Computational Intelligence Project and laboratory summary

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

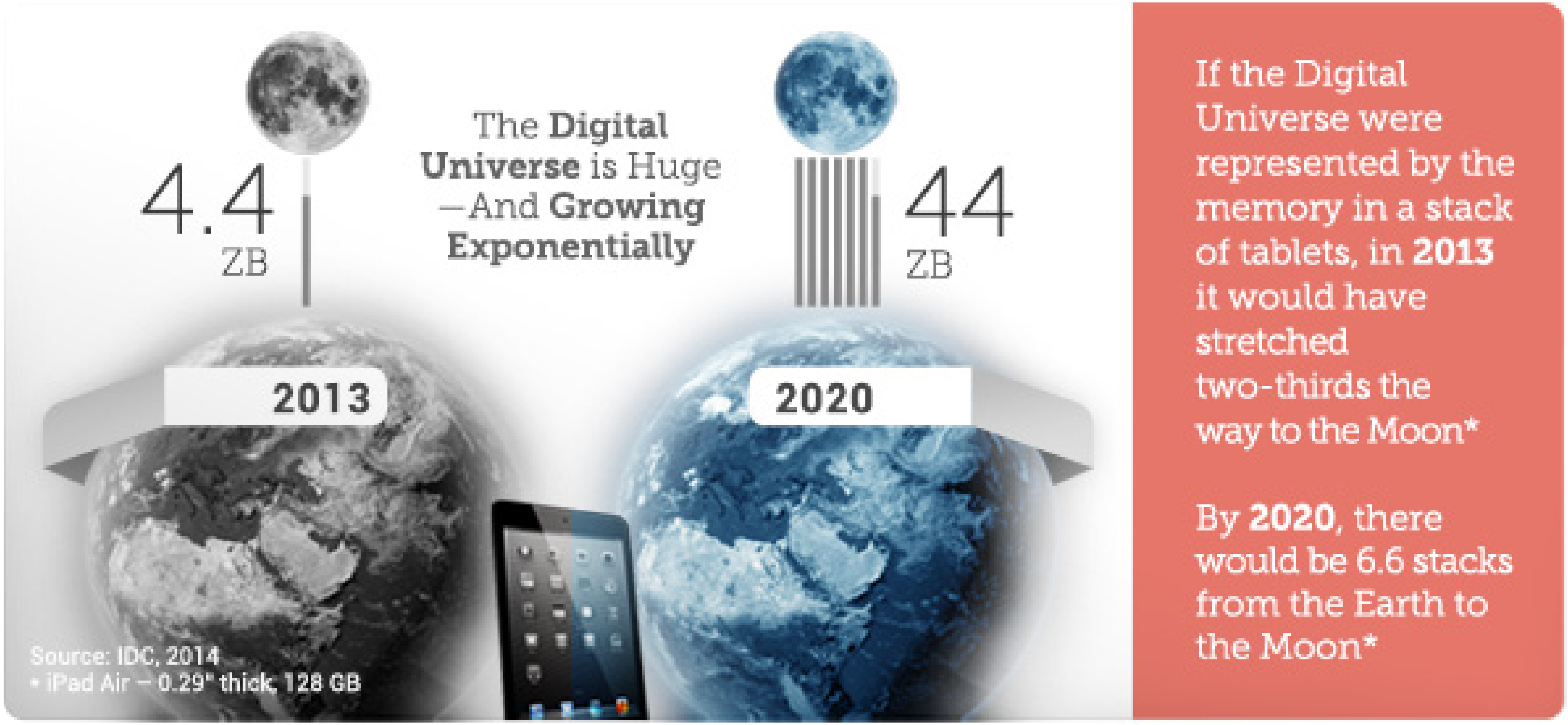
Kamil Lelowicz
Łukasz Radzio

Table of contest

Introduction

- Computational Intelligence is usually used to recognize, classify, predict in order to make decisions without human assistance or help people in the decision processes.
- We can use Computational Intelligence methods in Big data business which is one of hottest industries in the world today.
- 4.4 Zettabytes of data existed in the digital universe in 2013
- Only 0.5% of data was analyzed

https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm



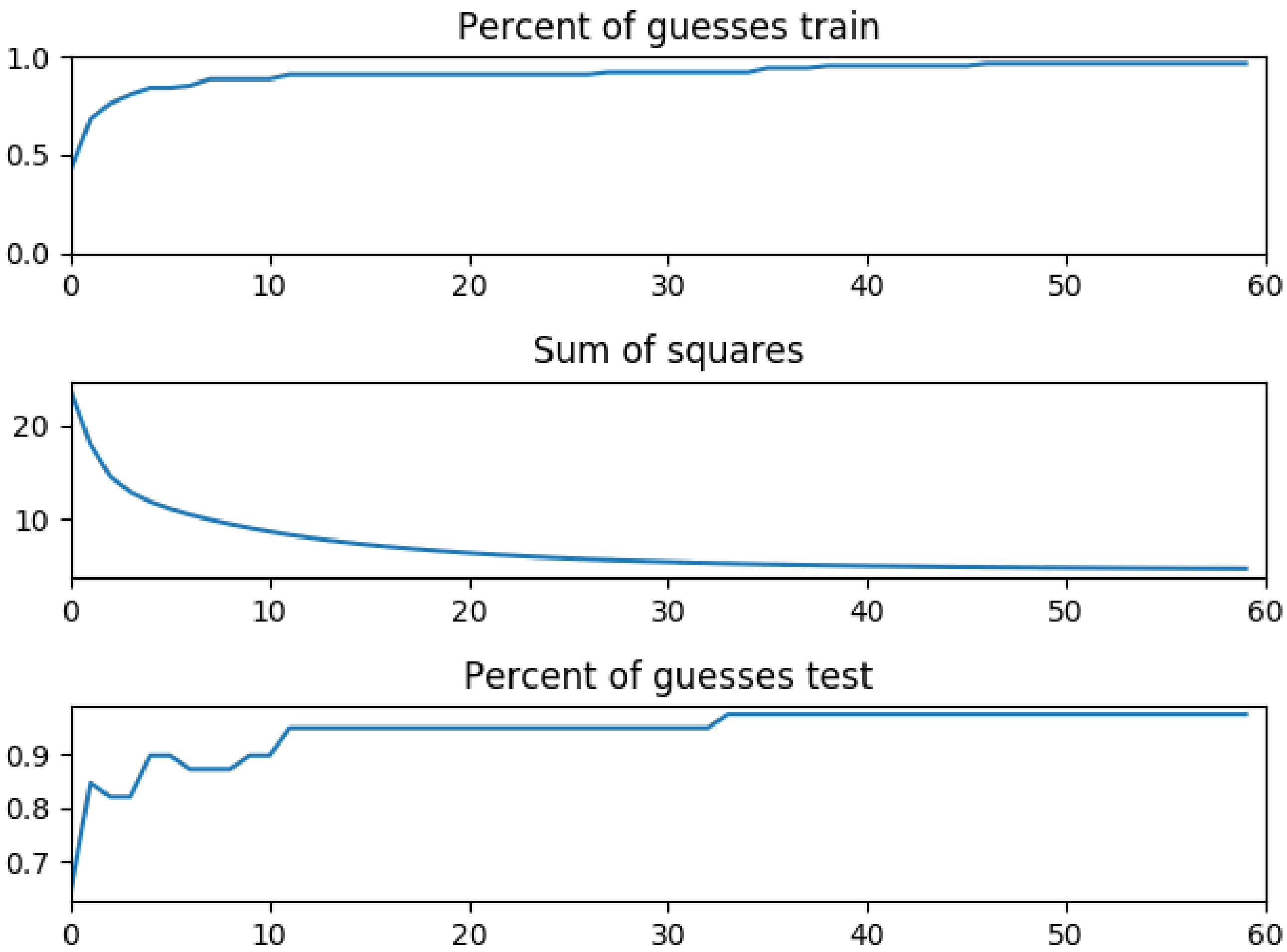
https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

Laboratory - Kamil Lelowicz

MLP with BP

- Standard algorithms for any supervised learning pattern recognition process
- Few words about implementation in Python:
  - Each neuron was implemented as a class
  - The input and error propagation phase were implemented recursively
  - Sigmoidal functions were used
  - Normalize input data

