$ . Number of dimensions of low dimensional data is usually 2 which is proper for plots or 3 for spacial models. Low-dimensional data representation  $\mathcal{Y}$  is sometimes referred as a *map* and individual data points  $y_i$  as *map points* [9]. Main goal of these methods is to convert these data points in such way, that structural relations between them will be preserved. This implies that similar high dimensional data points will stay close and dissimilar data will be distributed far apart.

### 4.2 Stochastic Neighbour Embedding (SNE)

*Stochastic Neighbour Embedding* is based on converting Euclidean distances between data points in high-dimensional space to probabilities that represent similarities. Similarity of data point  $x_j$  to data point  $x_i$  is the conditional probability  $p_{j|i}$ , that  $x_i$  will be neighbour of  $x_j$  with chance proportional to their probability density under Gaussian with center at  $x_i$ . This relation is represented with equation (4.1), where  $\sigma_i$  is variance of the mentioned Gaussian and can be used as parameter for adjusting densities of data point clusters. Equation (4.2) models similar conditional probability between data points  $y_i$  and  $y_j$ , only difference is, that variance parameter  $\sigma$  is fixedly set to  $\frac{1}{\sqrt{2}}$ , because in two dimensional space it has only scaling effect. Since both of these equations are modelling pairwise similarities, both values  $p_{i|i}$  and  $q_{i|i}$  are set to zero.

$$p_{i|j} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma^2)} \quad (4.1)$$

$$q_{i|j} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)} \quad (4.2)$$

In ideal case, map points  $y_i$  and  $y_j$  would correctly model similarities between points  $x_i$  and  $x_j$  and therefore probabilities  $p_{j|i}$  and  $q_{j|i}$  would be equal. This correctness of mapping high-dimensional space into low-dimensional space is expressed with objective function defined as *Kullback-Leibner (KL) divergence*. The SNE method uses gradient descent algorithm described in section 2.4.2, to stochastically find such mapping from high to low dimensional space, that KL function (see equation (4.3)) is minimized.

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \quad (4.3)$$

### 4.3 Crowding problem

The *crowding problem* can be illustrated with sphere centred on data point  $x_i$  with size given by radius  $r^m$ , where  $m$  is relatively high number of dimensions. If data points are evenly distributed in the region of the sphere, in attempt to model this sphere in two-dimensional map, the crowding problem arises, because area to accommodate data points in moderate distance is not large enough compared to area available to accommodate nearby data points. Therefore, if small distances will be modelled accurately in the map, moderate distances from data point  $x_i$  will have to be modelled too far away in the map. For more information about this problem see article [9].

### 4.4 t-Distributed Stochastic Neighbour Embedding (t-SNE)

Even though the SNE method gives quite good results in visualization of high-dimensional space, its objective function - Kullback-Leibner divergence is difficult to optimize and it is suffering from crowding problem described in section 4.3. As solution of crowding problem, Van der Maaten and Hinton proposed in [9] an alternative method, the *t-Distributed Stochastic Neighbour Embedding (t-SNE)*.

First major differences between SNE and t-SNE is that t-SNE uses joint probability distributions instead of the conditional probabilities which gives objective function represented with equation (4.4). This is referred to as symmetric SNE because for  $\forall i, j$  joint probability  $p_{ij} = p_{ji}$  and  $q_{ij} = q_{ji}$ . Advantage of symmetric SNE is simpler form of its gradient which implies faster computation in gradient descent algorithm.

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (4.4)$$

Second difference of t-SNE method lies in using of Student-t distribution with one degree of freedom instead of the Gaussian in the equation for *low-dimensional* space similarities (see equation (4.5)). Since in high-dimensional space is still used Gaussian distribution and Student-t distribution function has heavier tails, it models moderate similarities in high-dimensional space with larger distances in low dimensional space. This effectively copes the crowding problem.

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k,l=l}(1 + \|y_k - y_l\|^2)^{-1}} \quad (4.5)$$

Definition of similarities in high-dimensional space using joint probability distribution instead of conditional probabilities would cause problems in case of outlier data points.

Because of this, joint probabilities are forcefully set to be symmetric conditional probabilities as indicated by equation (4.6).

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \quad (4.6)$$

In summary, t-SNE method models dissimilar data points with much larger pairwise distances and similar data points with relatively small pairwise distances. This is consequence of using symmetric KL function and Student-t distribution in low-dimensional space. As a result, it solves crowding problem of the SNE method and produces better results of visualization.

It is necessary to emphasize that the low-dimensional output of t-SNE is based on probabilities, not distances, therefore only relative positions of data points provide informational value of the modelled data. This causes modelled data to be invariant to scale and rotation. Example of t-SNE method used for visualization of handwritten digits can be seen in figure 4.1. In this figure it is possible to see that digits similar in handwriting like 3 and 5 or 7 and 9 are placed relatively close, whereas dissimilar digits like 0 and 9 are placed far apart.

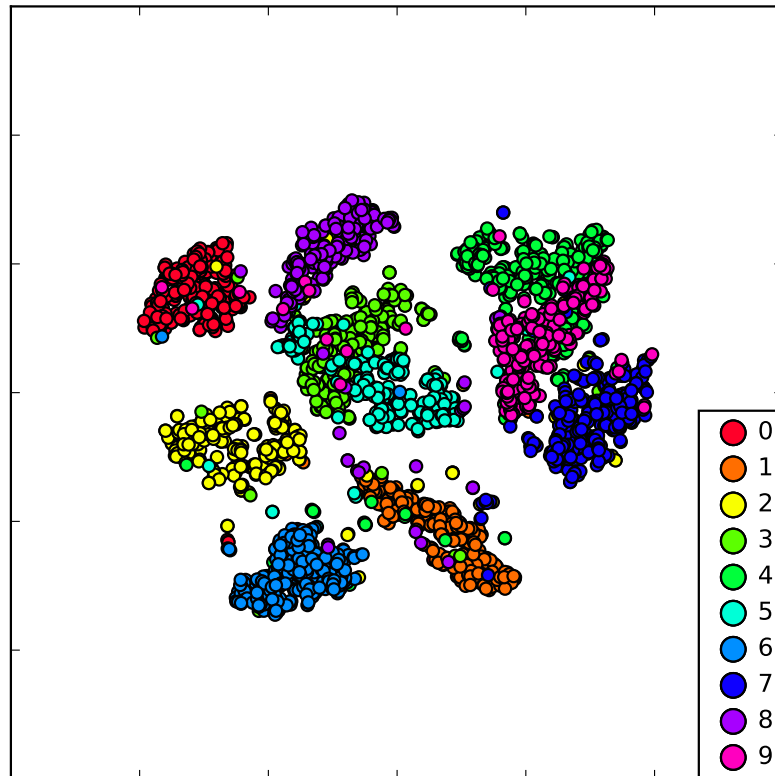


Figure 4.1: Demonstration of t-SNE method performance used on *MNIST* handwritten digits recognition task<sup>1</sup>.

<sup>1</sup>This image was generated using demonstration data distributed with t-SNE implementation in *python* from web site of the *t-SNE* author Laurens van der Maaten — <https://lvdmaaten.github.io/tsne/>

## Chapter 5

# Implementation of visualisation system

Chapter 2 – [Neural networks](#) and chapter 3 – [Automatic Speech Recognition](#) summarize basic theoretical background needed for understanding principles of neural networks and their use in automatic speech recognition. This chapter provides information about system designed for training and visualization of neural networks used for speech recognition. This system can be structurally divided into four parts:

1. Data preparation
2. Neural Network training
3. Activation extraction
4. Activation visualization

### 5.1 Data preparation

Goal of data preparation is extraction of phonetic features from speech recordings. *Kaldi toolkit* [12] and its standard scripts were used in this work. The Kaldi toolkit is an open-source framework for Automatic Speech Recognition providing functions for generating GMM / HMM or DNN / HMM systems, described in section 3.1.2. It contains simple command line tools written in C++, which are then called usually from *bash*, *python* or *perl* scripts.

In this work, Kaldi framework was used to extract speech features from AMI dataset IHM sound recordings. These features and AMI dataset labels were used to generate classic GMM / HMM recognition system producing feature alignments dataset for training and testing DNN, as described in section 3.2.

### 5.2 Neural Network training

Even though the Kaldi framework can be used to train DNNs and first experiments of the term project were based on this toolkit, after recommendations from my supervisor,

Martin Karafiát, CNTK framework was selected for DNN training. The CNTK has better and much simpler neural network design capabilities and it provides simple and efficient way of printing hidden units activations. Moreover it is compatible with Kaldi file format so no file conversions are necessary.

### 5.2.1 Computational Network Toolkit (CNTK)

The *Computational Network Toolkit (CNTK)* [1] is open-source framework for deep learning developed by Microsoft Research group. In the CNTK, neural networks are represented as computational nodes of directed graph. Leaf nodes of this graph represent input values or network parameters and other nodes represent matrix operations upon their inputs. This representation allows users to easily create and modify various types of neural networks. Another advantage of the CNTK is high performance on multiprocessor machines and high utilization of GPUs. Disadvantage of the CNTK framework is, that it is still in beta phase, so some features are not yet available and bugs are occurring.

## 5.3 Activation extraction

CNTK scripts similar to the one specified in appendix B.1 was used for activation printing. It prints values of activation functions of hidden units in response to neural network inputs. Activation values from every layer are stored in separate files. These files are formatted as Kaldi binary matrix feature files (sometimes called *ark files*) with format:

```
utterance_id_key1|BINARY_FLAG|DATATYPE_FLAG|row_count|col_count|data
utterance_id_key2|BINARY_FLAG|DATATYPE_FLAG|row_count|col_count|data
utterance_id_key3|BINARY_FLAG|DATATYPE_FLAG|row_count|col_count|data
...
```

Rows in these records are representing the reaction of hidden layer output to feature frame input indicated by line number in given utterance. Columns in these records are representing individual neuron outputs of specified layer and input. All outputs are floating point numbers with single precision as defined in CNTK configuration scripts.

For reading files containing extracted activations in Kaldi format described above, scripts providing *python* interface were used. These scripts (`kaldi_io.py` and `htk.py`) were created by Karel Veselý at BUT FIT and are available under the *Apache License, Version 2.0 (the "License")*. These scripts are reading the activation files one record at the time thanks to use of python iterators. This comes very handy since activation files tend to be quite big and loading them whole to memory is not possible.

## 5.4 Activation visualization

For the visualization and analysis of unit activations, special scripts in *python* was written. These scripts are based mainly on the previous work of Khe Chai Sim [16]. The steps of the algorithm for creation of interpretable activity regions and activation visualisation are described in following paragraphs.



### 5.4.1 Activity vectors

Consider activation value  $h_i^{(l)}(t)$ , which is response of the  $i$ -th neural unit in  $l$ -th layer to input feature at time  $t$ .

