DATA SCIENCE & SCIENTIFIC COMPUTING

Evolutionary architecture selection

Theory and application to CNN

Erika Lena, Francesco Ortu, Alessandro Serra

github.com/Francesc0rtu/Genetics_ANN

Overview

- Genetic Algorithms
- Grammatical Evolution
- DENSER
- Implementation
- Results

Genetic Algorithms

Genetic Algorithms

Genetic Algorithms belong to the broader class of Evolutionary Algorithms, that is optimization framework that relies on biological inspired heuristics.

They perform very well when the search space of the solutions is full of local minima and have an unknown and complicated behaviour.

The general approach involves initializing a population of candidate solution and evolving it through several iterations towards optimality.

Genetic Algorithms - 1

Each solution is encoded through a *genetic representation* and a *fitness function* is defined to assess the performance of an individual.

The evolution is composed by the following phases:

- Selection: The portion of the population which is fitting better is selected to breed a new generation
- Genetic Operators: The parents selected in the previous section are recombined through a *crossover* of their *genotype* and a random subset of the offspring is subjected to a *mutation*

The program iterates until a terminal condition is satisfied (i.e. number of generations, fitness of the individuals, etc..)

Genetic Algorithms

Grammatical Evolution

Grammatical Evolution

The genetic representation of a individual consist in the *genotype* which is an encoding of the solution that is computable and that can be handled by a machine while the actual object of the solution which is called *phenotype*.

A subclass of genetic algorithms is called Grammatical Evolution

A grammar define a series of syntactical rules through which construct a valid *phenotype* expression from a *genotype*.

Grammar

A grammar employs **Backus-Naur Form** notation. All the entities (or *symbols*) are either *non-terminal* i.e. defined by other *non-terminal* symbols and *terminal* which are the lexicon of the language. The definition of the symbols is hierarchical thus it starts from more abstract entities down to the *non-terminal* symbols :

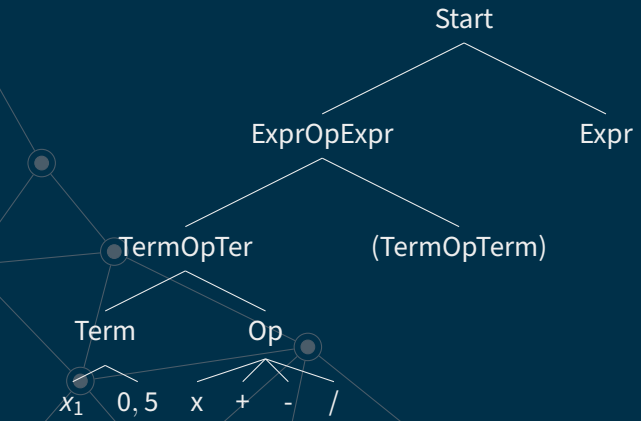
<start> ::= <expr> <op> <expr> | <expr> (0), (1)

<expr> ::= <term> <op> <term> | (<term> <op> <term>) (0), (1)

<op> ::= + | - | / | * (0), (1), (2), (3)

<term> ::= x_1 | 0.5 (0), (1)

Grammar Tree



Derivation Step

Derivation Step	Integers Left
<start>	[23; 7; 55; 22; 3; 4; 30; 16; 203; 24]
<expr> <op> <expr>	[7; 55; 22; 3; 4; 30; 16; 203; 24]
(<term> <op> <term>)<op><expr>	[55; 22; 3; 4; 30; 16; 203; 24]
(0.5 <op> <term>)<op><expr>	[22; 3; 4; 30; 16; 203; 24]
(0.5 / <term>)<op><expr>	[3; 4; 30; 16; 203; 24]
(0.5 / 1)<op><expr>	[4; 30; 16; 203; 24]
(0.5 / 1) + <expr>	[30; 16; 203; 24]
(0.5 / 1) + <term> <op> <term>	[16; 203; 24]
(0.5 / 1) + x_1 <op> <term>	[203; 24]
(0.5 / 1) + x_1 * <term>	[24]
(0.5 / 1) + x_1 * x_1	[ ]

DSGE

The original representation of the grammar used to describe expansions rules recursively and to led to poor locality of the solution and low stability to mutations and crossover. In the *Dynamic Structured Grammatical Evolution* the *genotype* associated to each symbols the list of all of its expansion, so each genetical operator will modify at most one of the entities of the grammatical expression of the individual.

