Evolving Architectures of Deep Neural Networks

Petra Vidnerová

Institute of Computer Science
The Czech Academy of Sciences

Hora Informaticae 2018
Outline

Evolving architectures

- Introduction
  - Deep Neural Networks, KERAS library, Related Work
- Our Approach
  - Evolution Strategies, Individuals, Genetic Operators
- Experiments
  - Sensor Data Set, MNIST

Deep RBF Networks

- Introduction
  - RBF Networks, Adversarial Examples
- Deep Neural Network with RBF Layer
- Experiments
Introduction

Deep Neural Networks

- neural networks with more hidden layers
- convolutional networks - convolutional layers
- our work: feed-forward neural networks, fully connected

Network Architecture

- typically designed by humans
- trial and error method
- our goal: automatic design
Related Work

- quite many attempts on architecture optimisation via evolutionary process (NEAT, HyperNEAT, COSyNE)
- neuroevolution - evolving both topology and weights

- various methods to represent the architecture by genome
  - **direct encoding**
    - binary encoding (Dasgupta and McGregor, 1992, Structured Genetic Algorithm)
    - graph encoding (Pujol and Poli, 1997, Parallel Distributed Genetic Programming)
  - **indirect encoding**
    - genoms are programs written in specialized graph transformation language
    - Gruau, 1993, cellular encoding
Related Work

NEAT - NeuroEvolution of Augmenting Topologies

- Ken Stanley, 2002,
  www.cs.ucf.edu/~kstanley/neat.html
Related Work

Deep Neural Networks

- architecture optimisation for DNN is very time consuming
- works focus on parts of network design
    number of layers fixed, only optimised number of neurons in individual layers, dropout rates, learning rates
    architecture is fixed, only a small controller evolved
Related Work

- optimising deep learning architectures through evolution
  - DeepNEAT - extending NEAT do deep networks, nodes are layers
  - CoDeepNEAT - two coevolving populations, one of modules, one of blueprints
Related Work

- Google: neural network for neural network design
- NLP task
- artificially designed network gives better results than the one designed by humans
KERAS Library