Result for Iris data



Best result:
98% for test set
100% for train set

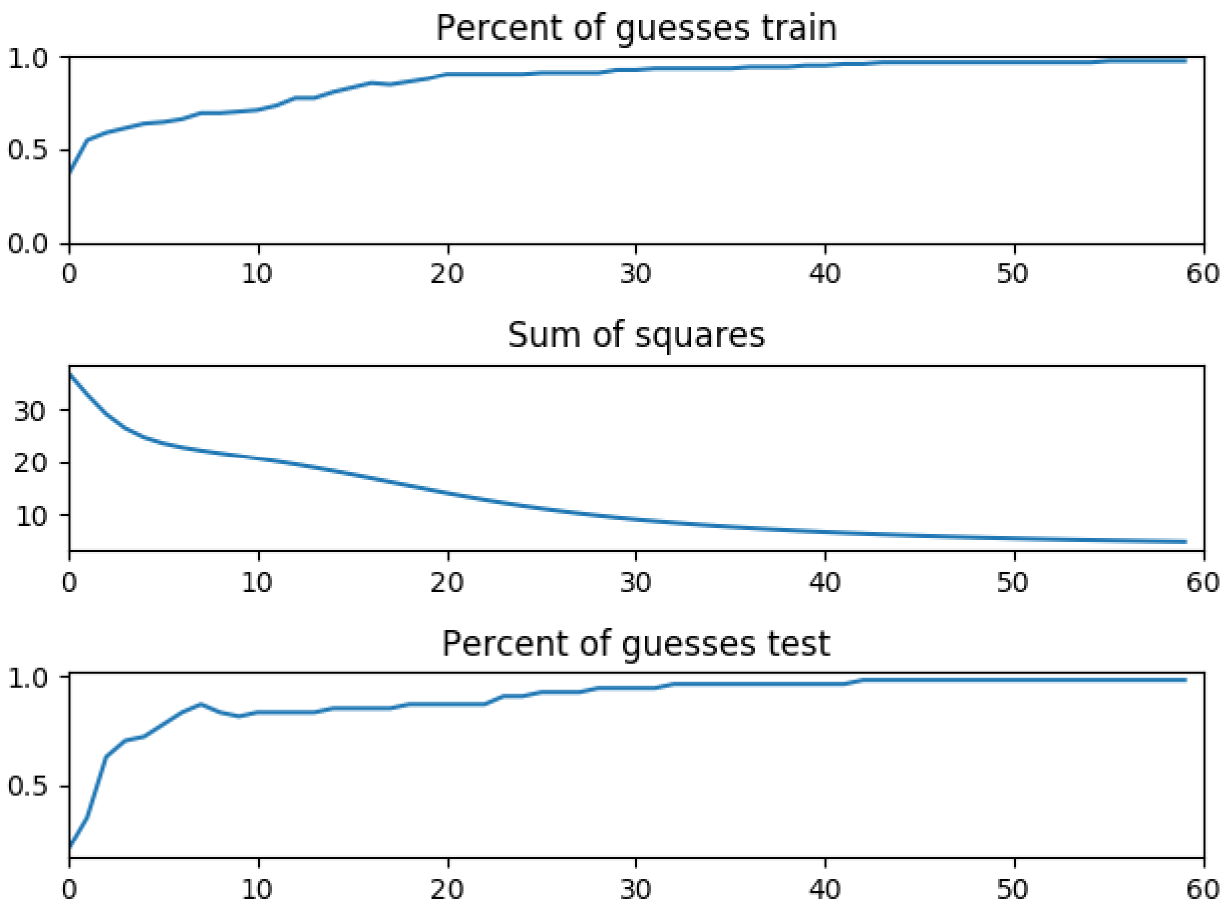
Initialization of weights:
(-1.5, 1.5)

One hidden layer

Learning rate:
0.1

During the adaptation process the learning rate was modifying

Result for Wine data



Best result:
98% for test set
99% for train set

Initialization of weights:
(-2, 2)

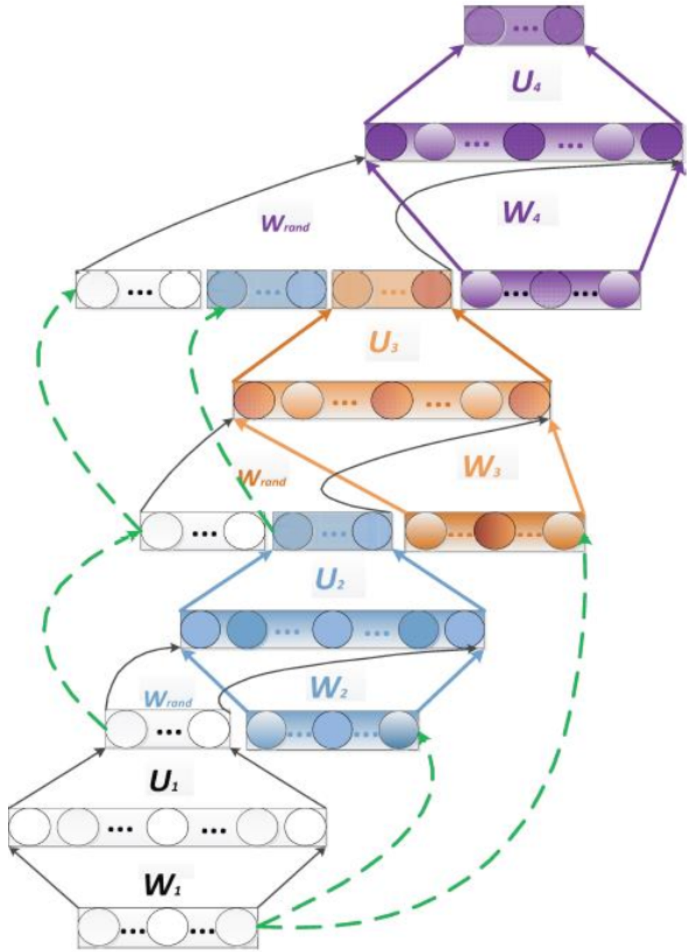
Two hidden layer

Learning rate:
0.2

During the adaptation process the learning rate was modifying

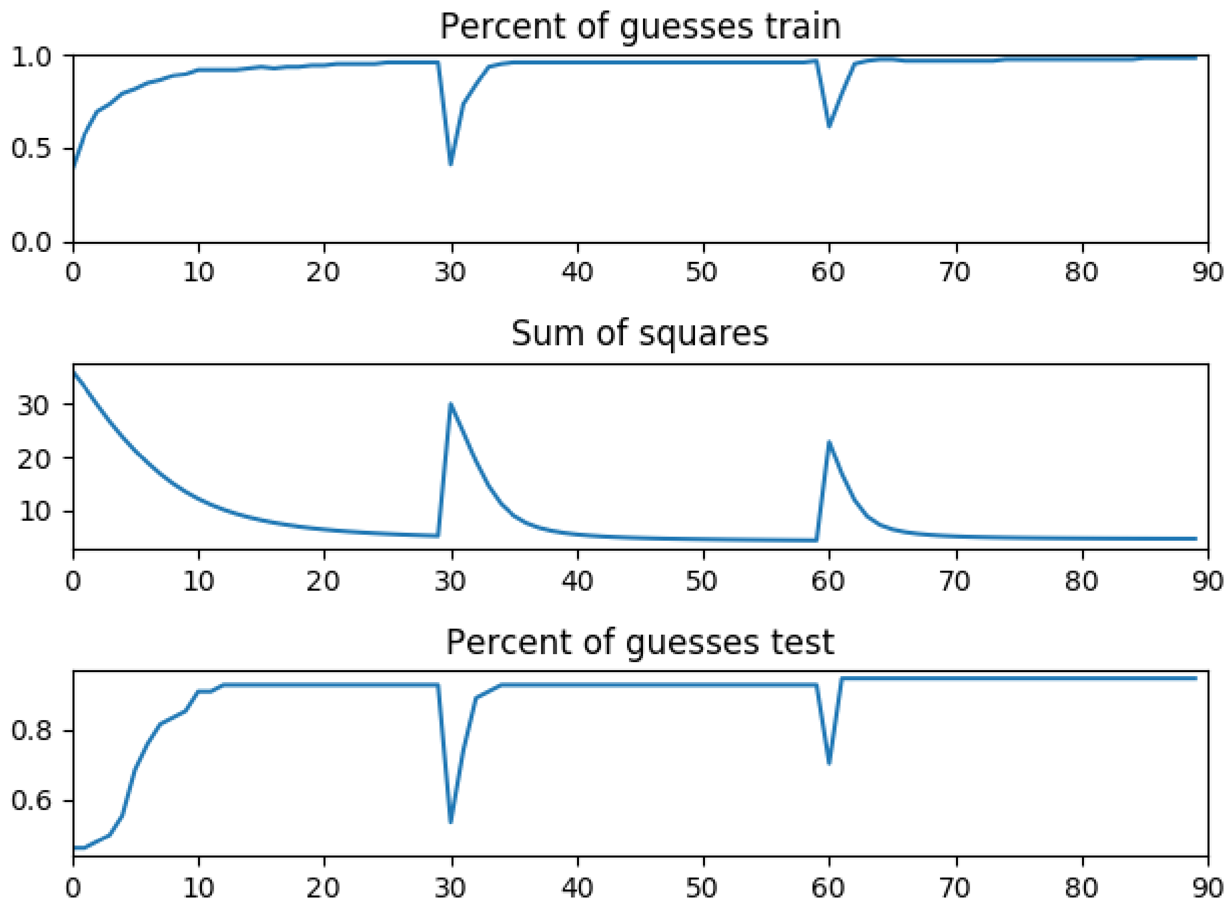
Simple Deep MLP

- Not all neurons between successive layers are connected
- Updating only a selected part of neurons
- Using many subnetworks



http://home.agh.edu.pl/~horzyk/lectures/ci/CI-DeepLearningStrategies.pdf

Result for Wine Data



Best result:
99% for test set
99.5% for train set

Initialization of weights:
(-2, 2)