Before all, the activation values  $h_i^{(l)}(t)$  extracted from hidden layers must be rescaled to satisfy condition  $h_i^{(l)}(t) \geq 0$ . This is true for sigmoid and rectified linear units. For LSTM unit, which output is based on *hyperbolic tangent function*, values must be rescaled using *feature scaling normalization*:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (5.1)$$

Activation values normalised in this way can be used to create *activation vectors* which represent activations of certain unit with respect to given attributes (for example phonemes, speakers or noise types). As proposed by Khe Chai Sim in [16], the activity vector for  $S$  instances of the observed attribute is defined as:

$$a_i^{(l)} = [ a_i^{(l)}(1), a_i^{(l)}(2), \dots, a_i^{(l)}(S) ] \quad (5.2)$$

The  $s$ -th element of this vector is then given by:

$$a_i^{(l)}(s) = \frac{\sum_t \gamma_s(t) h_i^{(l)}(t)}{\sum_{s=1}^S \sum_t \gamma_s(t) h_i^{(l)}(t)} \quad (5.3)$$

Value  $\gamma_s(t)$  represents the probability of associating attribute  $s$  with input feature at time  $t$ . Usually, number of attribute instances is the number of classes, which the neural network is supposed to classify. Then  $\gamma_s(t)$  is set to 1 if the frame at time  $t$  corresponds to correct class, respectively with the data label at time  $t$ , otherwise  $\gamma_s(t)$  is set to 0. The labels can be represented as probabilities, in which case the value of  $\gamma_s(t)$  will represent confidences of associating attribute  $s$  with given input feature  $t$ .

Using normalisation described above, it is evident that  $a_i^{(l)}(s) \geq 0$  for all  $s$  and  $\sum_s a_i^{(l)}(s) = 1$ . Based on this, it can be said that the value  $a_i^{(l)}(s)$  is weight of hidden units activation in respect to attribute  $s$ . In other words, value  $a_i^{(l)}(s)$  is high when input acoustic frame belongs to attribute  $s$  and low when it does not.

### 5.4.2 Normalised entropy

From activation vector  $a_i^{(l)}$  can be calculated *normalised entropy* indicating information content, respectively sensitivity of given neural network unit  $a_i$  to observed attributes. Formula for normalised entropy of neural unit  $a_i$  is defined as:

$$E_i^{(l)} = \frac{-\sum_{s=1}^S a_i^{(l)}(s) \log a_i^{(l)}(s)}{-\sum_{s=1}^S \frac{1}{S} \log \frac{1}{S}} \quad (5.4)$$

Lower entropy value  $E_i^{(l)}$  corresponds to unit with higher information content and higher sensitivity to given attribute. In ideal case, unit which has entropy  $E_i^{(l)} = 0$ , has perfect (100%) sensitivity to one of the attributes and no sensitivity at all to any other attribute. Opposite case is unit  $a_i^{(l)}$  which is uniform vector of values  $1/S$ , therefore its entropy is maximal -  $E_i^{(l)} = 1$  and its informational content is none - it is called *insensitive* unit. Sim in his experiments discovered that almost 50% of units are actually insensitive. From his

experiments with entropy based pruning it can be assumed that they have no contribution to the result of classification and they could be pruned from the neural network in order to speed up the system.

### 5.4.3 Hidden activity space

In order to create interpretable 2D images,  $S$ -dimensional activation vectors need to be projected to 2-dimensional plane. The t-SNE method, described in section 4.2, is used for this projection. This plane is called *hidden activity space*. Based on the principle of the t-SNE method, units are positioned with probabilities corresponding to the similarities of their respective activation vectors. That means, units with similar activity vectors are positioned close together and units with different activity vectors are positioned far apart. Units placed in this manner form *convex hull*. Shape of the convex hull is determined by most *outer* units, which are usually units with most entropy, therefore were placed as far as possible from each other. Since t-SNE place units according with probabilities and not distances, it is not useful to measure distances in hidden activity space, useful information is contained only in relative positions of units. Example of hidden activity space and convex hull without interpretable regions is in figure 5.1.

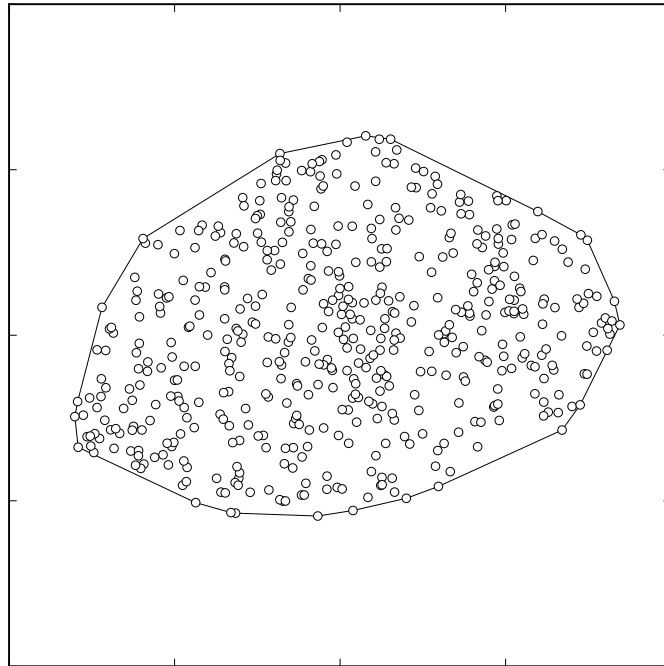


Figure 5.1: Example of hidden activity space.

It is necessary to note that t-SNE is randomly initialized and also highly data-dependent, because when reducing dimensions, pairwise probabilities are computed for every possible pair of points. In other words significant change in small number of points can entirely change the outcome. Therefore it is not possible to directly compare two visualisations created with different t-SNE projections. Solution to the random initialization is in this work solved by randomly pre-generating more initialization values than it is necessary. These values are then stored in file and loaded every time t-SNE is invoked. This change in t-SNE script does not have severe negative impact on quality

of visualisations, because it is possible to compensate for it with more iterations of the t-SNE algorithm. As for the data dependency problem, even for multiple different data batches to compare, it is possible to perform single t-SNE projection for all data-points as long as dimensions of the activity vectors are identical and they represent identical attributes. This is done simply by horizontally concatenating high-dimensional data points (over all layers of neural network or even over several neural networks), performing the t-SNE projection and afterwards horizontally splitting the low-dimensional vectors. This is possible because t-SNE method preserves order of the data-points. Subsequently can be split portions visualised and compared with each other. Only negative of this approach is that both time and memory complexity of the t-SNE method are  $O(n^2)$ , so concatenating large batches quickly becomes very resource-consuming.

#### 5.4.4 Delaunay triangulation

Algorithm for interpretable regions is dependent on computing *Delaunay triangulation* on points of the convex hull resulting from the t-SNE method. Delaunay triangulation was proposed by Boris Delaunay in 1934 [3].

Delaunay triangulation  $DT(P)$  is mathematically defined for set of non-linear points  $P$  in a plane. Triangles  $DT(P)$  are constructed in such way that no point from set  $P$  can be inside of any circle created by circumscribing all triangles  $DT(P)$ . When there are more ways to create triangulation, the way with maximal values of the smallest angles is selected, in order to prevent creation of slim triangles. Example of Delaunay triangulation performed on convex hull points is depicted in figure 5.2.

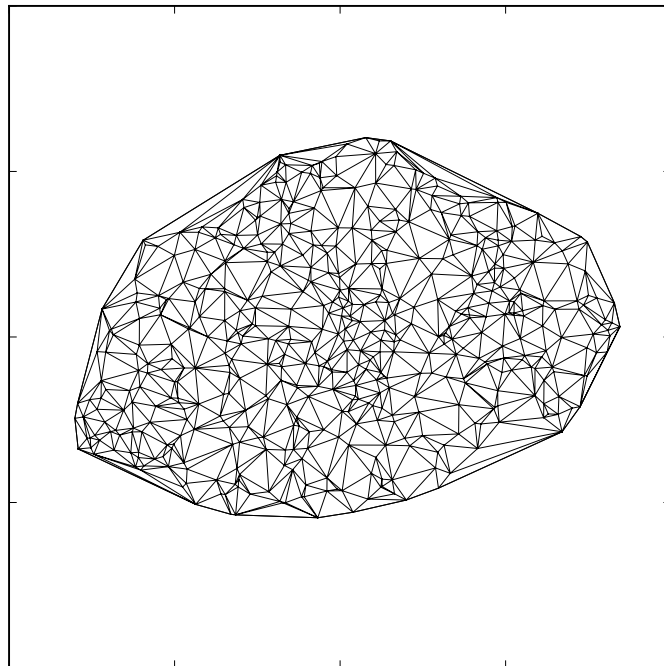


Figure 5.2: Example of Delaunay triangulation.

### 5.4.5 Ranking vectors

Before creating interpretable activity regions, it is necessary to compute *ranking vectors*. Ranking vectors are defined as:

$$r_i^{(l)} = [ r_i^{(l)}(1), r_i^{(l)}(2), \dots, r_i^{(l)}(S) ] \quad (5.5)$$

where  $r_i^{(l)}(s) \in 1, 2, \dots, S$  is the rank of attribute  $s$  computed from activation vector  $a_i^{(l)}(s)$  based on formula:

$$r_i^{(l)}(s) < r_i^{(l)}(s') \iff a_i^{(l)}(s) > a_i^{(l)}(s') \quad \forall s, s' \quad (5.6)$$

Practically this means that values of ranking vector represent order of attributes. Attribute for which the  $i$ -th unit activation is highest is ranked first, attribute for which the unit has the second highest activation is ranked second and so on up to the attribute for which the unit has the lowest activation value which is ranked as  $S$ -th.

### 5.4.6 Interpretable activity regions

*Interpretable activity regions* is the name of the first visualisation method proposed in [16]. Interpretable regions in hidden activity space are created according to following steps:

1. Find a *seed unit*  $O_s$  for each observed attribute  $s$ . The seed unit has the largest average activity value computed from itself and its immediate neighbours in Delaunay triangulation.
2. Use the seed units to initialise their respective regions:  $\rho_s = \{ O_s \}$ ;
3. Set the ranking threshold to  $r^* = 1$ ;
4. For each attribute  $s$ , recursively add all units, which satisfy condition  $r_i^{(l)}(s) \leq r^*$  and are connected by Delaunay triangulation to units in region.
5. Increment the threshold:  $r^* = r^* + 1$ ;
6. If  $r^* \leq S$ , go to step 4;

First seed units for each observed attribute are found. Subsequently seed units are used to initialize activity regions, which are incrementally expanding by including neighbouring units with higher or equal activation values (which are represented by lower or equal rank values) for current attribute. Adding of units continues until all units belong to some attribute.

For this work slight modification of the above algorithm was used. Instead of recursively adding neighbouring units, units were added iteratively. Iterative adding resulted in slightly more fair region creation since there were more cycles of the algorithm and therefore assigning of the units with high entropy (and low information value) is less dependant on the order of attributes in algorithm and more dependant on the activation values.

Example visualisation of neural units in hidden activity space using modified algorithm is depicted in figure 5.3.