- widely used tool for practical applications of DNNs

```python
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```
Our Approach

Keep the search space as simple as possible.

- only architecture is optimized, weights are learned by gradient based technique
- the approach is inspired by and designed for KERAS library
- architecture defined as list of layers, each layer fully connected with next layer (dense layers)
- layer defined by number of neurons, activation function, type of regularization

- future work: add convolutional and max-pooling layers
- metaparameters of learning algorithm (type of algorithm, learning rate, etc.)
Evolutionary Algorithms

- robust optimisation techniques
- work with population of *individuals* representing feasible solutions
- each individual has assigned a *fitness* value
- population evolves by means of *selection, crossover, and mutation*

Evolution Strategies

- initially designed for problems with continuous attributes
- Gaussian mutation is the key operator
- \((n + m)\)-ES or \((n, m)\)-ES
Evolution Strategies

t = 0
initialize(P(t)) # n individuals
evaluate(P(t))

while not terminating_criterion do
    P_c(t) ← reproduce(m, P(t))
evaluate(P_c(t))
    if PlusStrategy then
        P(t + 1) ← P_c(t) ∪ P(t)
    else
        P(t + 1) ← P_c(t)
    end if
    P(t + 1) ← selectBest(n, P(t + 1))
t ← t + 1
end while

Gaussian mutation

σ_i ← σ_i · (1 + α · N(0, 1))
\( x_i \leftarrow x_i + \sigma_i \cdot N(0, 1) \)
Individuals for Keras Architectures

- individual - deep neural network architecture

\[ I = ( [size_1, drop_1, act_1, \sigma_{\text{size}}^1, \sigma_{\text{drop}}^1], \ldots ,
\]
\[ [size_H, drop_H, act_H, \sigma_{\text{size}}^H, \sigma_{\text{drop}}^H]_H ), \]

\[ H \ldots \text{number of hidden layers} \]
\[ size_i \ldots \text{size of layer} \]
\[ drop_i \ldots \text{dropout rate} \]
\[ act_i \ldots \text{activation function} \]
\[ \sigma_{\text{size}}^i, \sigma_{\text{drop}}^i \ldots \text{strategy parameters} \]

- output layer is softmax or linear (classification or regression task)
Crossover

- one-point crossover working on the whole blocks (layers)

Parents:

\[ I_{p1} = (B_{p1}^{p1}, B_{p1}^{p2}, \ldots, B_{k}^{p1}) \]

\[ I_{p2} = (B_{p2}^{p2}, B_{p2}^{p2}, \ldots, B_{l}^{p2}) \]

Offspring:

\[ I_{o1} = (B_{p1}^{p1}, \ldots, B_{cp1}^{p1}, B_{cp2+1}^{p2}, \ldots, B_{l}^{p2}) \]

\[ I_{o1} = (B_{p2}^{p2}, \ldots, B_{cp2}^{p2}, B_{cp1+1}^{p1}, \ldots, B_{k}^{p1}) \]
Mutation

random changes to the individual

Roulette wheel selection of:

- mutateLayer - modifies one randomly selected layer
- addLayer - adds one random layer
- delLayer - deletes one random layer

mutateLayer

- change layer size \ldots Gaussian mutation
- change dropout \ldots Gaussian mutation
- change activation \ldots random choice
Fitness and Selection

Fitness Evaluation

- create network defined by individual
- evaluate crossvalidation error on trainset
- KFold crossvalidation
- for each fold train network using gradient based technique

Tournament selection

- k individuals selected at random, the best one selected for reproduction
Experiment 1: Sensor Data

Target application - Air Pollution Prediction