One hidden layer

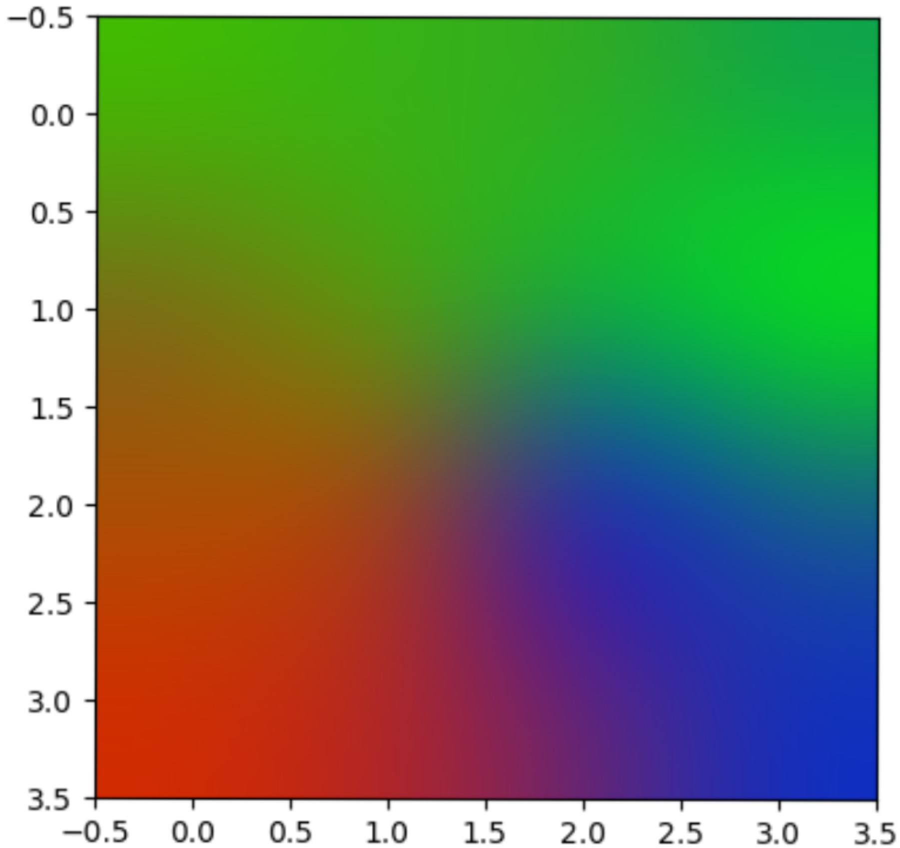
Learning rate:
0.3

During the adaptation process the learning rate was modifying

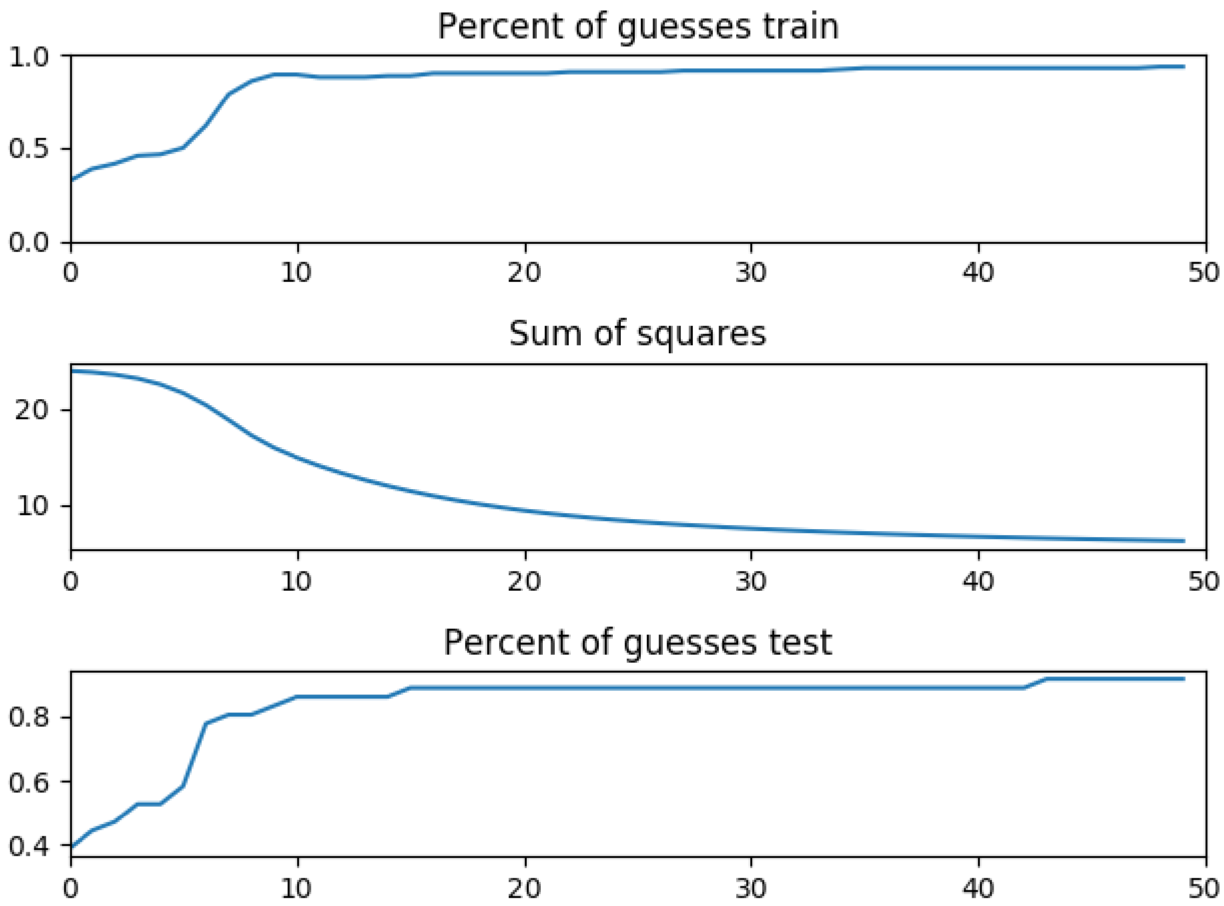
Deep SOM-MLP Classifier

- Kohonen’s SOM enable to represent multidimensional data in fewer dimensions, i.e. two or three
- Type of unsupervised learning method
- We can use Self-Organizing Maps for feature extraction

Two -dimensional representation of Wine dataset



Result for Wine data



Best result:
94% for test set
96% for train set

Initialization of weights:
(-2, 2)

Two hidden layer

Learning rate:
0.3

During the adaptation process the learning rate was modifying

AGDS

- A passive data structure, which make more faster operation for example searching similar group of elements, filtering by value or attribute
- We can substitute this operations by providing them in O(1)
- To implement it I use ordered dictionaries and quick sort algorithms

```
Most similar object to 51 is 53

List of similar objects to 51 sorted by the similarity:
[53, 74, 78, 70, 75, 98, 79, 86, 57, 62, 68, 85, 67, 71, 89, 97, 96, 84, 88, 100, 64, 95, 60, 83, 91, 93, 63, 90,
 54, 65, 58, 80, 81, 82, 99, 73, 94, 87, 61, 56, 77, 52, 76, 72, 92, 130, 140, 66, 134, 142, 69, 138, 128, 117,
 139, 148, 113, 127, 125, 103, 124, 146, 150, 108, 104, 141, 112, 145, 109, 131, 105, 147, 120, 129, 149, 55, 102,
 143, 122, 133, 137, 106, 136, 101, 110, 114, 123, 115, 107, 132, 118, 119, 17, 25, 14, 44, 20, 50, 37, 12, 22,
 24, 40, 19, 18, 59, 29, 28, 31, 38, 21, 11, 46, 4, 33, 2, 41, 1, 45, 6, 15, 32, 7, 126, 49, 34, 111, 10, 5, 8,
 13, 39, 47, 30, 9, 121, 35, 16, 27, 144, 116, 26, 135, 42, 36, 23, 3, 48, 43]


Most similar object to 53 is 51

List of similar objects to 53 sorted by the similarity:
[51, 78, 77, 76, 74, 72, 92, 59, 75, 66, 98, 86, 84, 69, 67, 71, 89, 96, 97, 88, 100, 95, 60, 83, 91, 93, 55, 63,
 90, 54, 65, 58, 80, 81, 82, 99, 73, 94, 61, 56, 52, 70, 130, 117, 79, 111, 148, 113, 57, 139, 126, 62, 146, 127,
 150, 85, 104, 103, 125, 112, 144, 108, 116, 109, 135, 105, 147, 64, 129, 131, 145, 102, 143, 133, 149, 106, 137,
 136, 114, 115, 101, 123, 107, 110, 132, 119, 118, 87, 17, 25, 134, 14, 19, 44, 138, 46, 20, 128, 2, 36, 23, 68,
 12, 37, 50, 22, 40, 34, 3, 18, 48, 29, 21, 11, 124, 33, 1, 6, 41, 39, 43, 45, 32, 15, 121, 7, 49, 30, 10, 141, 8,
 5, 35, 13, 47, 9, 120, 27, 16, 122, 26, 42, 140, 24, 142, 28, 31, 38, 4]
```