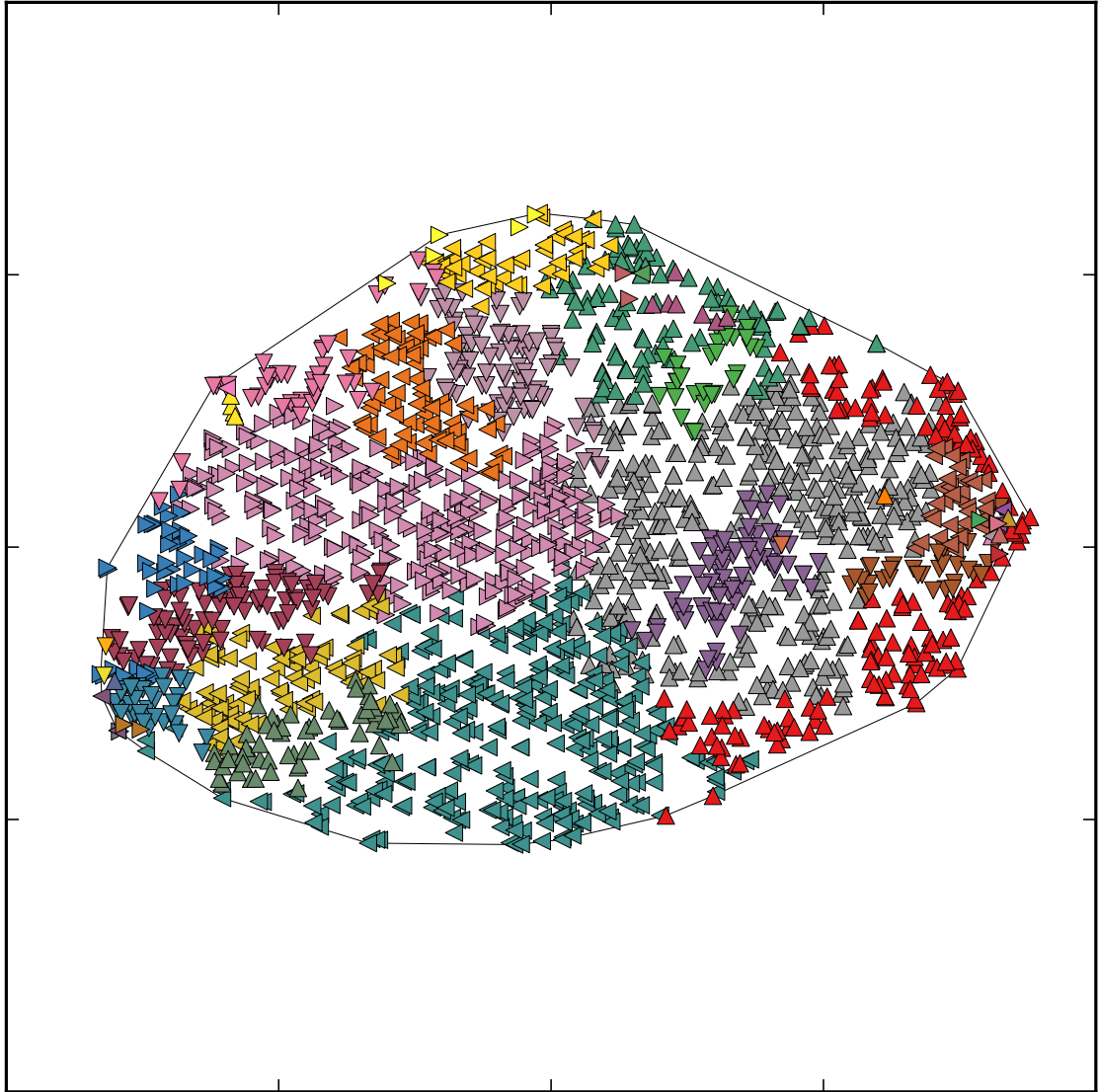


Figure 5.3: Example of interpretable attribute regions in 2D space.

### 5.4.7 Interpretable regions in 3D

Algorithm described in previous sections can be easily generalized for more than 2 resulting dimensions, because  $t$ -SNE method is capable of transforming its high dimensional input data to any dimension lower than the input dimension. The Delaunay triangulation can also be generalized to higher dimensionality — for example in three-dimensional space, tetrahedrons are created instead of triangles and are inscribed in spheres instead of circles. All other parts of the algorithm are invariant to change of output dimensionality. Example visualisation of neural units in three-dimensional hidden activity space is depicted in figure 5.4. Although creating three (and theoretically even more) dimensional regions is possible and should retain more information than 2D visualisation, interpretation of such visualisations is difficult, therefore it is not used in experiments of this work, but it is worth to note such possibility exists.

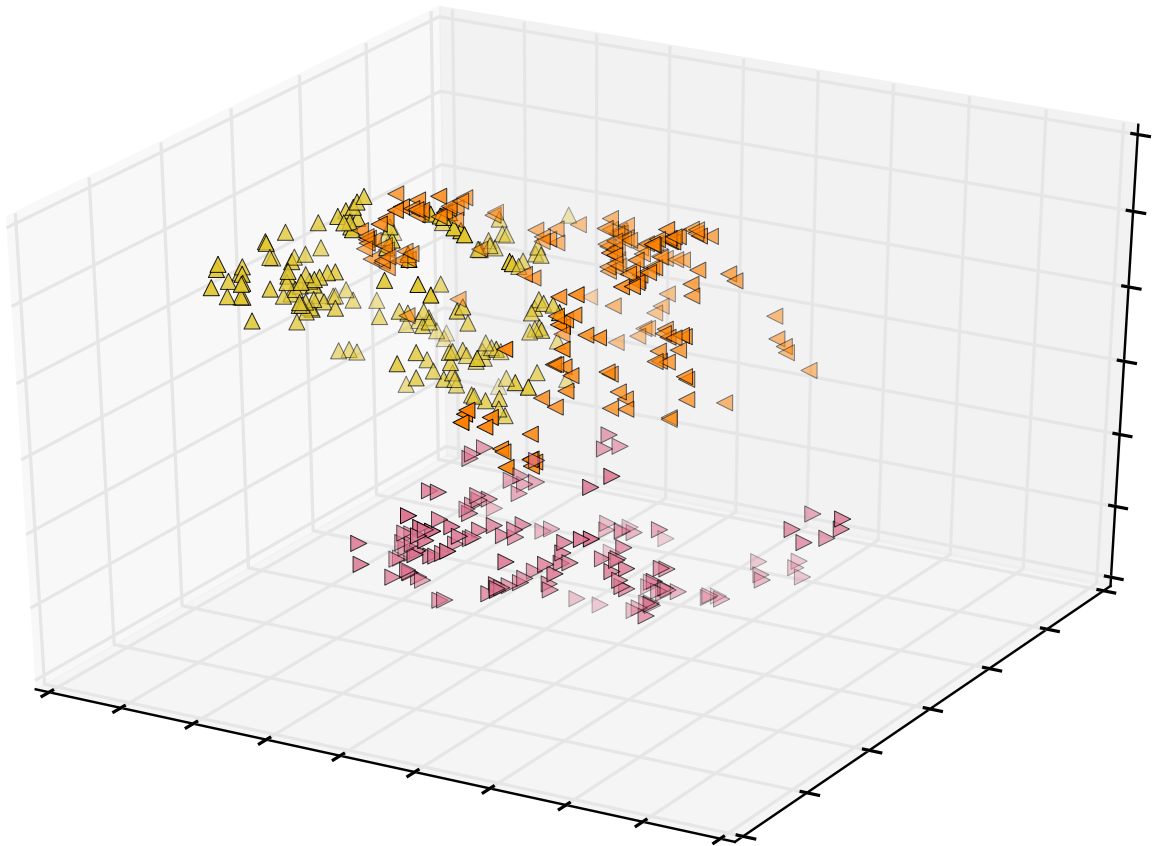


Figure 5.4: Example of interpretable regions in 3D space.  
(Only three groups of attributes are shown for clarity.)

### 5.4.8 Activity visualisation

*Activity visualisation* is alternative visualisation method proposed in [16]. Creating visualisations of hidden units activities is also dependent on producing hidden activity space by  $t$ -SNE method as described in section 5.4.3. Similarly to creation of interpretable activity regions, Delaunay triangulation is performed over all units creating triangular mesh. Every vertex of this triangular mesh is representing hidden unit

in activity space. To these units — vertices are assigned activation values of particular observed attribute instance from activity vector (described in section 5.4.1) corresponding to the individual unit. Subsequently, these activation values are converted to colours using suitable colour map. Finally, triangles produced by Delaunay triangulation are filled with colour gradient based on colours, respectively activations of their respective vertices.

Example of activity visualisation of three different attribute instances (*phonemes* /sil/, /AH/ and /S/) visualised for every layer of selected neural network is in figure 5.5. For conversion of unit activity to colour in this example, *jet* colour map was selected. Using jet colour map, red colour corresponds to high activation values and blue colour indicates low activation values of hidden units. This method makes possible to observe changes in activations corresponding to selected attribute instance over all layers of neural network. It would be even possible to observe these changes over time, although such experiment would require to snapshot activation values in every discrete time step, which would require tremendous amount of computational resources.

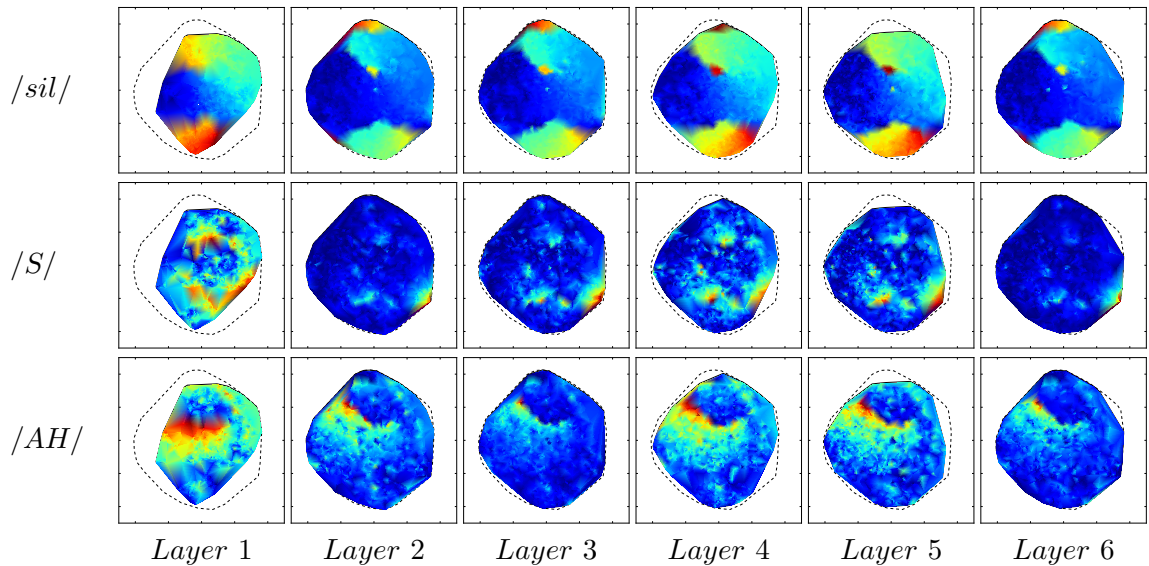


Figure 5.5: Example of activation visualisations of *phonemes* /sil/, /AH/ and /S/ over all 6 layers of neural network.