- a real-world data set from the application area of sensor networks for air pollution monitoring
- concentration of several gas pollutants
- 8 input values - 5 sensors, temperature, absolute and relative humidity
- 1 predicted value - concentration of CO, NO2, NOx, C6H6, and NMHC
**Sensor Data Set**

- First task - whole time period divided into five intervals, one for training, the rest for testing
- Second task - data for training and testing selected at random

<table>
<thead>
<tr>
<th>Task</th>
<th>First experiment train set</th>
<th>First experiment test set</th>
<th>Second experiment train set</th>
<th>Second experiment test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>1469</td>
<td>5875</td>
<td>4896</td>
<td>2448</td>
</tr>
<tr>
<td>NO2</td>
<td>1479</td>
<td>5914</td>
<td>4929</td>
<td>2464</td>
</tr>
<tr>
<td>NOx</td>
<td>1480</td>
<td>5916</td>
<td>4931</td>
<td>2465</td>
</tr>
<tr>
<td>C6H6</td>
<td>1799</td>
<td>7192</td>
<td>5994</td>
<td>2997</td>
</tr>
<tr>
<td>NMHC</td>
<td>178</td>
<td>709</td>
<td>592</td>
<td>295</td>
</tr>
</tbody>
</table>
**Parameter setup**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main GA</strong></td>
<td></td>
</tr>
<tr>
<td>( n )</td>
<td>(n,m) ES</td>
</tr>
<tr>
<td>( m )</td>
<td>(n,m) ES</td>
</tr>
<tr>
<td>( ng )</td>
<td>number of generations</td>
</tr>
<tr>
<td>( p_{cx} )</td>
<td>crossover probability</td>
</tr>
<tr>
<td>( p_{mut} )</td>
<td>mutation probability</td>
</tr>
<tr>
<td>( n_{layers} )</td>
<td>max number of layers</td>
</tr>
<tr>
<td>( max_lsize )</td>
<td>max layer size</td>
</tr>
<tr>
<td>( min_lsize )</td>
<td>minimum layer size</td>
</tr>
<tr>
<td><strong>Fitness</strong></td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td>( k )-fold crossover</td>
</tr>
<tr>
<td><strong>Selection</strong></td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td>tournament of ( k ) individuals</td>
</tr>
</tbody>
</table>

**Activation functions:** relu, tanh, sigmoid, hard sigmoid, linear  
**Learning algorithm:** RMSprop
## Experimental Results: ES vs. GA

<table>
<thead>
<tr>
<th></th>
<th>GA</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>std</td>
</tr>
<tr>
<td>CO part1</td>
<td>0.209</td>
<td>0.014</td>
</tr>
<tr>
<td>CO part2</td>
<td>0.801</td>
<td>0.135</td>
</tr>
<tr>
<td>CO part3</td>
<td>0.266</td>
<td>0.029</td>
</tr>
<tr>
<td>CO part4</td>
<td>0.404</td>
<td>0.226</td>
</tr>
<tr>
<td>CO part5</td>
<td>0.246</td>
<td>0.024</td>
</tr>
<tr>
<td>NOx part1</td>
<td>2.201</td>
<td>0.131</td>
</tr>
<tr>
<td>NOx part2</td>
<td>1.705</td>
<td>0.284</td>
</tr>
<tr>
<td>NOx part3</td>
<td>1.238</td>
<td>0.163</td>
</tr>
<tr>
<td>NOx part4</td>
<td>1.490</td>
<td>0.173</td>
</tr>
<tr>
<td>NOx part5</td>
<td>0.551</td>
<td>0.052</td>
</tr>
<tr>
<td>NO2 part1</td>
<td>1.697</td>
<td>0.266</td>
</tr>
<tr>
<td>NO2 part2</td>
<td>2.009</td>
<td>0.415</td>
</tr>
<tr>
<td>NO2 part3</td>
<td>0.593</td>
<td>0.082</td>
</tr>
<tr>
<td>NO2 part4</td>
<td>0.737</td>
<td>0.023</td>
</tr>
<tr>
<td>NO2 part5</td>
<td>1.265</td>
<td>0.158</td>
</tr>
<tr>
<td>C6H6 part1</td>
<td>0.013</td>
<td>0.005</td>
</tr>
<tr>
<td>C6H6 part2</td>
<td>0.039</td>
<td>0.015</td>
</tr>
<tr>
<td>C6H6 part3</td>
<td>0.019</td>
<td>0.011</td>
</tr>
<tr>
<td>C6H6 part4</td>