Laboratory – Łukasz Radzio

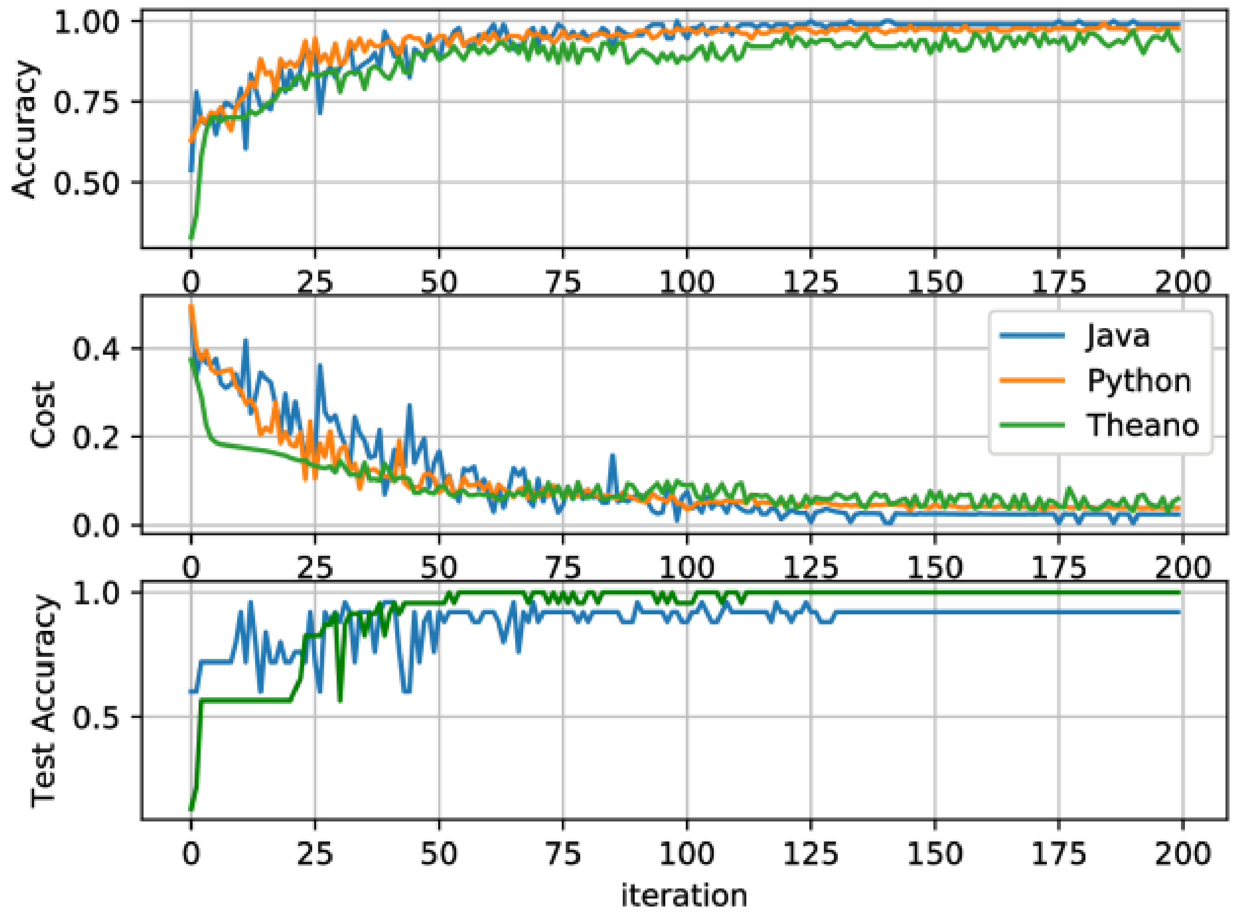
Laboratory part

1) Implementation of MLP – basic Python
2) MLP, SOM, Simple Deep – JAVA
3) AGDS – Python

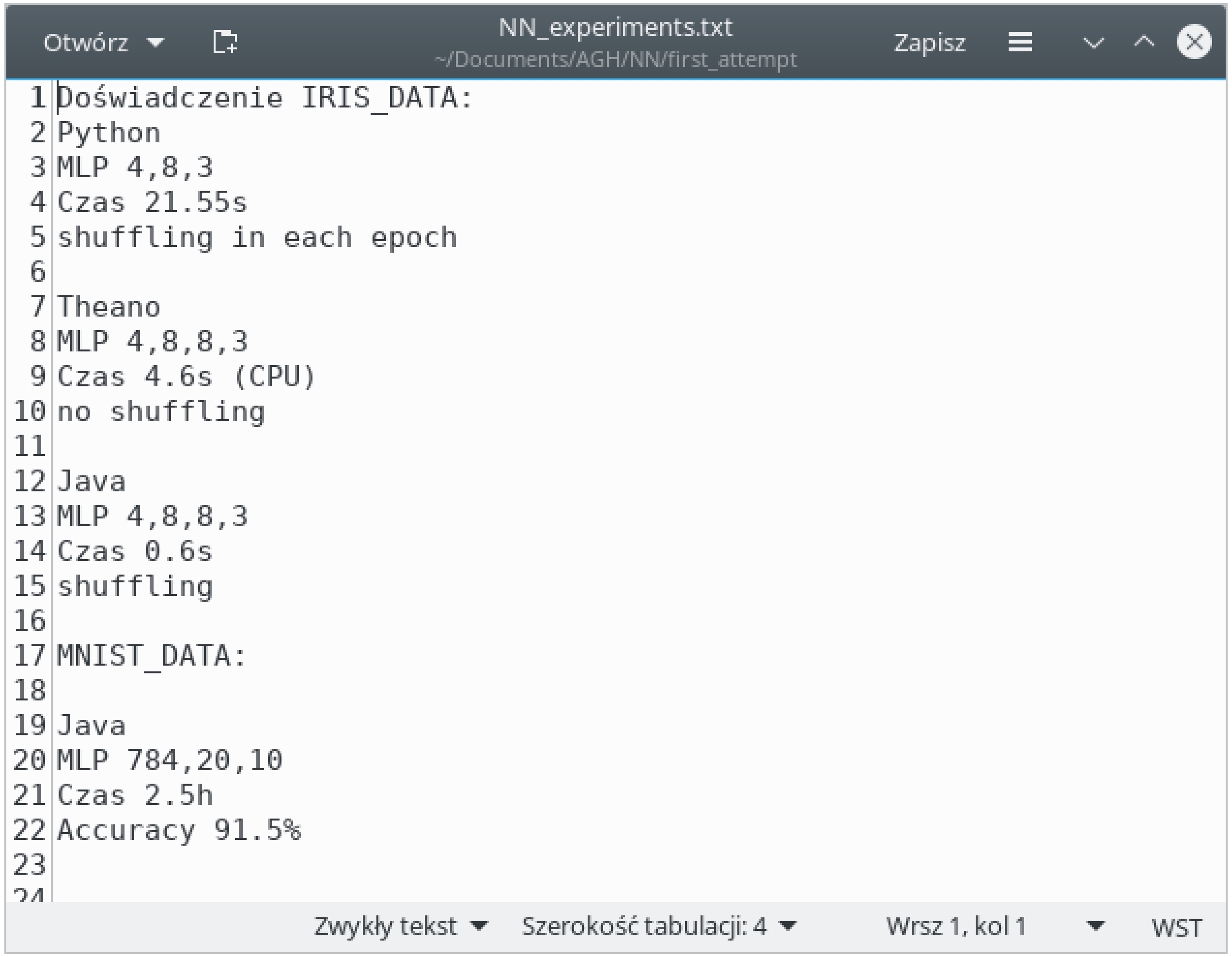
Conclusion:

1) MLP works quite good on its own
2) SOM produces nice pictures from random noise, but does not help in classification
3) Simple Deep as the name suggests is simple and useless (in case of analyzed datasets)
4) AGDS simply works (as expected)
5) JAVA implementation is unscalable to bigger datasets
6) Serious algorithms should be implemented under GPU

Comparison of various implementations of MLP - Iris



Experiments



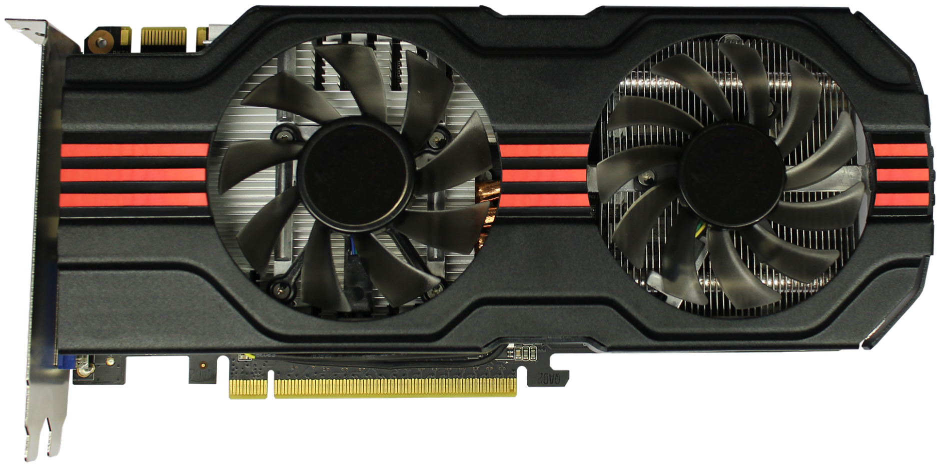

```
Doświadczenie IRIS_DATA:
Python
MLP 4,8,3
Czas 21.55s
shuffling in each epoch

Theano
MLP 4,8,8,3
Czas 4.6s (CPU)
no shuffling

Java
MLP 4,8,8,3
Czas 0.6s
shuffling

MNIST_DATA:

Java
MLP 784,20,10
Czas 2.5h
Accuracy 91.5%
```

Project

Deep Convolutional Neural Networks

Main aim

Use deep convolutional neural network on gpu for image recognition



Used technologies

- Python 3
- Theano framework
- Keras framework
- Cuda Toolkit (with cuDNN)
- Opencv (for visualization)

Quick guide how to install Theano with GPU support on Windows

- Download and install Cuda Toolkit v8.0 link
- Download CuDNN v5.1 (nvidia website)
- Download and install VC 14.0 link
- Install Anaconda (or miniconda)
- Install packages from Anaconda (numpy, scipy, mkl-service, libpython, m2w64-toolchain, nose, nose-parameterized, pydot-ng, theano, pygpu)
- Configure .theanorc file in your home directory

.theanorc

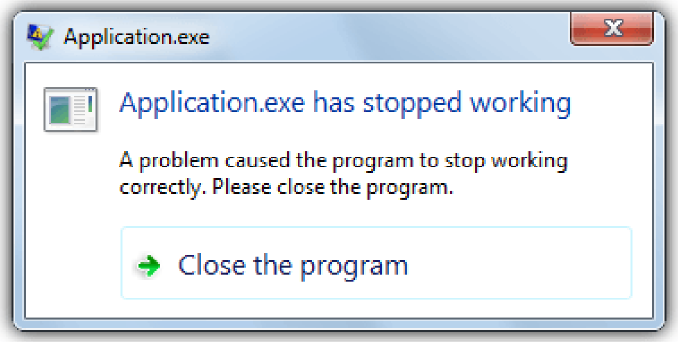
```
[global]
floatX = float32
device = cuda
cxx = c:\Users\Kamil\Anaconda3\Library\mingw-w64\bin\g++.exe

[cuda]
root = C:\CUDA\v8.0\

[dnn]
include_path=c:\Users\Kamil\Downloads\cudnn-8.0-windows7-x64-v5.1\cuda\include
library_path=c:\Users\Kamil\Downloads\cudnn-8.0-windows7-x64-v5.1\cuda\lib\x64

[nvcc]
fastmath = True
compiler_bindir=C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\cl.exe
```