## Chapter 6

# Experiments

Since this work deals with visualisations of neural networks used in ASR, for experiments in this work was selected the most basic task of this field — the *phoneme recognition*. Following sections discuss choice of the dataset, modifications in approach of visualisation, neural network architectures and interpretations of resulting visualisations.

### 6.1 Dataset description

The *AMI Corpus* dataset was selected for experiments with retrieving activations from NN. Main reason for this selection was, that this dataset is well established in community of automatic speech recognition, and there are prepared examples for its use in most of the ASR frameworks like Kaldi or CNTK.

AMI (*Augmented Multi-party Interaction*) corpus is multi-modal data set consisting of 100 hours of conference room meeting recordings, intended for developing meeting browsing technology. This work will concern only with the audio part of the dataset. The audio data are divided into three parts based on microphone devices used for recording:

- **Independent headset microphone (IHM)**
- **Multiple distant microphone (MDM)**
- **Single distant microphone (SDM)**

Multiple distant microphone part of the corpus was recorded with microphone array put on the table. Single distant microphone part was generated as output of one of the microphones in array. Unfortunately, these parts are considerably suffering from microphone and environment noise, which presents itself also in speech recognition results in comparison to IHM data part. For this reason, only the IHM data part was used in experiments for neuron activations extraction.

The IHM data part contains 27 822 979 **frames** in 108 221 **utterances**. Features used in neural network training were computed by standard AMI recipe for Kaldi. Type of the features is *MFCC* (see section 3.1.1) with *Constrained Maximum Likelihood Linear Regression* for speaker adaptation.



## 6.2 Compensating dataset statistical characteristics

In first experiments with interpretable region visualisations of neural networks came up problem with large regions of one or more frequent attributes (*e.g* /sil/, /S/, /AH/ phones in phoneme recognition), which covered more than 90% of all units in every layer and every architecture type. This is caused by the statistical imbalance of input data. Large portion (almost 25%) frames of the dataset is actually silence and the rest of the frames is more or less evenly distributed amongst other 42 phonemes. This way the average portion of frames for phone other than silence is under 1.8% of dataset frames. When majority of the units have very high rank for silence, they are “swallowed” by it in the recursive part of the algorithm. This problem occurred most evidently in visualisations depicting individual layers transformed by the t-SNE method given only data of the single respective individual layer.

Because visualisations taken over almost entirely by the silence or another very common phones like /S/ or /AH/ have almost none informational value, two modifications of the visualisation algorithm were suggested. These modifications are described in following subsections.

### 6.2.1 Attribute occurrence normalisation

First option is to weight individual phonemes by number of occurrences of given phoneme in the dataset. This effectively changes the equation (5.3) to:

$$a_i^{(l)}(s) = \frac{\sum_t \gamma_s(t) h_i^{(l)}(t) \mathbf{w}(s)}{\sum_{s=1}^S \sum_t \gamma_s(t) h_i^{(l)}(t)} \quad (6.1)$$

where  $w(s)$  is the relative number of occurrences of attribute  $s$  in the dataset. This results in more evenly distributed values in activation vector, which subsequently results in higher entropy for neurons and more evenly distributed ranks. This manifests itself in the higher number of smaller attribute regions in resulting visualisations. Although this method artificially minimizes regions with large occurrence rate it provides the detailed view on the rare units.

### 6.2.2 Silence trimming

This solution is particular to visualisation of phoneme recognizing neural networks on dataset with very large ratio of silence in data. Since main purpose of silence is only to divide words, it is possible to discard large portion of it by restricting the sequences of silence in the dataset to maximal value of 15 consecutive frames. This modification of the dataset shows benefit in visualisations, because more attributes can take up the units usurped by the silence region.

Sadly, this approach cannot be used when dealing with large counts of phones other than silence and also it cannot be used for other other types of attributes, but it can be generalized to selecting statistically uniform subset of the dataset and use it for the neural network training. Of course in this way characteristics of the original dataset are lost, but such approach could be useful in comparison of different visualisation methods. Visualisations using the uniform subset should be similar to the ones using attribute occurrence normalisation described in section 6.2.2.

### 6.3 Experimental set-ups

For visualization and analysis of activation, 5 different neural networks were trained on the whole AMI dataset. These networks were divided into two experimental set-ups. First experimental set-up consists of four NNs with different architectures. Second experimental set-up consist of two identical neural networks trained on different datasets. Every set-up was individually processed whole in single t-SNE pass. This makes networks within the set-up comparable by using same high-to-low dimensional t-SNE mapping for every network / layer / unit. Specific architectures and parameters of neural network set-ups are described in following subsections. Finally, every set-up was processed in three different *views* — *ORIGINAL*, *NORMED* and *SILENCE*. The *ORIGINAL* view is the basic view with no normalization used. The *NORMED* view is based on normalizing dataset with attribute occurrences as described in section 6.2.1. The *SILENCE* view is based on silence trimming normalisation described in section 6.2.2.

#### 6.3.1 Experimental architectures

Four different types of DNN architectures were selected for experiments in this work. These neural networks were given names by their distinct properties: “*Normal*” DNN, *Bottleneck DNN*, *RELU DNN* and *LSTM DNN*. Baseline for this work is “Normal” DNN. It is standard feed-forward NN with 6 layers of sigmoid units and 2048 units per layer and Softmax output layer. Bottleneck DNN was derived from “Normal” DNN by reducing number of units in the 5-th hidden layer to 80 units. RELU DNN has also 6 hidden layers, 2048 neurons per layer and Softmax output layer, but it uses rectified linear activation function instead of sigmoid. Last NN architecture is LSTM which has only 3 hidden layers with 1024 LSTM units per layer reduced by projection layer to 512. The error prediction metric used in training of all mentioned architectures was *cross-entropy*. Parameters of trained architectures are recapitulated in table 6.1.

Experiment	Layers	Units per layer	Activation function
“Normal” DNN	6	2048	sigmoid
Bottleneck DNN	6	2048 (80) <sup>1</sup>	sigmoid
RELU DNN	6	2048	ReLU
LSTM DNN	3	1024(512)	LSTM

Table 6.1: Distinctions between selected architectures for visualisation experiments.

#### 6.3.2 Experimental environments

For comparing behaviour of the neural networks with identical architecture on different dataset, neural network referred to as “*Noised*” DNN was trained. Architecture of “Noised” DNN is exactly the same as “Normal” DNN, so it has 6 hidden layers with 2048 sigmoid units per layer and Softmax output function and cross-entropy error prediction metric, but it is trained on slightly different dataset. In order to train this network, the AMI dataset was randomly noised with 9 different types of environmental noise (*restaurant, fan, music,*

<sup>1</sup>Number of units in the 5-th layer – the bottleneck.

keyboard, nature, dishes, crowd, motor and office noise) with 5 different types of SNR ( $-5dB$ ,  $0dB$ ,  $5dB$ ,  $10dB$ ,  $15dB$ ).

## 6.4 Performance of the analysed networks

This section provides comparisons of performances of individual trained neural networks. These comparisons are separated into two parts based on common factors.

Table 6.2 displays performances of neural networks with different architectures, trained on the clean AMI dataset described in section 6.1. The Kaldi DNN results in this table are results provided by authors of Kaldi AMI recipe [12]. These results should correspond with the “Normal” DNN network trained in CNTK framework. It serves as baseline for comparison with other types of networks. Since differences between the official Kaldi results and my baseline network are well within margin of error, it can be concluded that the “Normal” DNN is correctly trained and suitable for use as reference to other networks trained for this work.

It is worth to mention that even with such changes in architecture like reducing one of the layers to 80 units in Bottleneck DNN or changing the type of activation function to from sigmoid to ReLU, trained neural networks were able to maintain roughly similar word error rates. It is also noteworthy that best results has the LSTM network with only half count of layers and half count of units per layer compared to baseline “Normal” DNN.

<b>Experiment</b>	<b>Total Error [%]</b>	Subs. [%]	Ins. [%]	Del. [%]	<b>Total Correct [%]</b>
<b>Kaldi DNN</b>	27.2	15.5	3.5	8.1	76.2
<b>“Normal” DNN</b>	27.4	15.7	3.4	8.1	76.2
<b>Bottleneck DNN</b>	27.5	15.8	3.3	8.3	75.8
<b>RELU DNN</b>	27.6	15.8	3.5	8.4	75.9
<b>LSTM DNN</b>	25.1	14.3	3.1	7.7	78.0

Table 6.2: Word error rates of different neural networks trained on AMI dataset.

Table 6.3 displays performances of neural networks with identical architectures described in section 6.3.2. The “Normal” DNN was trained on clean AMI dataset, whereas the “Noised” DNN was trained on noised AMI dataset also described in section 6.3.2. It resulted in its noticeably worse performance.

<b>Experiment</b>	<b>Total Error [%]</b>	Subs. [%]	Ins. [%]	Del. [%]	<b>Total Correct [%]</b>
<b>“Normal” DNN</b>	27.4	15.7	3.4	8.1	76.2
<b>“Noised” DNN</b>	54.2	29.3	4.7	20.2	50.5

Table 6.3: Word error rates of identical neural networks trained on clean and noised AMI dataset.

## 6.5 Comparison of different views on experimental data

As described in the beginning of section 6.3, all experimental set-ups were processed in three different views motivated by unevenness of phoneme distribution in interpretable activity regions. This comparison tries to determine the best of the proposed views for neural network analysis using interpretable activity regions.

In figures 6.1, 6.2 and 6.3 are depicted views of the third layer of the baseline “NORMAL” DNN. Numbers in legend indicate count of the units assigned to the concrete phoneme region. The dashed polygon in these figures represents the convex hull of the whole network. The solid polygon represents the convex hull of the current displayed layer.

The ORIGINAL view in figure 6.1 is very similar to the SILENCE view depicted in figure 6.3, with only difference of reduced silence attribute cluster in SILENCE view. Motivation behind creation of the SILENCE view was expectation that after reducing the significantly dominating silence phone, it will be possible to observe other less dominant phones. This assumption was based on inner workings of the activity region algorithm, especially the ranking vector creation. It was assumed that silence which had first rank in very high number of phonemes was effectively “hiding” phones which were ranked second or third in ranking vectors. Unfortunately, presented images indicate that neurons corresponding to silence in the ORIGINAL view were taken over by phoneme /AH/. Similar phenomenon can be observed by comparing other layers and networks of the ORIGINAL and SILENCE view. These images are available on the enclosed DVD. It can be concluded, that silence trimming view is not beneficial to the interpretable activity region method in terms of enabling more detailed observation of attributes.