<td>0.030</td>
<td>0.015</td>
</tr>
<tr>
<td>C6H6 part5</td>
<td>0.017</td>
<td>0.015</td>
</tr>
<tr>
<td>NMHC part1</td>
<td>1.719</td>
<td>0.168</td>
</tr>
<tr>
<td>NMHC part2</td>
<td>0.623</td>
<td>0.164</td>
</tr>
<tr>
<td>NMHC part3</td>
<td>1.144</td>
<td>0.181</td>
</tr>
<tr>
<td>NMHC part4</td>
<td>1.220</td>
<td>0.206</td>
</tr>
<tr>
<td>NMHC part5</td>
<td>1.222</td>
<td>0.126</td>
</tr>
</tbody>
</table>

|                | 11   | 15   |
|                | 44%  | 60%  |
## Experiments Results: Evolved vs. SVR

### Testing errors

<table>
<thead>
<tr>
<th>Task</th>
<th>Evolved avg</th>
<th>Evolved std</th>
<th>Evolved min</th>
<th>Evolved max</th>
<th>SVR linear</th>
<th>SVR RBF</th>
<th>SVR Poly.</th>
<th>SVR Sigmoid</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO_part1</td>
<td>0.229</td>
<td>0.026</td>
<td>0.195</td>
<td>0.267</td>
<td>0.340</td>
<td>0.280</td>
<td>0.285</td>
<td>1.533</td>
</tr>
<tr>
<td>CO_part2</td>
<td>0.657</td>
<td>0.024</td>
<td>0.631</td>
<td>0.694</td>
<td>0.614</td>
<td>0.412</td>
<td>0.621</td>
<td>1.753</td>
</tr>
<tr>
<td>CO_part3</td>
<td>0.256</td>
<td>0.045</td>
<td>0.199</td>
<td>0.349</td>
<td>0.314</td>
<td>0.408</td>
<td>0.377</td>
<td>1.427</td>
</tr>
<tr>
<td>CO_part4</td>
<td>0.526</td>
<td>0.108</td>
<td>0.308</td>
<td>0.701</td>
<td>1.127</td>
<td>0.692</td>
<td>0.535</td>
<td>1.375</td>
</tr>
<tr>
<td>CO_part5</td>
<td>0.235</td>
<td>0.025</td>
<td>0.199</td>
<td>0.277</td>
<td>0.348</td>
<td>0.207</td>
<td>0.198</td>
<td>1.568</td>
</tr>
<tr>
<td>NOx_part1</td>
<td>2.132</td>
<td>0.086</td>
<td>2.021</td>
<td>2.284</td>
<td>1.062</td>
<td>1.447</td>
<td>1.202</td>
<td>2.537</td>
</tr>
<tr>
<td>NOx_part2</td>
<td>1.599</td>
<td>0.077</td>
<td>1.444</td>
<td>1.685</td>
<td>2.162</td>
<td>1.838</td>
<td>1.387</td>
<td>2.428</td>
</tr>
<tr>
<td>NOx_part3</td>
<td>1.339</td>
<td>0.242</td>
<td>1.106</td>
<td>1.955</td>
<td>0.594</td>
<td>0.674</td>
<td>0.665</td>
<td>2.705</td>
</tr>
<tr>
<td>NOx_part4</td>
<td>1.610</td>
<td>0.164</td>
<td>1.435</td>
<td>2.041</td>
<td>0.864</td>
<td>0.903</td>
<td>0.778</td>
<td>2.462</td>
</tr>
<tr>
<td>NOx_part5</td>
<td>0.622</td>
<td>0.075</td>
<td>0.521</td>
<td>0.726</td>
<td>1.632</td>
<td>0.730</td>
<td>1.446</td>
<td>2.761</td>
</tr>
<tr>
<td>NO2_part1</td>
<td>1.506</td>
<td>0.217</td>
<td>1.132</td>
<td>1.823</td>
<td>2.464</td>
<td>2.404</td>
<td>2.401</td>
<td>2.636</td>
</tr>
<tr>
<td>NO2_part2</td>
<td>1.371</td>
<td>0.048</td>
<td>1.242</td>
<td>1.415</td>
<td>2.118</td>
<td>2.250</td>
<td>2.409</td>
<td>2.648</td>
</tr>
<tr>
<td>NO2_part3</td>
<td>0.660</td>
<td>0.078</td>
<td>0.599</td>
<td>0.863</td>
<td>1.308</td>
<td>1.195</td>
<td>1.213</td>
<td>1.984</td>
</tr>
<tr>
<td>NO2_part4</td>
<td>0.782</td>
<td>0.043</td>
<td>0.711</td>
<td>0.856</td>
<td>1.978</td>
<td>2.565</td>
<td>1.912</td>
<td>2.531</td>
</tr>
<tr>
<td>NO2_part5</td>
<td>0.730</td>
<td>0.111</td>
<td>0.520</td>
<td>0.905</td>
<td>1.0773</td>
<td>1.047</td>
<td>0.967</td>
<td>2.129</td>
</tr>
<tr>
<td>C6H6_part1</td>
<td>0.013</td>
<td>0.004</td>
<td>0.007</td>
<td>0.018</td>
<td>0.300</td>
<td>0.511</td>
<td>0.219</td>
<td>1.398</td>
</tr>
<tr>
<td>C6H6_part2</td>
<td>0.034</td>
<td>0.010</td>
<td>0.020</td>