Tips

» Theano-cache purge

» If the program stopped and no error reported, your compiler will probably be different from the compiler used to compile libgpuarray

» Not recommended ways simply don't work



Comparison gpu and cpu

- Gpu is over 12 times faster than cpu during computing
- Nvida GT 650M is over 4 times faster than GT 820M

GPU-CPU comparison

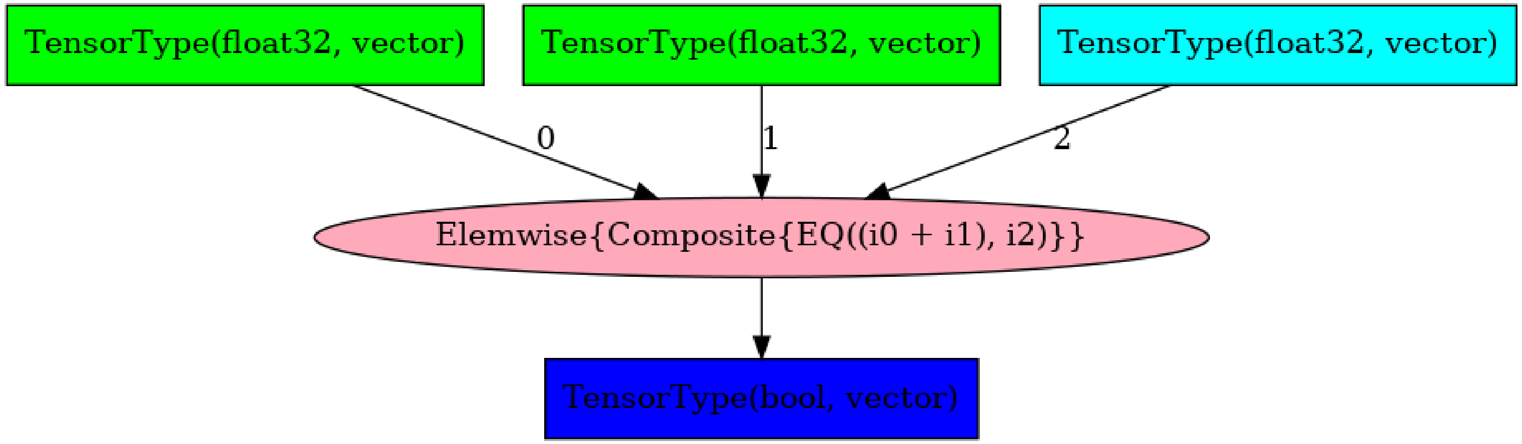
```
46900/46900 [==============================] - 915s - loss: 1.0484 - acc: 0.7471     7
Epoch 2/20
 1536/46900 [..............................] - ETA: 890s - loss: 0.3927 - acc: 0.8952
```

```
Epoch 1/20
46900/46900 [==============================] - 75s - loss: 1.1773 - acc: 0.6788
Epoch 2/20
46900/46900 [==============================] - 74s - loss: 0.3014 - acc: 0.9108
Epoch 3/20
46900/46900 [==============================] - 74s - loss: 0.2149 - acc: 0.9354
Epoch 4/20
46900/46900 [==============================] - 74s - loss: 0.1700 - acc: 0.9489
Epoch 5/20
46900/46900 [==============================] - 74s - loss: 0.1422 - acc: 0.9576
Epoch 6/20
46900/46900 [==============================] - 74s - loss: 0.1226 - acc: 0.9635
Epoch 7/20
46900/46900 [==============================] - 74s - loss: 0.1071 - acc: 0.9679
Epoch 8/20
46900/46900 [==============================] - 74s - loss: 0.0960 - acc: 0.9709
Epoch 9/20
46900/46900 [==============================] - 74s - loss: 0.0872 - acc: 0.9739
Epoch 10/20
46900/46900 [==============================] - 74s - loss: 0.0792 - acc: 0.9757
Epoch 11/20
46900/46900 [==============================] - 74s - loss: 0.0737 - acc: 0.9776
Epoch 12/20
46900/46900 [==============================] - 74s - loss: 0.0686 - acc: 0.9793
Epoch 13/20
29440/46900 [=================>............] - ETA: 27s - loss: 0.0631 - acc: 0.9814
```

Theano framework

- tensors (multidimensional arrays)
- computational graphs
- interface similar to numpy
- computations on GPU
- automatic differentiation

Theano framework – graph

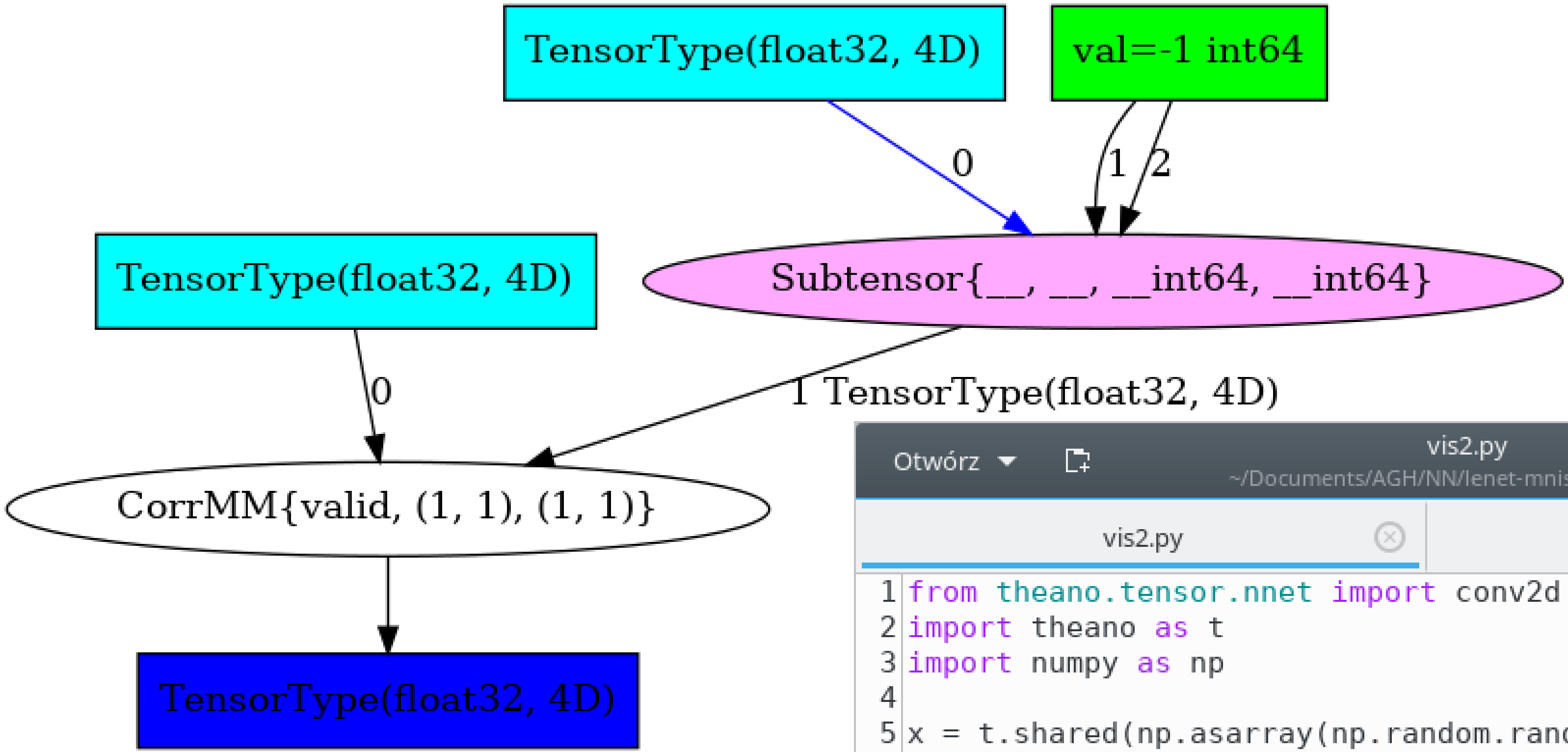


vis.py
~/Documents/AGH/NN/lenet-mnist/prezentacja

```python
import theano as t
import numpy as np

x = t.tensor.vector()
y = t.tensor.vector()
z = x + y

z_check = t.shared(np.asarray([7,3],dtype='float32'))
check = t.tensor.eq(z,z_check)
sum_v = t.function([x,y],[check])

print(sum_v([2,1],[5,2]))
t.printing.pydotprint(sum_v, outfile="vis_test.png")
```

Otwórz Zapisz Python Szerokość tabulacji: 4 Wrsz 1, kol 1 WST

Theano - convolution



vis2.py — ~/Documents/AGH/NN/lenet-mnist/prezentacja

vis2.py | theano_net.py

```python
from theano.tensor.nnet import conv2d
import theano as t
import numpy as np

x = t.shared(np.asarray(np.random.random
((4,3,100,100)),dtype='float32'),borrow=True)
f = t.shared(np.asarray(np.random.random
((10,3,7,7)),dtype='float32'),borrow=True)
z = conv2d(input=x,filters=f,input_shape=(4,3,100,100))

co = t.function([],z)
co_shape = np.array(co()).shape
print(co_shape)
t.printing.pydotprint(co, outfile="vis2_test.png")
```