The NORMED view in figure 6.2 on the other hand displays many different regions than the ORIGINAL view. With some exceptions like /B/ or /S/ in can be seen that numbers of units corresponding to certain phonemes are inverse — high number of units for certain phone in ORIGINAL view corresponds to low number of units in NORMED view and *vice versa*. This is expected because NORMED view was created by normalizing the ORIGINAL view by number of occurrences. The exceptions can be explained by units being taken over by neighbouring cluster with similar phoneme properties. For example, phoneme /B/ should have high number of units in NORMED view but it does not, instead phone /D/ has very significant number of units. Both phone clusters are located in close proximity in both views, therefore it can be assumed that some of the units that phone /B/ was supposed to have in NORMED view were “stolen” by /D/. Similar inverse units “stealing” can be seen for /V/ units being stolen by /F/. This was most probably caused in recursive stage of the activity regions method where “stealing phone” got to the proximity of the “victim’s” units sooner and took them over even thou he had smaller rank than victim. These exceptions could be solved by grouping phones or generally observed attributes into larger logical categories with similar attributes like /B/ and /D/ in category “stops” or /F/ and /V/ in category “fricatives”. Regardless of exceptions, it can be concluded that NORMED view provides way to display units suppressed in original view. However, it should be noted that activation rate of these suppressed units is inverse.

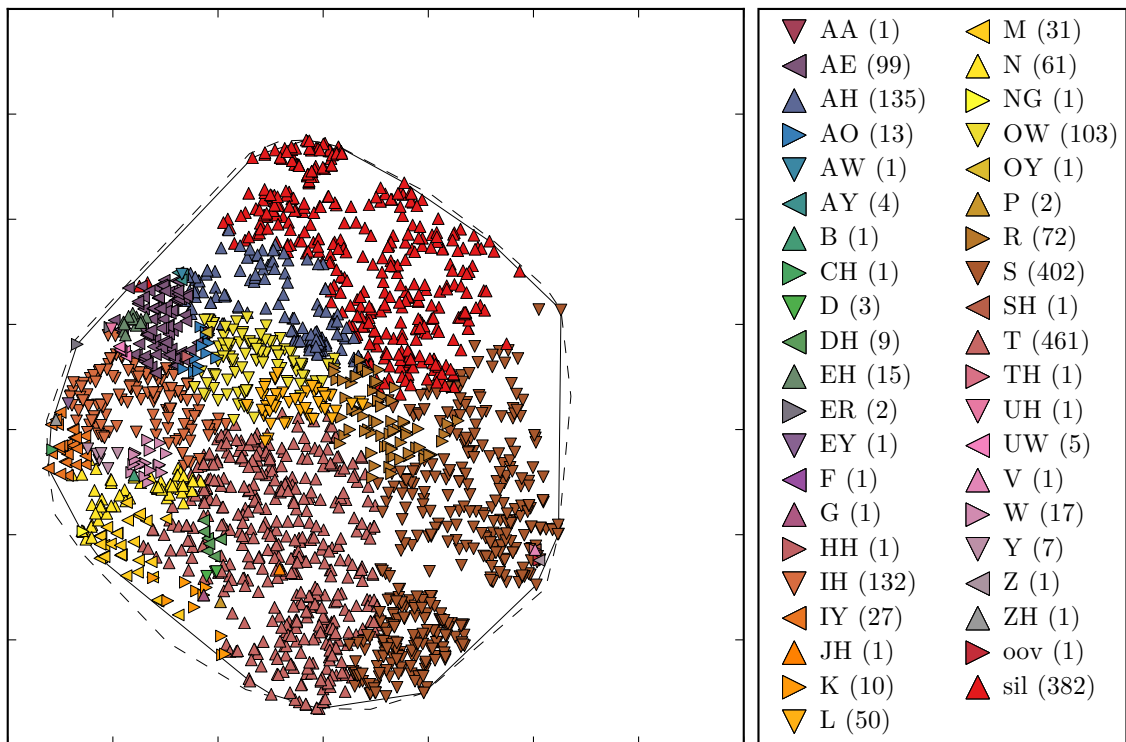


Figure 6.1: Third layer of “NORMAL” DNN in ORIGINAL view.

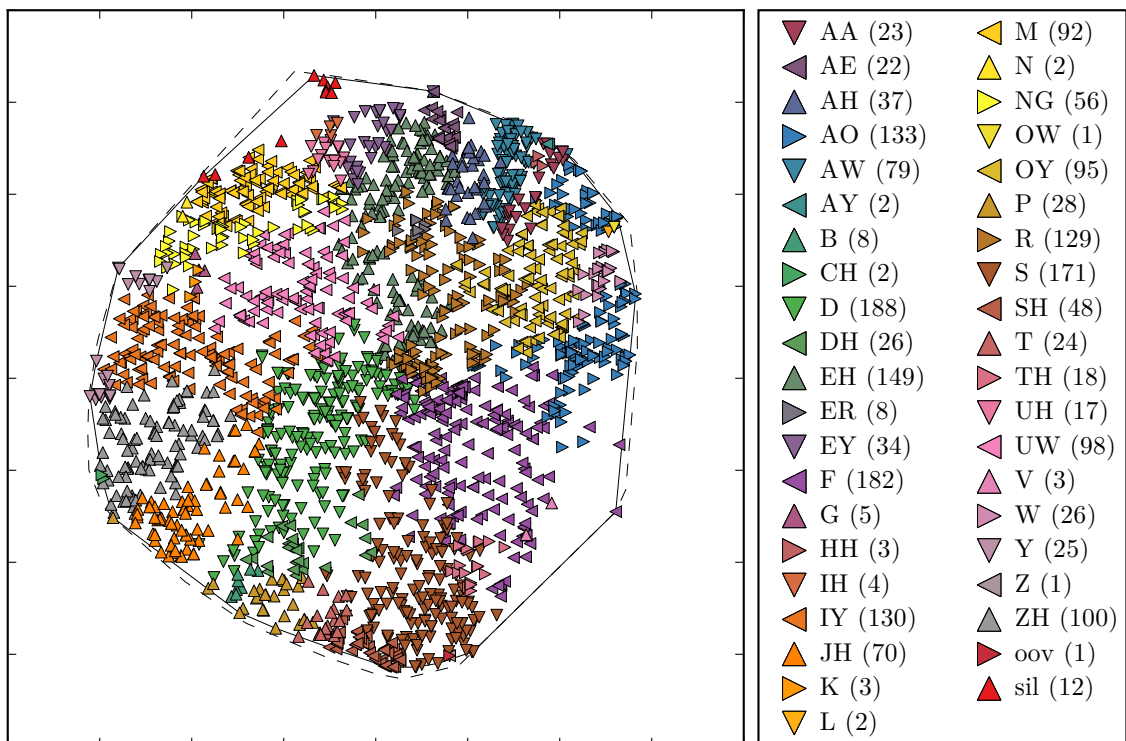


Figure 6.2: Third layer of “NORMAL” DNN in NORMED view.

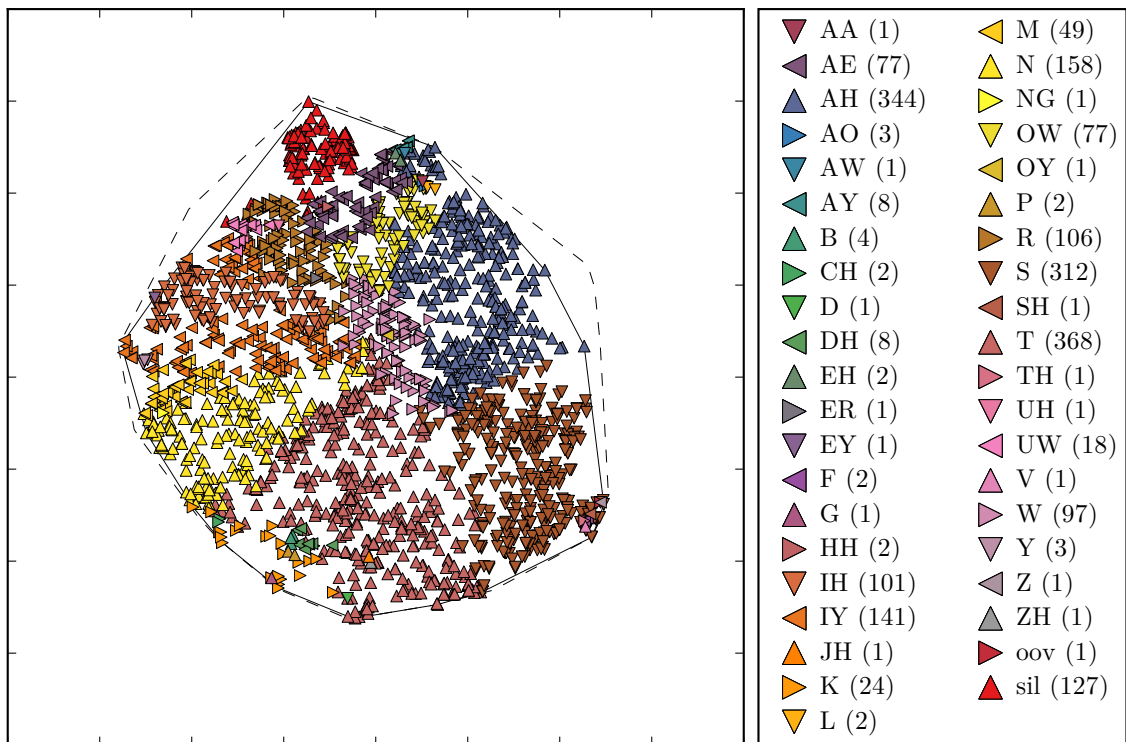


Figure 6.3: Third layer of “NORMAL” DNN in SILENCE view.

## 6.6 Comparison of visualisation methods

This section compares two implemented visualisation methods proposed by Khe Chai Sim in [16]. In figures 6.4 and 6.5 is depicted layer three of the “NORMAL” DNN in the ORIGINAL view. Visualisation in figure 6.4 was generated using interpretable activity region method described in section 5.4.6. Figure 6.5 was created using activity visualisation method described in section 5.4.6. It depicts unit activation vectors for the /sil/ phone. The silence phone has very active units in the top of the convex hull, which corresponds to its interpretable region but it has also another region in the bottom which is not indicated in the interpretable regions. This can be caused by other phones having better rank in this region and therefore effectively hiding the silence. Another, more probable, explanation is that lack of silence region in the bottom of the convex hull is caused by the nature of interpretable region algorithm, since it allows only one seed unit being chosen. In this case, the seed unit is chosen in the top of the convex hull where the activation is highest, but other phone clusters in the middle are effectively preventing it (as they should) to reach the second large activity region.

From above-mentioned can be concluded, that interpretable regions provide fairly good overall information about most attribute regions with most significant activation values, but they have lower informational value, since regions are created only in one place, whereas in activity visualisations there is detailed information about individual unit activations.