<td>0.050</td>
<td>0.378</td>
<td>0.489</td>
<td>0.369</td>
<td>1.478</td>
</tr>
<tr>
<td>C6H6_part3</td>
<td>0.048</td>
<td>0.015</td>
<td>0.016</td>
<td>0.075</td>
<td>0.520</td>
<td>0.663</td>
<td>0.538</td>
<td>1.317</td>
</tr>
<tr>
<td>C6H6_part4</td>
<td>0.020</td>
<td>0.010</td>
<td>0.010</td>
<td>0.042</td>
<td>0.217</td>
<td>0.459</td>
<td>0.123</td>
<td>1.279</td>
</tr>
<tr>
<td>C6H6_part5</td>
<td>0.027</td>
<td>0.011</td>
<td>0.014</td>
<td>0.051</td>
<td>0.215</td>
<td>0.297</td>
<td>0.188</td>
<td>1.526</td>
</tr>
<tr>
<td>NMHC_part1</td>
<td>1.685</td>
<td>0.256</td>
<td>1.448</td>
<td>2.378</td>
<td>1.718</td>
<td>1.666</td>
<td>1.621</td>
<td>3.861</td>
</tr>
<tr>
<td>NMHC_part2</td>
<td>0.713</td>
<td>0.097</td>
<td>0.566</td>
<td>0.865</td>
<td>0.934</td>
<td>0.978</td>
<td>0.839</td>
<td>3.651</td>
</tr>
<tr>
<td>NMHC_part3</td>
<td>1.097</td>
<td>0.270</td>
<td>0.775</td>
<td>1.560</td>
<td>1.580</td>
<td>1.280</td>
<td>1.438</td>
<td>2.830</td>
</tr>
<tr>
<td>NMHC_part4</td>
<td>1.099</td>
<td>0.166</td>
<td>0.898</td>
<td>1.443</td>
<td>1.720</td>
<td>1.565</td>
<td>1.917</td>
<td>2.715</td>
</tr>
<tr>
<td>NMHC_part5</td>
<td>1.023</td>
<td>0.050</td>
<td>0.963</td>
<td>1.116</td>
<td>1.238</td>
<td>0.944</td>
<td>1.407</td>
<td>2.960</td>
</tr>
</tbody>
</table>

| 17 | 2 | 2 | 4 |
| 68% | 8% | 8% | 16% |
Evolved networks are quite small.

- A typical network:
  - One hidden layer of about 70 neurons
  - Dropout rate 0.3
  - ReLU activation function.
## Experiments Results: Evolved vs. fixed architecture

### Testing errors

<table>
<thead>
<tr>
<th>Task</th>
<th>Evolved</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>std</td>
<td>avg</td>
<td>std</td>
<td>avg</td>
<td>std</td>
<td>avg</td>
</tr>
<tr>
<td>CO_part1</td>
<td>0.229</td>
<td>0.026</td>
<td>0.230</td>
<td>0.032</td>
<td>0.250</td>
<td>0.023</td>
<td>0.377</td>
</tr>
<tr>
<td>CO_part2</td>
<td>0.657</td>
<td>0.024</td>
<td>0.861</td>
<td>0.136</td>
<td>0.744</td>
<td>0.142</td>
<td>0.858</td>
</tr>
<tr>
<td>CO_part3</td>
<td>0.256</td>
<td>0.045</td>
<td>0.261</td>
<td>0.040</td>
<td>0.305</td>
<td>0.043</td>
<td>0.302</td>
</tr>
<tr>
<td>CO_part4</td>
<td>0.526</td>
<td>0.108</td>
<td>0.621</td>
<td>0.279</td>
<td>0.638</td>
<td>0.213</td>
<td>0.454</td>
</tr>
<tr>
<td>CO_part5</td>
<td>0.235</td>
<td>0.025</td>
<td>0.283</td>
<td>0.072</td>
<td>0.270</td>
<td>0.032</td>
<td>0.309</td>
</tr>
<tr>
<td>NOx_part1</td>
<td>2.132</td>
<td>0.086</td>
<td>2.158</td>
<td>0.203</td>
<td>2.095</td>
<td>0.131</td>
<td>2.307</td>
</tr>
<tr>
<td>NOx_part2</td>
<td>1.599</td>
<td>0.077</td>
<td>1.799</td>
<td>0.313</td>
<td>1.891</td>
<td>0.199</td>
<td>2.083</td>
</tr>
<tr>
<td>NOx_part3</td>
<td>1.339</td>
<td>0.242</td>
<td>1.077</td>
<td>0.125</td>
<td>1.092</td>
<td>0.178</td>
<td>0.806</td>
</tr>
<tr>
<td>NOx_part4</td>
<td>1.610</td>
<td>0.164</td>
<td>1.303</td>
<td>0.208</td>
<td>1.797</td>
<td>0.461</td>
<td>1.600</td>
</tr>
<tr>
<td>NOx_part5</td>
<td>0.622</td>
<td>0.075</td>
<td>0.644</td>
<td>0.075</td>
<td>0.677</td>
<td>0.055</td>
<td>0.778</td>
</tr>
<tr>
<td>NO2_part1</td>
<td>1.506</td>
<td>0.217</td>
<td>1.659</td>
<td>0.250</td>
<td>1.368</td>
<td>0.135</td>
<td>1.677</td>