Python | Szerokość tabulacji: 4 | Wrsz 12, kol 53 | WST

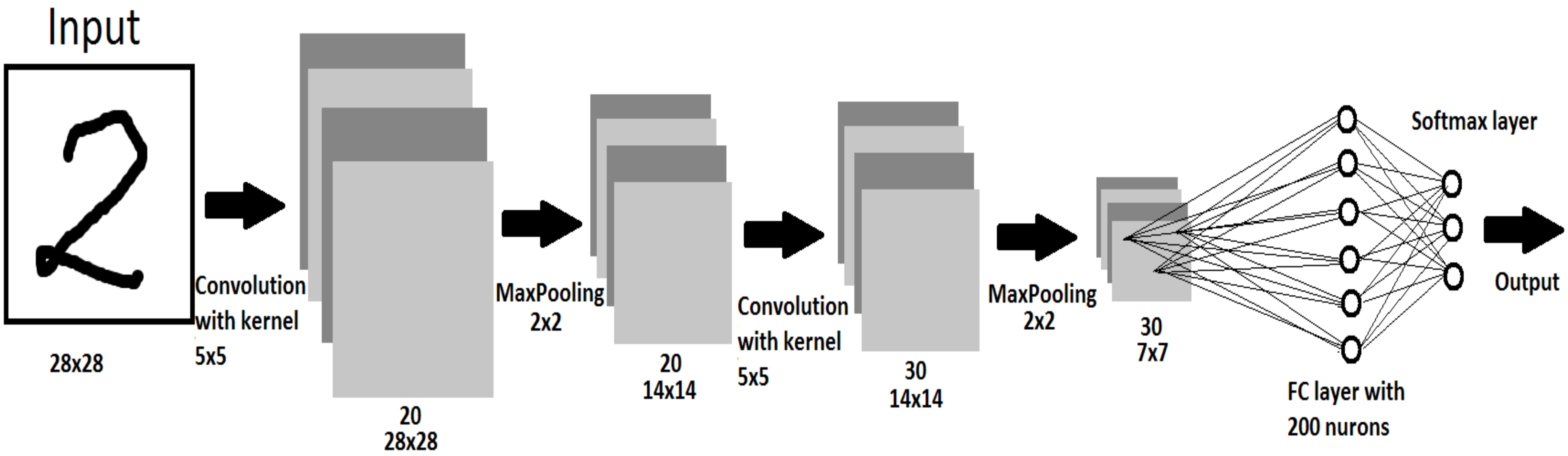
Network topology

» Two convolutional layer with ReLu activation function na MaxPooling

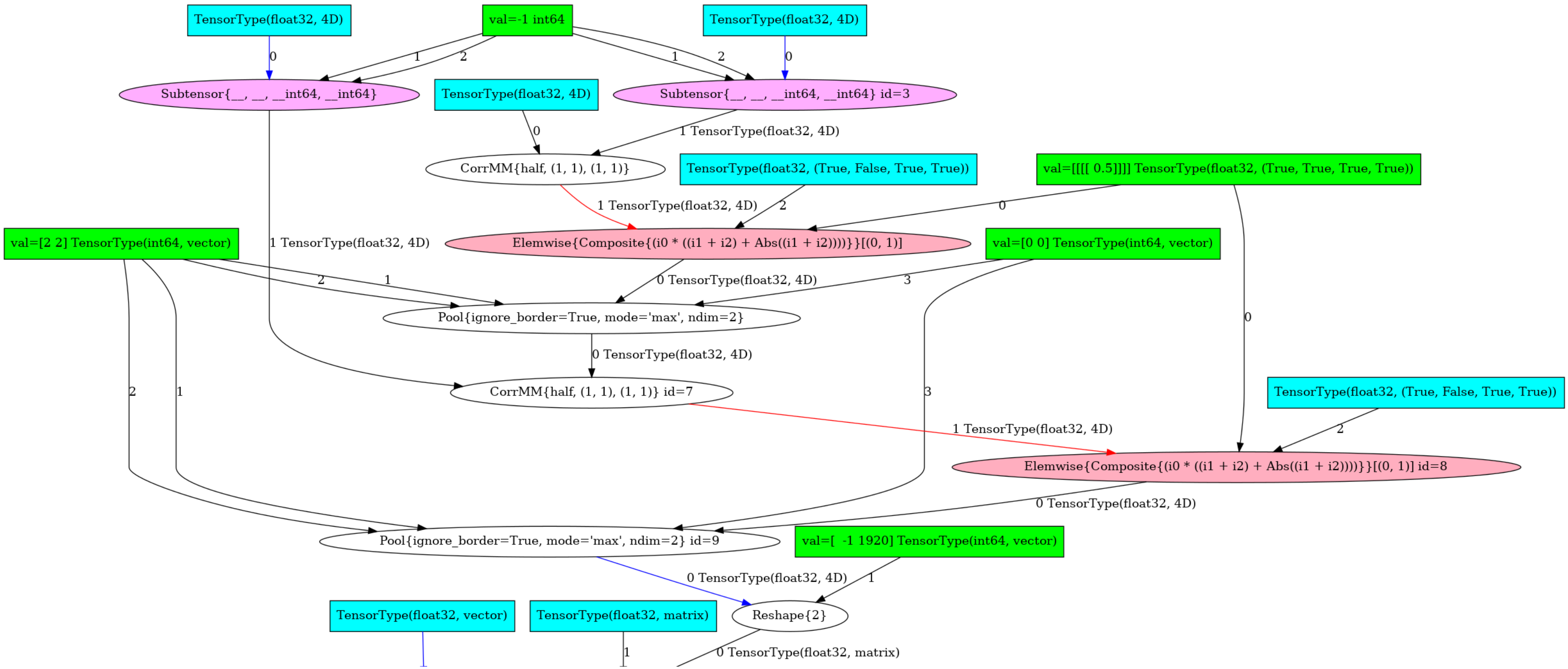
» One FC layer

» The last layer was softmax

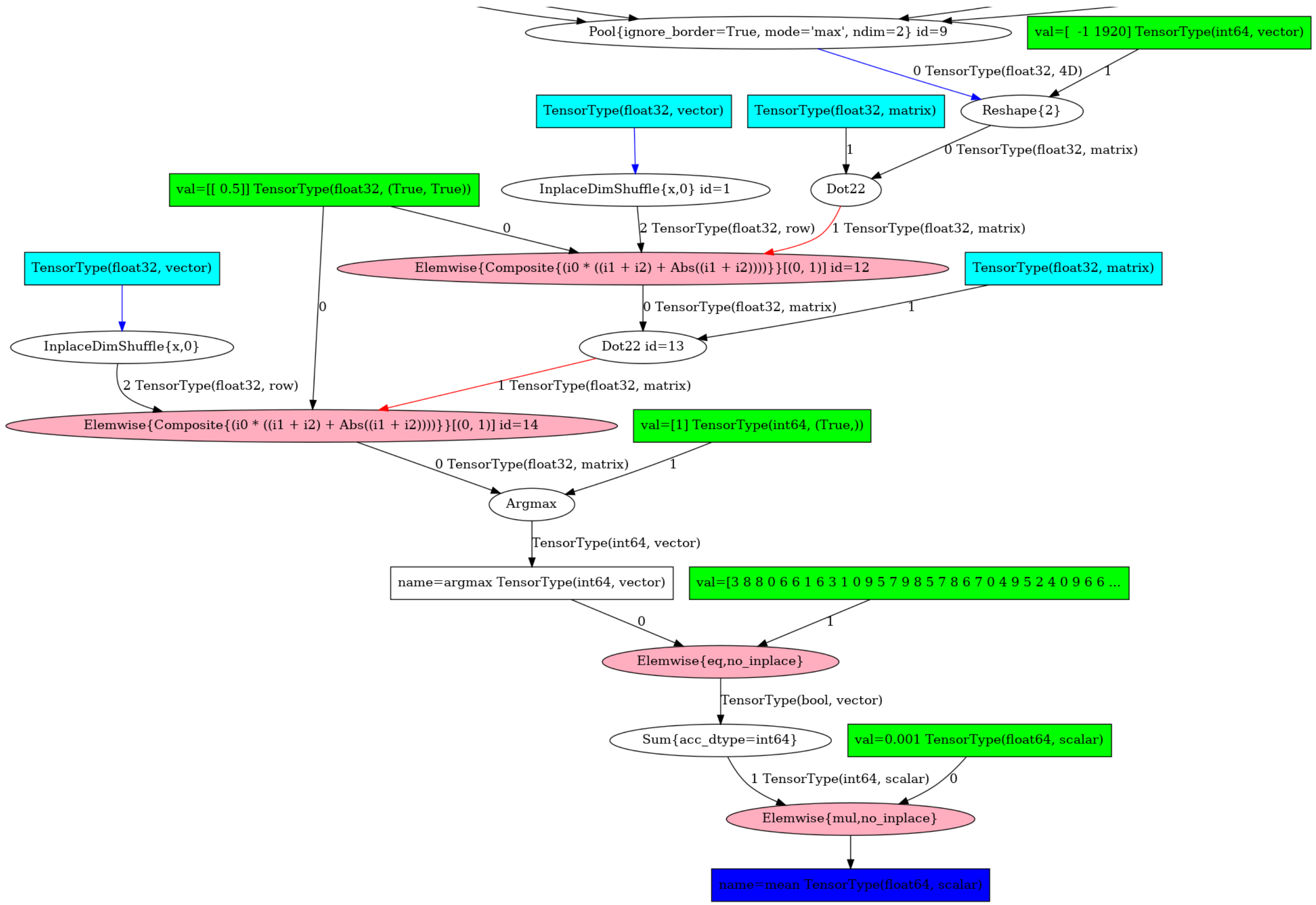
Network topology



Graph of our network



Graph of our network



Keras code example

Theano code example

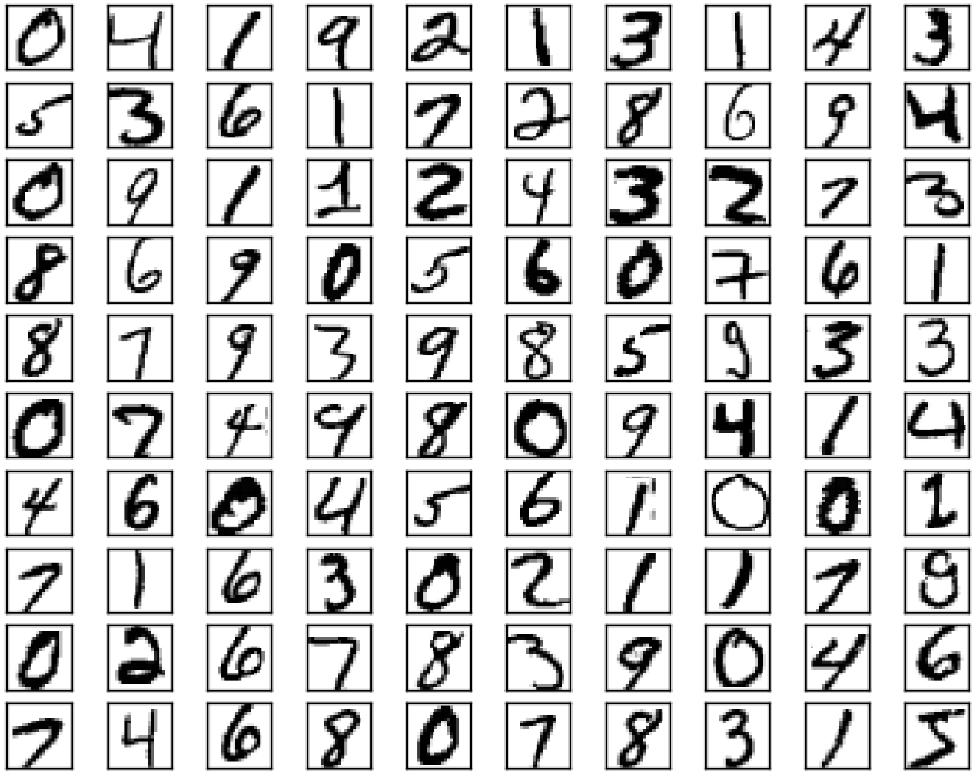
Data sets

- Mnist
- Cifar 10

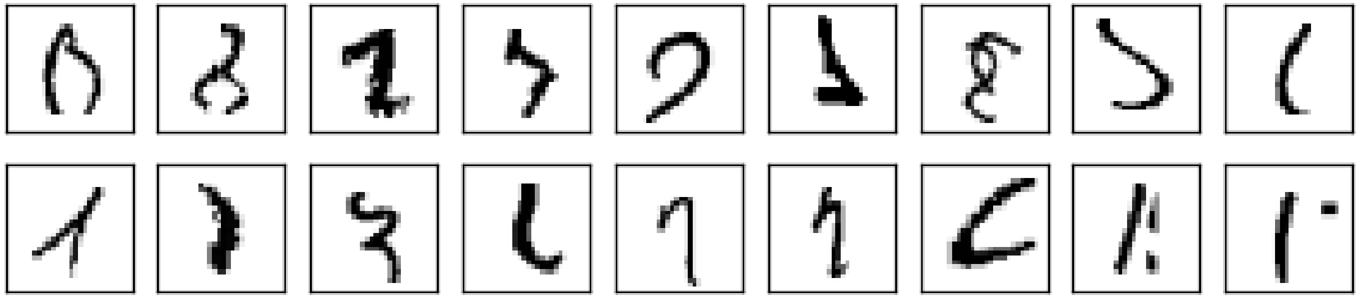
Mnist

- Handwritten digits from 0 to 9 (black and white)