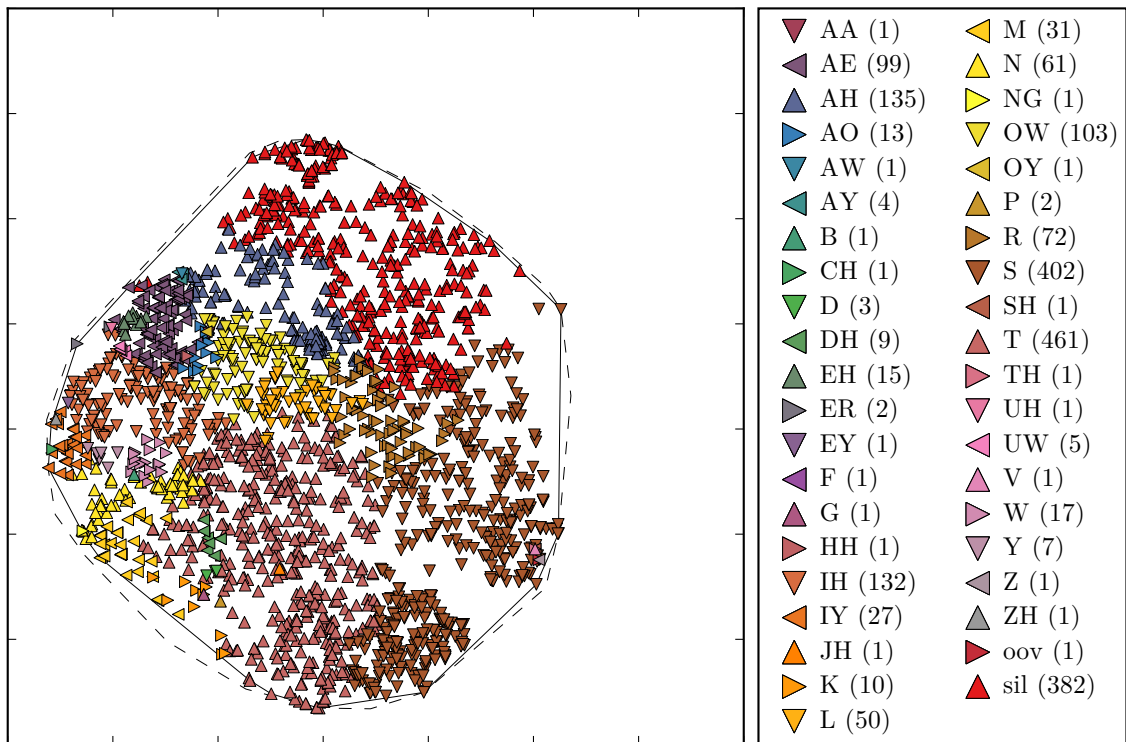


Figure 6.4: Interpretable regions in third layer of “NORMAL” DNN in ORIGINAL view.

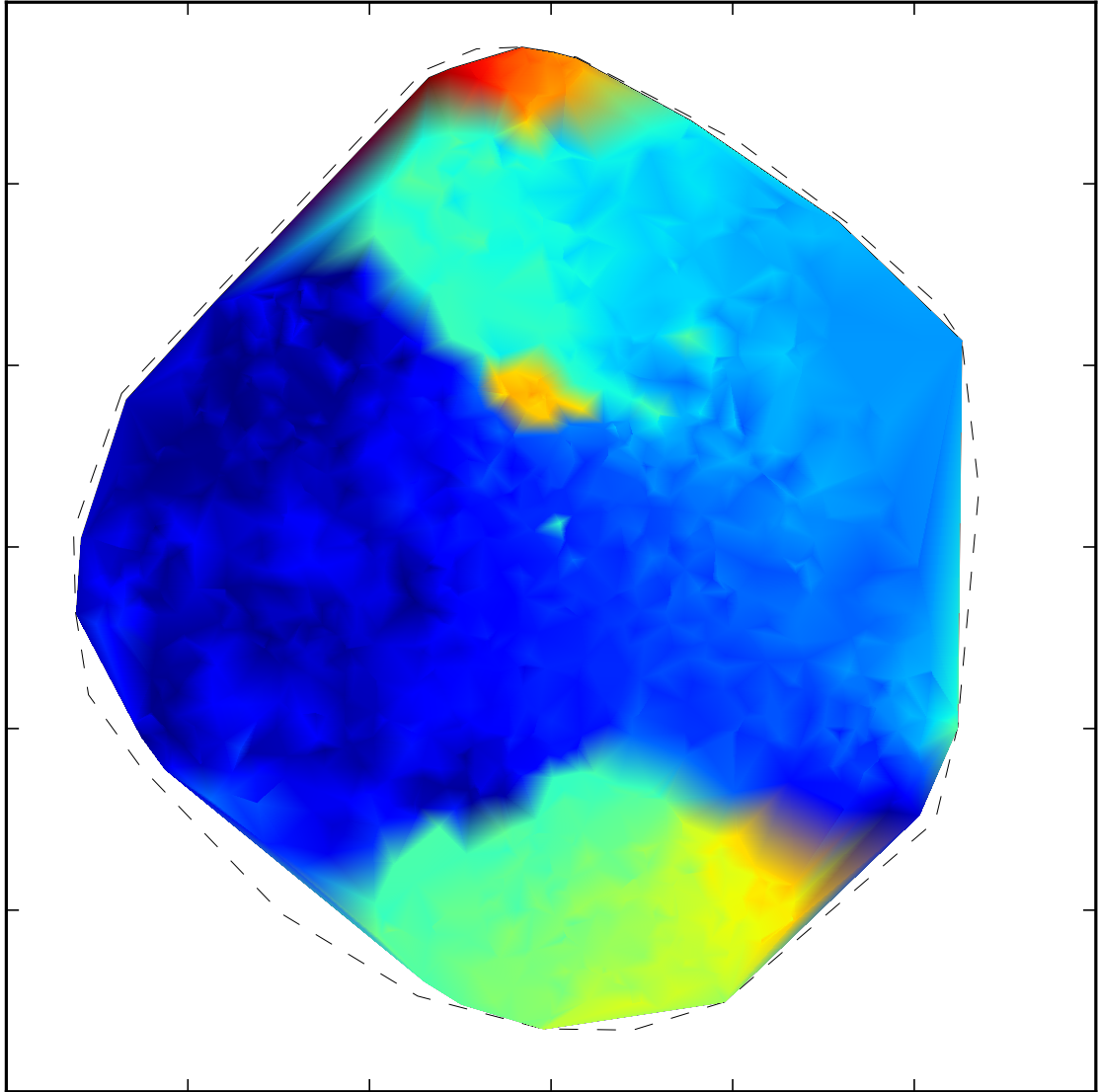


Figure 6.5: Activity visualisation of *sil* phone in third layer of “NORMAL” DNN in ORIGINAL view. (Red colour indicates high unit activation values, blue colour indicates low unit activation values.)



## 6.7 Analysis of NNs with different architectures

Purpose of this experiment is to compare 4 DNNs with different architectures trained on identical dataset. Neural networks are compared using activation visualisation for 4 phonemes: */sil/*, */AH/*, */AE/* and */S/* which are displayed in figures 6.6, 6.7, 6.8 and 6.9. All layers of all networks were computed in single t-SNE pass and therefore lie in the common activity space. The dashed polygon in these figures represents the convex hull of the whole network. The solid polygon represents the convex hull of the current displayed layer. Intensity of activation values are represented using jet colour map — red colour represents high activation values and blue colour represents low activation values.

At first glance, the layers of first three neural networks look very similar in shape and also in colour. Unfortunately, the shape of LSTM neural network came out deformed in arc shape. This is almost certainly caused by data scaling of the LSTM values to conform to interval  $< 0; 1 >$ , therefore there was correlation between all LSTM units which t-SNE modelled in this way. This reasoning is also confirmed by the fact that their region is relatively small in comparison to other networks and it is placed far from the centre of the common activity space. Because of the deformation, there can not be made any other conclusions about the LSTM neural networks in this experiment.

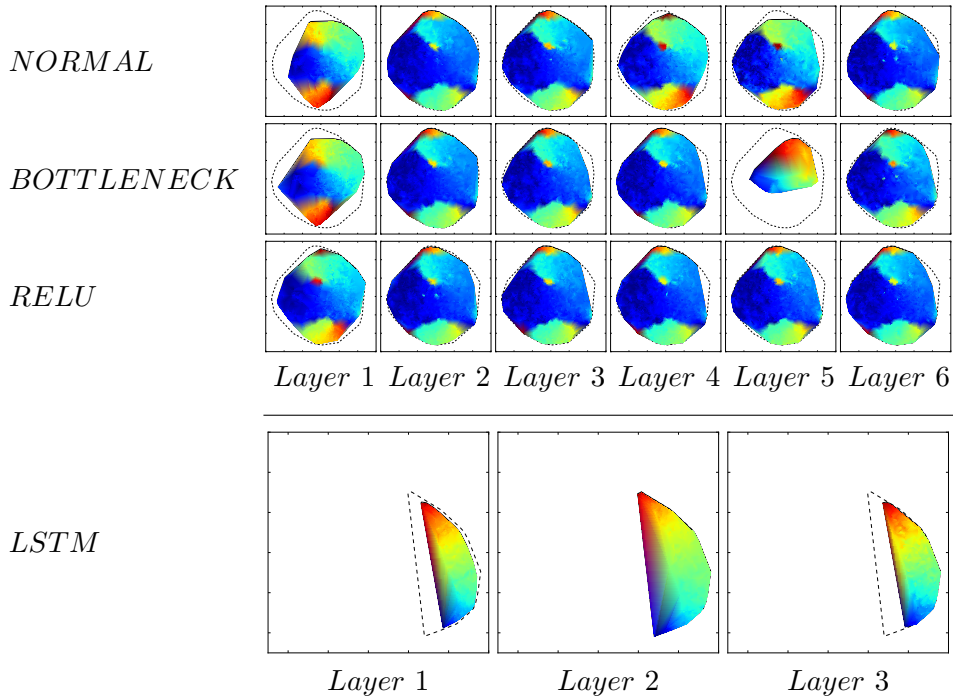


Figure 6.6: Visualisation of */sil/* phoneme activations in different NN architectures.

Layers of the Bottleneck DNN are almost identical to layers of the baseline “NORMAL” DNN. This is caused by fact that they have identical architecture with exception of the 5-th layer which is the bottleneck. In the bottleneck is represented with small region of more active units for every phone. This is in concert with theory that bottleneck compresses the information. It seems that the bottleneck is influencing the activations in both, the preceding and the following layer. Interesting is that 6-th layer of the “NORMAL” DNN and 4-th layer of the BOTTLENECK DNN are almost identical. It is possibly caused by

the the Softmax layer following the 6-th layer which has only 660 outputs and therefore it can be acting like the bottleneck.

There are also interesting similarities in shapes and activations between “NORMAL” DNN and RELU DNN layers. Layer 6, respectively 4 of “NORMAL” DNN is similar to layer 2 respectively 1 of RELU DNN for silence phoneme. There are similarities in other phonemes for higher layers of the “NORMAL” DNN and lower layers of the RELU DNN. This can be explained only by difference between their activation functions (sigmoid vs. ReLU). It is possible that steeper gradient in ReLU propagates more information back into the lower layers.

It is necessary to note that most significant differences in activations between “NORMAL”, BOTTLENECK and RELU DNNs are occurring in first layers (with exception of the bottleneck layer). It is possible that in this layer the neural network is learning some other properties like extraction of phone information from features or trimming speaker information which manifests itself in phoneme attributes in presented way.