</tr>
<tr>
<td>NO2_part2</td>
<td>1.371</td>
<td>0.048</td>
<td>1.762</td>
<td>0.237</td>
<td>1.687</td>
<td>0.202</td>
<td>1.827</td>
</tr>
<tr>
<td>NO2_part3</td>
<td>0.660</td>
<td>0.078</td>
<td>0.682</td>
<td>0.148</td>
<td>0.576</td>
<td>0.044</td>
<td>0.603</td>
</tr>
<tr>
<td>NO2_part4</td>
<td>0.782</td>
<td>0.043</td>
<td>1.109</td>
<td>0.923</td>
<td>0.757</td>
<td>0.059</td>
<td>0.802</td>
</tr>
<tr>
<td>NO2_part5</td>
<td>0.730</td>
<td>0.111</td>
<td>0.646</td>
<td>0.064</td>
<td>0.734</td>
<td>0.107</td>
<td>0.748</td>
</tr>
<tr>
<td>C6H6_part1</td>
<td>0.013</td>
<td>0.004</td>
<td>0.012</td>
<td>0.006</td>
<td>0.081</td>
<td>0.030</td>
<td>0.190</td>
</tr>
<tr>
<td>C6H6_part2</td>
<td>0.034</td>
<td>0.010</td>
<td>0.039</td>
<td>0.012</td>
<td>0.101</td>
<td>0.015</td>
<td>0.211</td>
</tr>
<tr>
<td>C6H6_part3</td>
<td>0.048</td>
<td>0.015</td>
<td>0.024</td>
<td>0.007</td>
<td>0.091</td>
<td>0.047</td>
<td>0.115</td>
</tr>
<tr>
<td>C6H6_part4</td>
<td>0.020</td>
<td>0.010</td>
<td>0.026</td>
<td>0.010</td>
<td>0.051</td>
<td>0.026</td>
<td>0.096</td>
</tr>
<tr>
<td>C6H6_part5</td>
<td>0.027</td>
<td>0.011</td>
<td>0.025</td>
<td>0.008</td>
<td>0.113</td>
<td>0.025</td>
<td>0.176</td>
</tr>
<tr>
<td>NMHC_part1</td>
<td>1.685</td>
<td>0.256</td>
<td>1.738</td>
<td>0.144</td>
<td>1.889</td>
<td>0.119</td>
<td>2.378</td>
</tr>
<tr>
<td>NMHC_part2</td>
<td>0.713</td>
<td>0.097</td>
<td>0.553</td>
<td>0.045</td>
<td>0.650</td>
<td>0.078</td>
<td>0.799</td>
</tr>
<tr>
<td>NMHC_part3</td>
<td>1.097</td>
<td>0.270</td>
<td>1.128</td>
<td>0.089</td>
<td>0.901</td>
<td>0.124</td>
<td>0.789</td>
</tr>
<tr>
<td>NMHC_part4</td>
<td>1.099</td>
<td>0.166</td>
<td>1.116</td>
<td>0.119</td>
<td>0.918</td>
<td>0.119</td>
<td>0.751</td>
</tr>
<tr>
<td>NMHC_part5</td>
<td>1.023</td>
<td>0.050</td>
<td>0.970</td>
<td>0.094</td>
<td>0.889</td>
<td>0.085</td>
<td>0.856</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>6</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>40%</td>
<td>24%</td>
<td>16%</td>
<td>20%</td>
</tr>
</tbody>
</table>
### Experimental Results: Sensors Second Task

#### Testing errors

<table>
<thead>
<tr>
<th>Task</th>
<th>Evolved</th>
<th>SVR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>std</td>
</tr>
<tr>
<td>CO</td>
<td>0.115</td>
<td>0.006</td>
</tr>
<tr>
<td>NOx</td>
<td>0.128</td>
<td>0.016</td>
</tr>
<tr>
<td>NO2</td>
<td>0.276</td>
<td>0.012</td>
</tr>
<tr>
<td>C6H6</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>NMHC</td>
<td>0.247</td>
<td>0.058</td>
</tr>
</tbody>
</table>

*evolved networks several layers, dominating activation function ReLU*
Experiment 2: MNIST

Data Set

- well known data set, classification of handwritten digits
- 28 × 28 pixels
- 60000 for training, 10000 for testing

Results

- ES run for 30 generations, n = 5, m = 10

<table>
<thead>
<tr>
<th></th>
<th>avg</th>
<th>std</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>98.34</td>
<td>0.13</td>
<td>98.18</td>
<td>98.55</td>
</tr>
<tr>
<td>evolved</td>
<td><strong>98.64</strong></td>
<td>0.05</td>
<td>98.55</td>
<td>98.73</td>
</tr>
</tbody>
</table>
Convolutional Neural Networks

- convolutional layers
- max-pooling layers
Convolutional Networks in Keras