- 28x28 pixels (784 inputs)
- Training set 50 000 examples
- Test set 10 000 examples
- Web page: http://yann.lecun.com/exdb/mnist/

Mnist examples



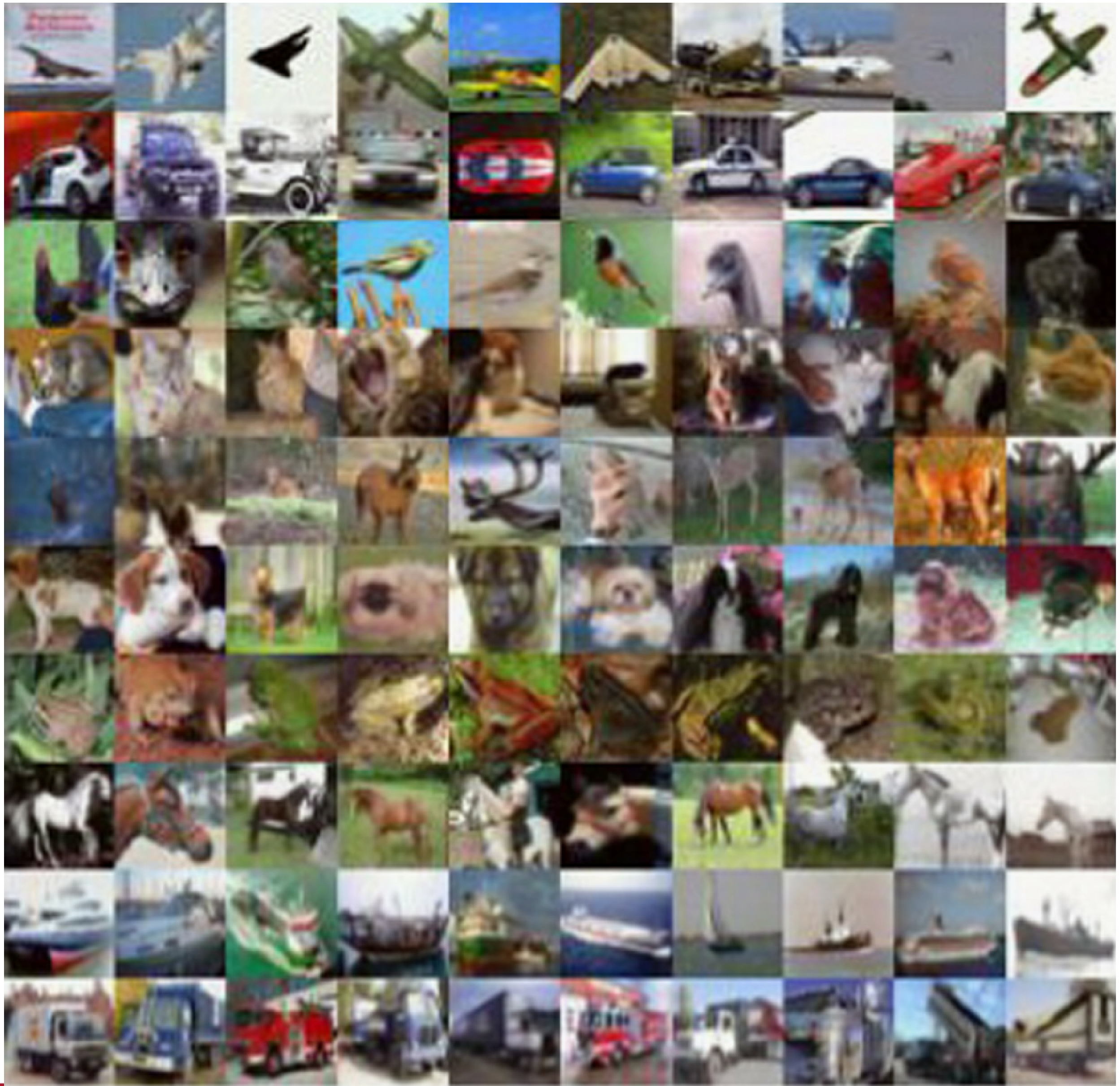
Mnist really bad images



Cifar-10

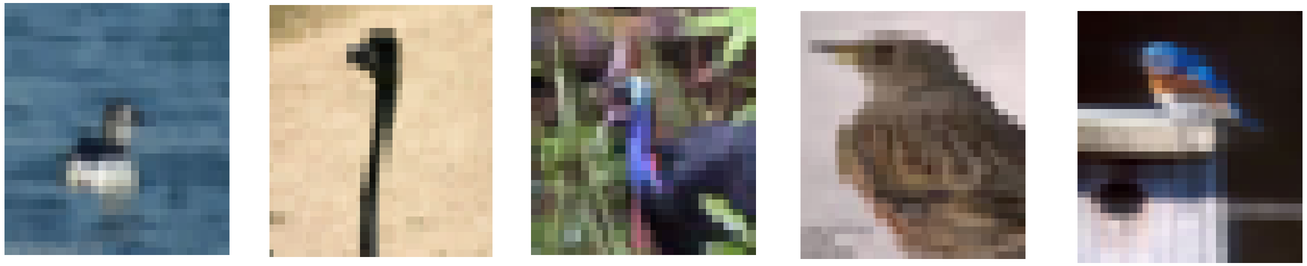
- The CIFAR-10 is labeled subsets of the 80 million tiny images dataset
- 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, track)
- "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.
- 32x32 colour image (3072 inputs)
- 50000 training images
- 10000 test images
- Web page: https://www.cs.toronto.edu/~kriz/cifar.html