Another interesting observation is that similar phones have high activity units placed near each other. This can be seen by comparing same phonemes /AH/ and /AE/. These phones belong to same phonetic group — vowels. They are voiced (modulated by vocal cords) and they also sound similar, therefore it is expected that they will have similar activity vectors and units responsible for these phones were placed by the t-SNE method close to each other. On the other hand /S/ phone is unvoiced consonant. It is expected that it will be placed as far as possible from voiced vowels and it is also possible that it will be assigned to considerable number of units since it is similar to noise. The silence phone /sil/, had most occurrences in dataset. For this reason it is expected that it have many significant activation values.

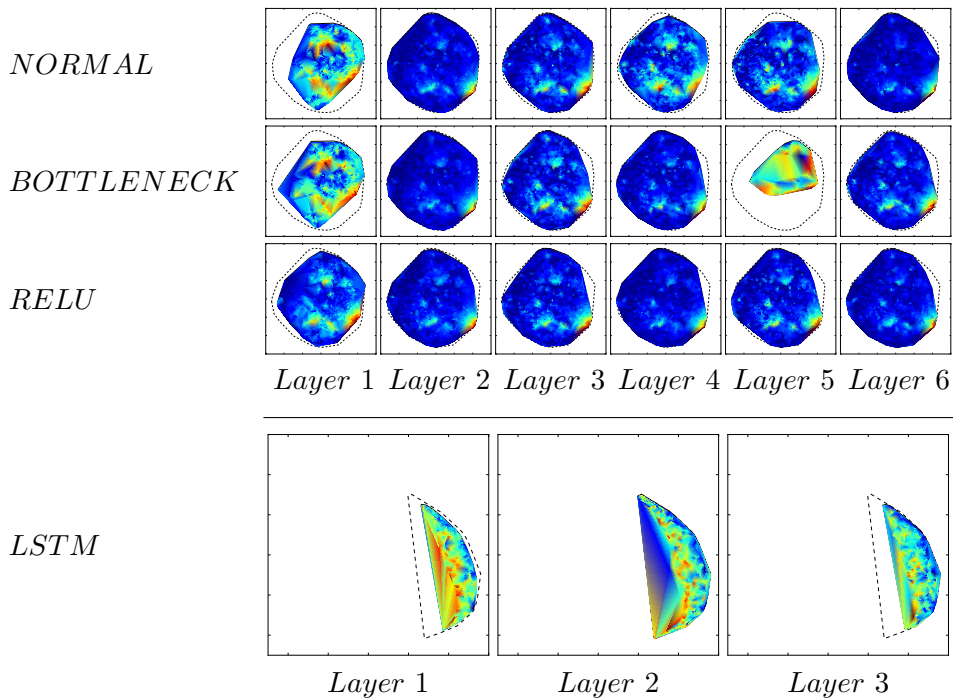


Figure 6.7: Visualisation of /S/ phoneme activations in different NN architectures.

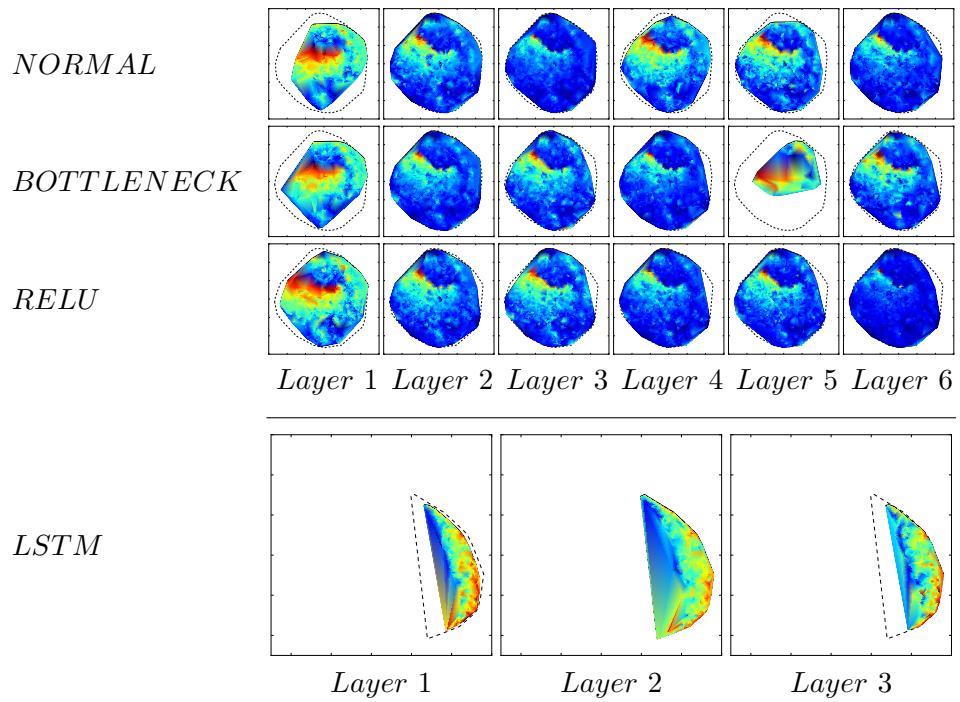


Figure 6.8: Visualisation of /AH/ phoneme activations in different NN architectures.

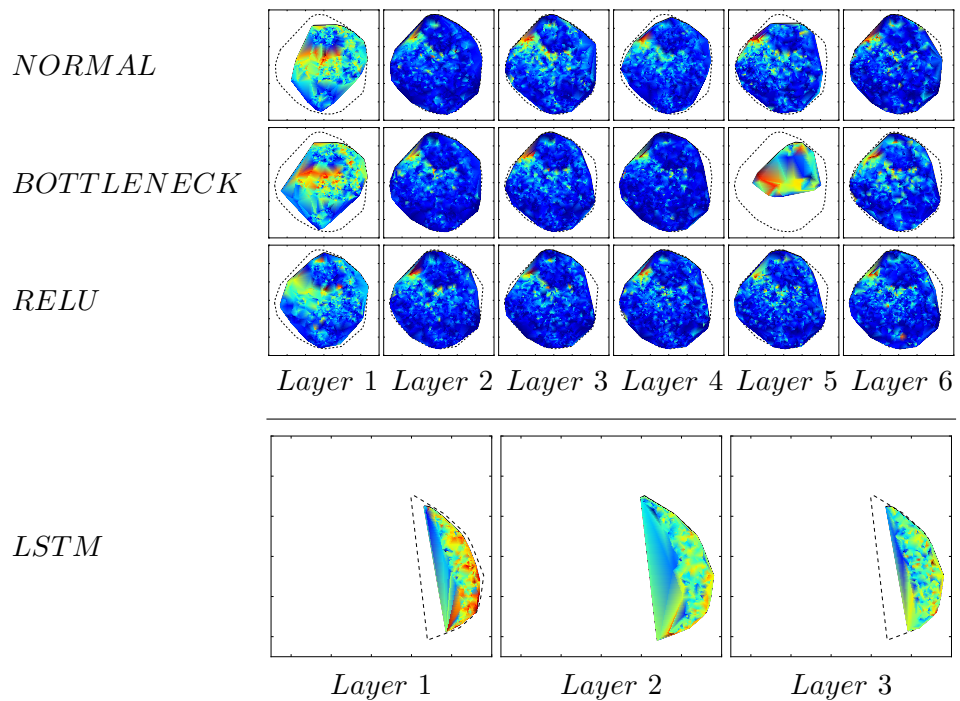


Figure 6.9: Visualisation of /AE/ phoneme activations in different NN architectures.

## 6.8 Analysis of NNs trained with different datasets

This experiment concerns two neural networks with identical architectures trained on different datasets. The analysis will be based on the same phones as in the previous experiment, namely */sil/*, */AH/*, */AE/* and */S/* which are displayed in figures 6.10, 6.11, 6.12 and 6.13. It must be noted that figures in this section can not be directly compared to figures from previous section, because they were mapped from high to low dimensional space using different t-SNE passes.

In figure 6.10 depicting silence, it can be seen that the NOISED DNN has slightly less activation values than the “NORMAL” DNN. It is most evident in the 5-th layer. This is very much expected because of the noise added to the recording. The noised neural network is mistakenly classifying silence as some of the phonemes. This occurs most evidently for */AE/* phone in layers 2, 3 and especially 4.

Another implication caused by noised data is that NOISED dataset has bigger convex hulls. This can be clearly seen in the first layer of every phoneme. The noise is effectively diversifying the data and therefore the activation vectors. The t-SNE method is then trying to place units with diverse activation vectors as far as possible from each other which results in increase of the convex hull size.

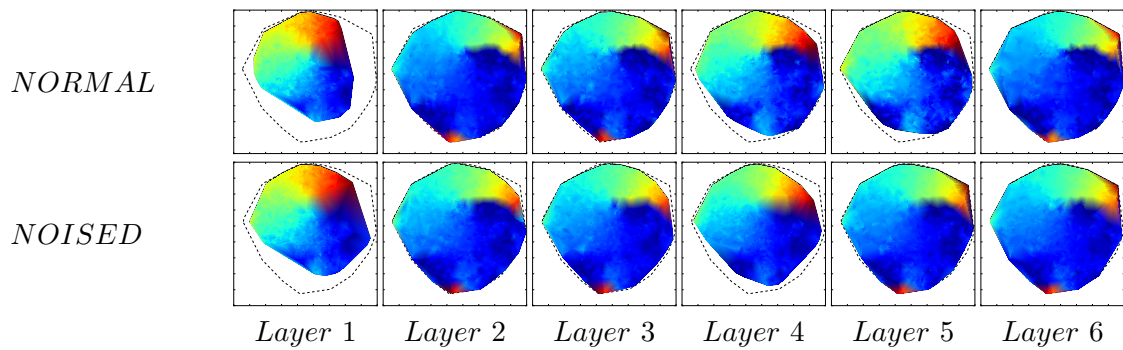


Figure 6.10: Visualisation of */sil/* phoneme activations in different environments.

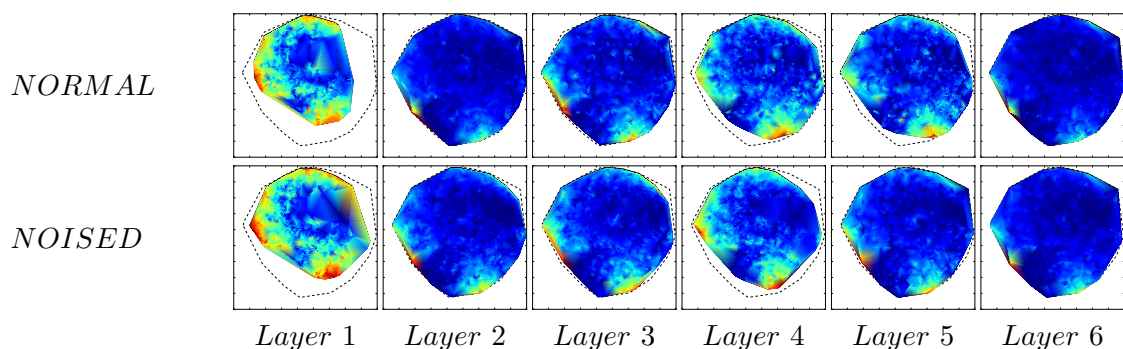


Figure 6.11: Visualisation of */S/* phoneme activations in different environments.