```python
model = Sequential()
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

- each block has its type: convolutional, max-pooling, dense
individuals consists of two parts convolutional and dense

convolutional part - convolutional and max-pooling layers - feature extraction

dense part - only dense layers - classification
Individuals

Convolutional layer

- number of filters
- kernel size
- activation function type

Max-pooling layer

- pool-size

Genetic operators

- crossover convolutional and dense part separately
- mutation stays the same
Parallel approach

Classic approach

- very time consuming
- each fitness evaluation includes crossvalidation

Parallel approach

- fitness evaluations are independent
- can be done in parallel
- disadvantage: fitness evaluations are not of same duration, some processors waiting
Asynchronous evolution

- individuals evaluated one by one
- no notion of generation
- as soon as there is an idle processor, new individual is created
- arbitrary number of processors

1. get evaluated individual $I$
2. append $I$ to the population
3. discard the worst individual
4. generate new individual $I'$ by genetic operators
5. send $I'$ for fitness evaluation
asynchronous evolution
population size 20
20 generations

<table>
<thead>
<tr>
<th>model</th>
<th>avg</th>
<th>std</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>98.97</td>
<td>0.07</td>
<td>98.84</td>
<td>99.13</td>
</tr>
<tr>
<td>evolved</td>
<td><strong>99.17</strong></td>
<td>0.11</td>
<td>98.92</td>
<td>99.36</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

- proposed ES for DNN architecture design
- demonstrated the algorithm on experiments
- works for feed-forward DNN with dense layers, convolutional networks

Future Work

- evolve also other parameters of learning
- speed up the evolution - asynchronous evolution, surrogate modeling
Deep Networks and RBF Networks

- combinations of Deep Networks and RBF Networks
- RBF layers can be also included in evolution

- RBF networks less vulnerable to adversarial examples
- Does add RBF layers to deep network help to prevent adversarial examples?
RBF Networks

- feed-forward neural networks with one hidden layer of RBF units
- local units alternative to MLP
- RBF unit:
  \[ y = \varphi(\xi); \quad \xi = \beta \| \vec{x} - \vec{c} \|^2 \]
  where \( \varphi : \mathbb{R} \rightarrow \mathbb{R} \) is suitable activation function, typically Gaussian \( \varphi(z) = e^{-z^2} \).
- the network computes the function \( \vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m \):
  \[ f_s(\vec{x}) = \sum_{j=1}^{h} w_{js} \varphi(\beta_j \| \vec{x} - \vec{c}_j \|) \]
RBF Networks Learning

- wide range of methods

Three Step Learning

1. **set the centers** - approximate the distribution of training samples
   - random or uniform samples, various clustering methods
2. **set the widths** - cover the input space by unit’s fields
   - heuristics (k-neighbours)
3. **compute the output weights**
   - linear system, pseudoinverse

Gradient Learning

- analogous to backpropagation for MLP
Adversarial examples

Applying an imperceptible non-random perturbation to an input image, it is possible to arbitrarily change the machine learning model prediction.

Such perturbed examples are known as adversarial examples. For human eye, they seem close to the original examples.

They represent a security flaw in classifier.

Figure from Explaining and Harnessing Adversarial Examples by Goodfellow et al.
Crafting adversarial examples

- several methods for crafting adversarial examples

- FGSM - Fast Gradient Sign Method


\[ \eta = \varepsilon \operatorname{sgn}(\nabla_x J(\theta, x, y)) \]

- our work - using Genetic Algorithm to craft adversarial examples
  does not need access to models weights
Adversarial examples by FGSM

Legitimate samples:

Adversarial samples $\epsilon = 0.2$:

Adversarial samples $\epsilon = 0.3$:

Adversarial samples $\epsilon = 0.4$: 
Adversarial examples by GA
Proposed architecture DNNRBF

- stacking deep neural network and RBF network
DNNRBF learning

1. train the DNN
2. set the centers of $RBF$ randomly, drawn from uniform distribution on $(0, 1.0)$
3. set the parameters $\beta$ to the constant value
4. init the weights of RBF output layer to random small values
5. retrain the whole network DNNRBF (by back propagation)
Experiments

Architectures

- **MLP**
  - dense layer of 512 ReLU
  - dense layer of 512 ReLU
  - dense layer of 10 softmax units

- **CNN**
  - convolutional layer with 32 3x3 filters and ReLU activation
  - convolutional layer with 32 3x3 filters and ReLU activation
  - 2x2 max pooling layer
  - dense layer of 128 ReLU
  - dense layer of 10 softmax units
Experiments

Implementation

- FGSM for crafting adversarial examples

- Keras for MLP and CNN
  *Keras, François Chollet, https://github.com/fchollet/keras, 2015*

- our implementation of RBF Keras layers
  *http://github.com/PetraVidnerova/rbf_keras*