Cifar - 10



Difference between images in one class

Birds



Cars

Achieved result in Mnist database

- The training took half- day
- Best result on test set is 99.5%
- Comparison

Achieved result in Cifar-10 database

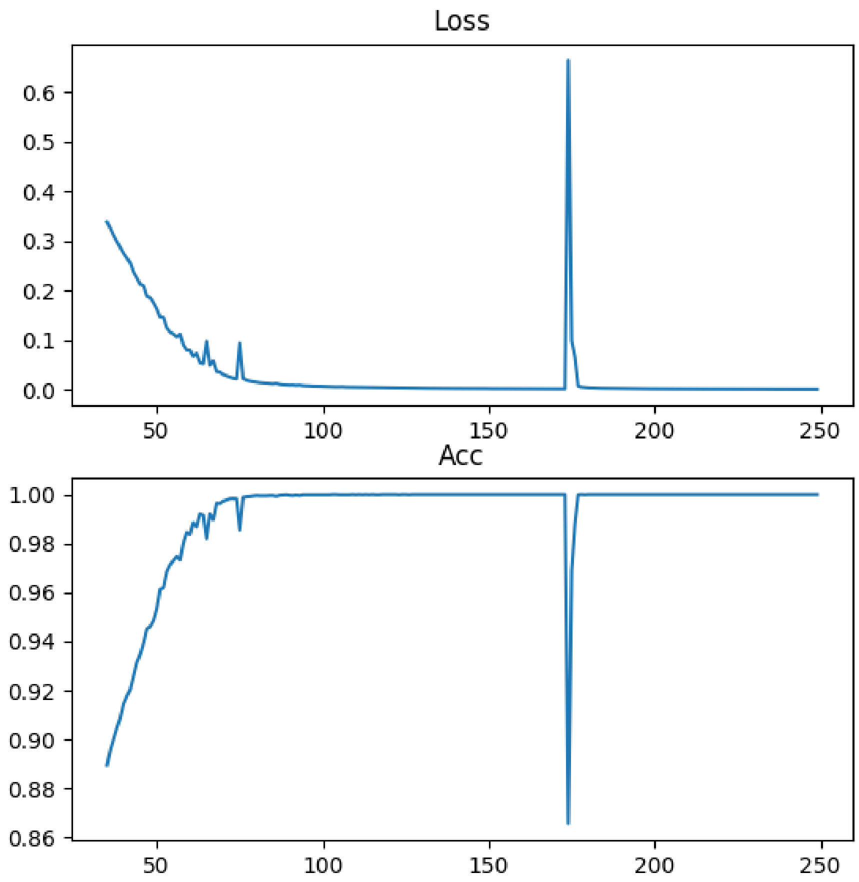
» The training took half- day

» Best result on test set is 77.6%

» It is very easy to overfit

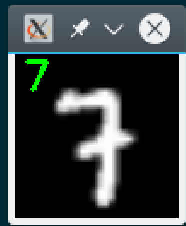
» comparison

Learning process – cifar10 - overfitting

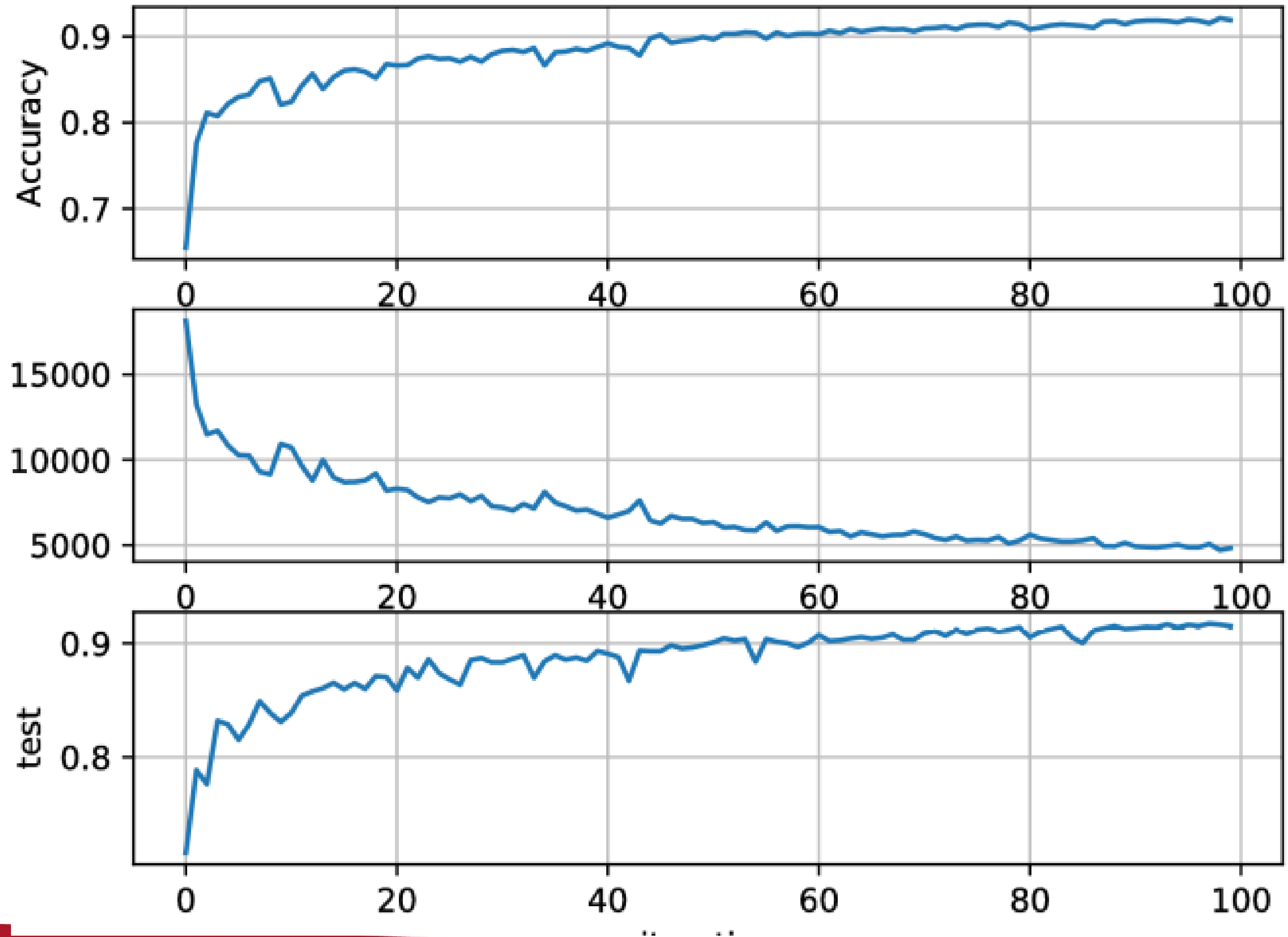


Whole learning process MNIST

```
[INFO] compiling model...
[INFO] training...
Epoch 1/20
46900/46900 [==============================] - 75s - loss: 1.1773 - acc: 0.6788
Epoch 2/20
46900/46900 [==============================] - 74s - loss: 0.3014 - acc: 0.9108
Epoch 3/20
46900/46900 [==============================] - 74s - loss: 0.2149 - acc: 0.9354
Epoch 4/20
46900/46900 [==============================] - 74s - loss: 0.1700 - acc: 0.9489
Epoch 5/20
46900/46900 [==============================] - 74s - loss: 0.1422 - acc: 0.9576
Epoch 6/20
46900/46900 [==============================] - 74s - loss: 0.1226 - acc: 0.9635
Epoch 7/20
46900/46900 [==============================] - 74s - loss: 0.1071 - acc: 0.9679
Epoch 8/20
46900/46900 [==============================] - 74s - loss: 0.0960 - acc: 0.9709
Epoch 9/20
46900/46900 [==============================] - 74s - loss: 0.0872 - acc: 0.9739
Epoch 10/20
46900/46900 [==============================] - 74s - loss: 0.0792 - acc: 0.9757
Epoch 11/20
46900/46900 [==============================] - 74s - loss: 0.0737 - acc: 0.9776
Epoch 12/20
46900/46900 [==============================] - 74s - loss: 0.0686 - acc: 0.9793
Epoch 13/20
46900/46900 [==============================] - 74s - loss: 0.0636 - acc: 0.9805
Epoch 14/20
46900/46900 [==============================] - 74s - loss: 0.0605 - acc: 0.9823
Epoch 15/20
46900/46900 [==============================] - 74s - loss: 0.0577 - acc: 0.9822
Epoch 16/20
46900/46900 [==============================] - 74s - loss: 0.0541 - acc: 0.9835
Epoch 17/20
46900/46900 [==============================] - 74s - loss: 0.0512 - acc: 0.9844
Epoch 18/20
46900/46900 [==============================] - 74s - loss: 0.0491 - acc: 0.9845
Epoch 19/20
46900/46900 [==============================] - 74s - loss: 0.0471 - acc: 0.9857
Epoch 20/20
46900/46900 [==============================] - 74s - loss: 0.0455 - acc: 0.9858
[INFO] evaluating...
23040/23100 [============================>.] - ETA: 0s[INFO] accuracy: 98.15%
[INFO] Predicted: 7, Actual: 7
[INFO] Predicted: 2, Actual: 2
[INFO] Predicted: 8, Actual: 8
[INFO] Predicted: 7, Actual: 7
```



MNIST learning process visualization



Cost functions

The learning slowdown problem: We've now built up considerable familiarity with softmax layers of neurons. But we haven't yet seen how a softmax layer lets us address the learning slowdown problem. To understand that, let's define the *log-likelihood* cost function. We'll use x to denote a training input to the network, and y to denote the corresponding desired output. Then the log-likelihood cost associated to this training input is

$$C \equiv -\ln a_y^L. \tag{80}$$

cross-entropy

Quadratic function

Achievments

- implementation of convolutional neural network in Keras
- implementation of convolutional neural network in Theano
- visualization of results in openCV
- adaptation of mnielsen's repo to work under python3 and gpu
- finding bug in theano code, issue