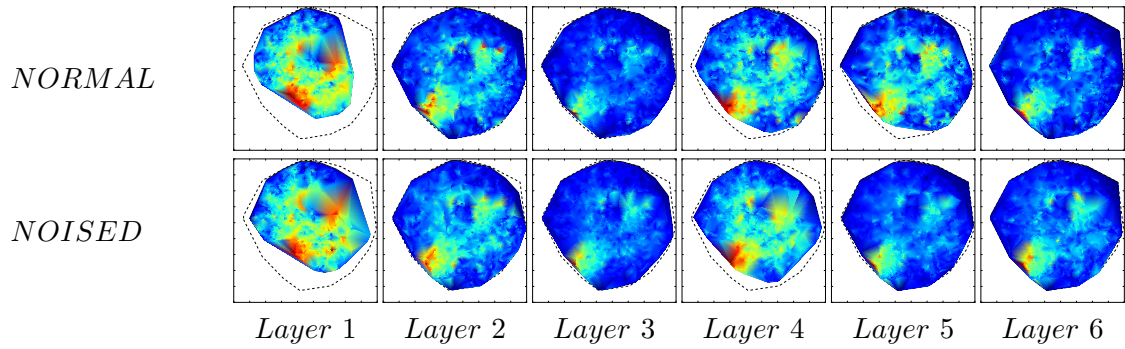


Figure 6.12: Visualisation of /AH/ phoneme activations in different environments.

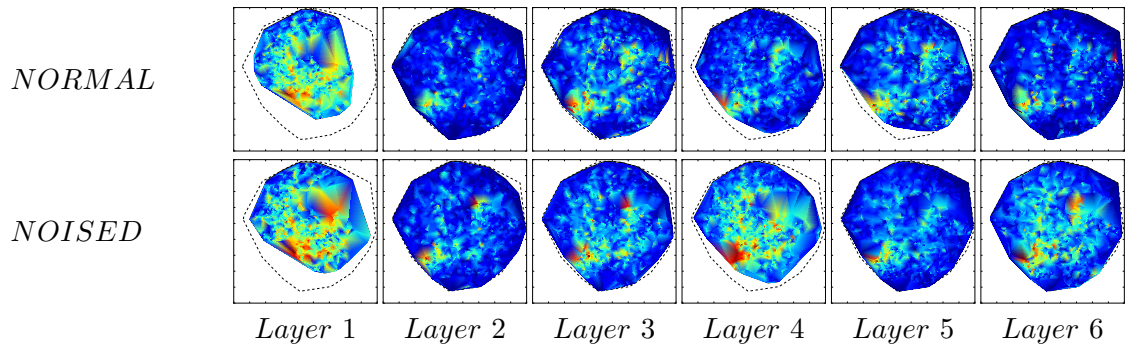


Figure 6.13: Visualisation of /AE/ phoneme activations in different environments.

## 6.9 Experiments epilogue

It is important to note that experiments in this chapter were evaluated by closely observing zoomed detailed versions of images. All images generated by the implemented visualisation system for these experiments can be found on enclosed DVD disk (see appendix A).

## Chapter 7

# Conclusion and future plans

In the first part of this work, basic theoretical knowledge of neural networks, their structure and learning principles was provided in chapter 2 — **Neural networks**.

Chapter 3 — **Automatic Speech Recognition** concerned with brief introduction to the usage of neural networks for acoustic modelling in automatic speech recognition. In Chapter 4 — **High dimensional data projection** were described basic informations about t-SNE — the method, selected for high-to-low dimensional mapping.

Second part of this work described implementation of system capable of visualising activations of neural networks used for automatic speech recognition in chapter 5 — **Implementation of visualisation system**. This part was based on article [16] written by *Khe Chai Sim*. Proposed method was in this work extended with creating two different views on the dataset. Motivation for this extension was statistical non-uniformity of the AMI dataset, which had negative impact on method creating interpretable regions. Fortunately, it did not influence creation of activity visualisations. Also, the possibility of creating 3D visualisations was noted.

Last part of this work contained experiments using implemented system. These are described in chapter 6 — **Experiments**. In these experiments were first compared different dataset views and visualisation methods. Conclusion of these comparisons is that creation of interpretable activity regions is very prone to hiding relevant detailed information about unit activations. On the other hand activity visualisation method provides stable, consistent and detailed visualisations. Subsequently, experiments analysing neural networks with different architectures and neural networks trained in different environments were evaluated. These experiments were performed on the original dataset view using activity visualisation method.

All work on this thesis was processed on the Czech National Computational grid — Metacentrum. This includes compilation of Kaldi and CNTK frameworks, feature preparation using Kaldi, training and printing activations of 5 different neural networks using CNTK, each on 27 822 979 **frames in 108 221 utterances** of the AMI dataset, which produced overall **over 4.5 TB of unit activation data**. Subsequently, these data were processed into activation vectors which were in three different views (original, normalized with occurrences and with trimmed silence) transformed using t-SNE spanning over 4 neural networks with different architectures and 2 neural networks in different environments. Using results of t-SNE, interpretable activity regions and activity visualisations were created and analysed. Overall **CPU usage** of this work was more than 660 **CPUdays**. Statistics for GPU usage are not available but it is expected to be several times the CPU usage since all NN training, activation printing

and substantial part of feature preparation ran on GPU.

This work opens wide range of possibilities for future work. It is possible to use implemented system for analysing more different architectures and more different environments. The visualisation system could also be easily extended to other fields of neural network research like image processing or stock market prediction. It is also possible to increase speed of neural networks by analysing activity vectors of neurons and pruning neurons with low informational value as suggested in [16]. It should also be worthwhile to research advantages of visualising interpretable regions in 3D space. It is probable that there would be more information retained after high-to-low dimensional mapping.

# Bibliography

- [1] Agarwal, A.; Akchurin, E.; Basoglu, C.; et al.: An Introduction to Computational Networks and the Computational Network Toolkit. Technical Report MSR-TR-2014-112. August 2014.
- [2] Aggarwal, C.: *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Taylor & Francis. 2014. ISBN 9781466586741. 707 pp.
- [3] Delaunay, B.: Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*. vol. 7, no. 793-800. 1934: pp. 1–2. [In French].
- [4] Greff, K.; Srivastava, R. K.; Koutník, J.; et al.: LSTM: A Search Space Odyssey. *CoRR*. vol. abs/1503.04069. 2015.
- [5] Hinton, G.; Deng, L.; Yu, D.; et al.: Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Signal Processing Magazine*. 2012.
- [6] Hinton, G. E.; Roweis, S. T.: Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems 15*, edited by S. Becker; S. Thrun; K. Obermayer. MIT Press. 2003. pp. 857–864.
- [7] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. *Neural Comput.*. vol. 9, no. 8. November 1997: pp. 1735–1780. ISSN 0899-7667. doi:10.1162/neco.1997.9.8.1735.
- [8] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *Nature*. vol. 521, no. 7553. May 2015: pp. 436–444. ISSN 0028-0836. insight.
- [9] Van der Maaten, L.; Hinton, G. E.: Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*. vol. 9: 2579–2605. Nov 2008.
- [10] Mcculloch, W. S.; Pitts, W. H.: A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*. vol. 5. 1943: pp. 115–133.
- [11] Munakata, T.: *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More (Texts in Computer Science)*. Springer Publishing Company, Incorporated. second edition. 2008. ISBN 184628838X.
- [12] Povey, D.; Ghoshal, A.; Boulianne, G.; et al.: The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society. Dec 2011. iEEE Catalog No.: CFP11SRW-USB.



- [13] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*. 1958: pp. 65–386.
- [14] Sak, H.; Senior, A. W.; Beaufays, F.: Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *CoRR*. vol. abs/1402.1128. 2014.
- [15] Sak, H.; Senior, A. W.; Beaufays, F.: Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, edited by H. Li; H. M. Meng; B. Ma; E. Chng; L. Xie. ISCA. 2014. pp. 338–342.
- [16] Sim, K. C.: On Constructing and Analysing An Interpretable Brain Model for the DNN Based on Hidden Activity Patterns. In *The 2015 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2015), Scottsdale, Arizona, USA, December 13-17, 2015*. IEEE. 2015.
- [17] Zbořil, F. V.: Soft Computing - Neuronové sítě Madaline a Back Propagation. 2016. [In Czech].
- [18] Černocký, H.: Zpracování řečových signálů — studijní opora. 2006. [In Czech].

# Appendices

## List of Appendices

<b>A Contents of DVD</b>	<b>48</b>
<b>B Example scripts for CNTK</b>	<b>49</b>
B.1 Activation printing script . . . . .	49

# Appendix A

## Contents of DVD

/	
— SRC_CODES	
— KALDI_SCRIPTS .....	Standard Kaldi AMI recipe scripts used for feature preparation.
— CNTK_SCRIPTS .....	Standard CNTK scripts edited for training and activation printing.
— UTILITY_SCRIPTS .....	Miscellaneous utility scripts mostly for Metacentrum set-up.
— VISUALISATION_SCRIPTS .....	Implementation of the described visualisation system.
— TECHNICAL_REPORT .....	This technical report.
— TRAINED_NEURAL_NETWORKS .....	CNTK network files and configuration files.
— BOTTLENECK	
— LSTM	
— NOISE	
— NORMAL	
— RELU	
— VISUALISATIONS	
— ARCHITECTURES_EXPERIMENT	
— NORMED	
— ORIGINAL	
— SILENCE	
— ENVIRONMENTS_EXPERIMENT	
— NORMED	
— ORIGINAL	
— SILENCE	

# Appendix B

## Example scripts for CNTK

### B.1 Activation printing script

```
#### deviceId = -1 for CPU, >= 0 for GPU devices
DeviceNumber = $DeviceNumber$
numCPUThreads=$numThreads$
command = $action$
precision = float

ACTIVATIONS=[
  action=write
  modelPath = $modelName$
  outputNodeNames = {L$nj$.S}

  # deviceId=-1 for CPU, >=0 for GPU devices
  deviceId=$DeviceNumber$
  traceLevel=1
  useValidation=true
  printValues=true

  reader=[
    # reader to use
    readerType=Kaldi2Reader
    readMethod=blockRandomize
    frameMode=false
    miniBatchMode=Partial
    randomize=None
    verbosity=0
    features=[
      dim=$featDim$
      scpFile=$inputCounts$
      rx=$inputFeats$
    ]
  ]
]
writer=[
  # reader to use
  writerType=Kaldi2Reader
  readMethod=blockRandomize
  frameMode=false
  miniBatchMode=Partial
  randomize=None
  verbosity=0
  L$nj$.S=[
    #dim=$labelDim$
    dim=2048
    Kaldicmd="ark:/storage/brno6/home/xfabry01/L$nj$.txt"
    scpFile=$inputCounts$
  ]
]
]
```