  *http://github.com/PetraVidnerova/rbf_tests*
## Experiments Results - MLP

<table>
<thead>
<tr>
<th>model</th>
<th>Mean</th>
<th>Std</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>98.35</td>
<td>0.12</td>
<td>98.04</td>
<td>98.59</td>
<td>1.95</td>
<td>0.41</td>
<td>1.30</td>
<td>2.86</td>
</tr>
<tr>
<td>MLPRBF(0.01)</td>
<td>97.62</td>
<td>2.43</td>
<td>88.44</td>
<td>98.65</td>
<td>2.56</td>
<td>2.09</td>
<td>1.16</td>
<td>10.71</td>
</tr>
<tr>
<td>MLPRBF(0.1)</td>
<td>88.61</td>
<td>8.56</td>
<td>69.91</td>
<td>98.36</td>
<td>10.04</td>
<td>6.45</td>
<td>1.71</td>
<td>23.10</td>
</tr>
<tr>
<td>MLPRBF(1.0)</td>
<td>98.23</td>
<td>0.10</td>
<td>98.08</td>
<td>98.48</td>
<td>81.77</td>
<td>7.84</td>
<td>64.18</td>
<td>94.06</td>
</tr>
<tr>
<td><strong>MLPRBF(2.0)</strong></td>
<td><strong>98.19</strong></td>
<td><strong>0.14</strong></td>
<td><strong>97.91</strong></td>
<td><strong>98.38</strong></td>
<td><strong>89.21</strong></td>
<td><strong>5.03</strong></td>
<td><strong>66.28</strong></td>
<td><strong>94.83</strong></td>
</tr>
<tr>
<td>MLPRBF(3.0)</td>
<td>98.18</td>
<td>0.14</td>
<td>97.88</td>
<td>98.45</td>
<td>81.66</td>
<td>4.38</td>
<td>70.13</td>
<td>87.23</td>
</tr>
<tr>
<td>MLPRBF(5.0)</td>
<td>97.64</td>
<td>2.09</td>
<td>89.34</td>
<td>98.36</td>
<td>69.47</td>
<td>13.26</td>
<td>13.01</td>
<td>81.95</td>
</tr>
<tr>
<td>MLPRBF(10.0)</td>
<td>80.94</td>
<td>11.82</td>
<td>58.57</td>
<td>98.33</td>
<td>21.49</td>
<td>16.32</td>
<td>2.48</td>
<td>65.11</td>
</tr>
</tbody>
</table>

### Average accuracies

- **On legitimate data**: All models perform well, with MLP showing the highest accuracy, followed by MLPRBF(2.0) and then the rest. The standard deviation is low, indicating consistent performance.
- **On adversarial data**: The models struggle more with adversarial samples, with MLP PRBF(2.0) maintaining a reasonable accuracy, while others drop significantly. The standard deviation is higher, indicating greater variability.

![Average accuracies graph](image-url)
## Experiments Results - CNN

<table>
<thead>
<tr>
<th>Model</th>
<th>Legitimate samples</th>
<th>Adversarial samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>std</td>
</tr>
<tr>
<td>CNN</td>
<td>98.97</td>
<td>0.07</td>
</tr>
<tr>
<td>CNNRBF(0.01)</td>
<td>98.36</td>
<td>1.73</td>
</tr>
<tr>
<td>CNNRBF(0.1)</td>
<td>94.19</td>
<td>8.21</td>
</tr>
<tr>
<td>CNNRBF(1.0)</td>
<td>98.83</td>
<td>0.13</td>
</tr>
<tr>
<td><strong>CNNRBF(2.0)</strong></td>
<td><strong>98.85</strong></td>
<td><strong>0.13</strong></td>
</tr>
<tr>
<td>CNNRBF(3.0)</td>
<td>98.82</td>
<td>0.14</td>
</tr>
<tr>
<td>CNNRBF(5.0)</td>
<td>98.74</td>
<td>0.11</td>
</tr>
<tr>
<td>CNNRBF(10.0)</td>
<td>97.86</td>
<td>2.24</td>
</tr>
</tbody>
</table>

---

![Average accuracies on legitimate data](image)

![Average accuracies on adversarial data](image)
# Experiments Results

<table>
<thead>
<tr>
<th>model</th>
<th>Accuracy on adversarial data</th>
<th>$\epsilon = 0.2$</th>
<th>$\epsilon = 0.3$</th>
<th>$\epsilon = 0.4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>std</td>
<td>avg</td>
<td>std</td>
</tr>
<tr>
<td>CNN</td>
<td>33.85</td>
<td>7.58</td>
<td>8.49</td>
<td>3.52</td>
</tr>
<tr>
<td>CNNRBF</td>
<td>76.88</td>
<td>6.25</td>
<td>74.57</td>
<td>7.69</td>
</tr>
<tr>
<td>MLP</td>
<td>3.01</td>
<td>0.69</td>
<td>1.95</td>
<td>0.41</td>
</tr>
<tr>
<td>MLPRBF</td>
<td>90.14</td>
<td>4.82</td>
<td>89.21</td>
<td>5.03</td>
</tr>
</tbody>
</table>
Thank you! Questions?