Derivation Steps

Derivation Step	Integers Left
<start>	[[0], [0,1], [2,0,3], [1,1,0,0]]
<expr> <op> <expr>	[[], [0,1], [2,0,3], [1,1,0,0]]
(<term> <op> <term>)<op><expr>	[[], [1], [2,0,3], [1,1,0,0]]
(0.5 <op> <term>)<op><expr>	[[], [1], [2,0,3], [1,0,0]]
(0.5 / <term>)<op><expr>	[[], [1], [0,3], [1,0,0]]
(0.5 / 1)<op><expr>	[[], [1], [0,3], [0,0]]
(0.5 / 1) + <expr>	[[], [1], [3], [0,0]]
(0.5 / 1) + <term> <op> <term>	[[], [], [3], [0,0]]
(0.5 / 1) + x_1 <op> <term>	[[], [], [3], [0]]
(0.5 / 1) + x_1 * <term>	[[], [], [], [0]]
(0.5 / 1) + x_1 * x_1	[[], [], [], []]

DENSER

DENSER

Evolving DNN

DENSER stands for **deep evolutionary network structured representation** and it is a new approach for the construction of grammar-based ANNs.[**?**]

It makes use of DSGE for the development of multi-layered and multiple output nodes neural networks and of GA principles for what concerns evolution in order to obtain an individual suitable to the task we want to solve.

DENSER

Motivations

The idea is to have automated machine learning tools (Auto-ML).
We want to exploit previous concepts in order to deal with:

- feature engineering
- model-selection
- hyperparameter configuration

The aim is not have to manually configure the DNNs we want to use, which is time consuming and error-prone.

DENSER

The Grammar

⟨*features*⟩ ::= <convolution> | <pooling>

⟨*convolution*⟩ ::= layer:conv [num-filters,int,1,32,256] [filter-shape,int,1,1,5] [stride,int,1,1,3]<pad><activation><bias><batch-norm>

⟨*batch-norm*⟩ := batch-normalisation:True | batch-normalisation:False

⟨*pooling*⟩ ::= <pool-type>[kernel-size,int,1,1,5][stride,int,1,1,3]<pad>

⟨*pool-type*⟩ ::= layer:pool-avg | layer:pool-max

⟨*pad*⟩ ::= padding:same | padding:valid

⟨*classification*⟩ ::= <fully-connected>

⟨*fully-connected*⟩ ::= layer:fc<activation>[num-units,int,1,128,2048<bias>

⟨*activation*⟩ ::= act:linear | act:relu | act:sigmoid <bias> ::= bias:True | bias:False

⟨*softmax*⟩ ::= layer:fc act:softmax num-units:10 bias:True

⟨*learning*⟩ ::= learning:gradient-descent [lr,float,1,0.0001,0.1]

DENSER

Representation

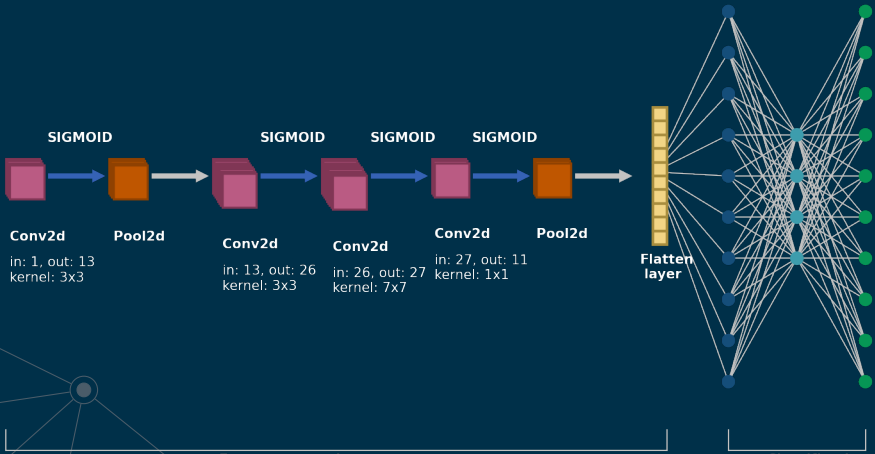
The innovation provided by DENSER is that of combine GA and DSGE in a unique representation of the ANN.
Each candidate solution is represented at two levels:

- the GA level: encodes the macro-structure of the DNN (the "genes");
- the DSGE level: encodes the specific representation of each layer, along with its parameters.

Example of GA structure, used later on:

[(features, 1, 30), (classification, 1, 10), (softmax, 1, 1)];

Network representation



DENSER

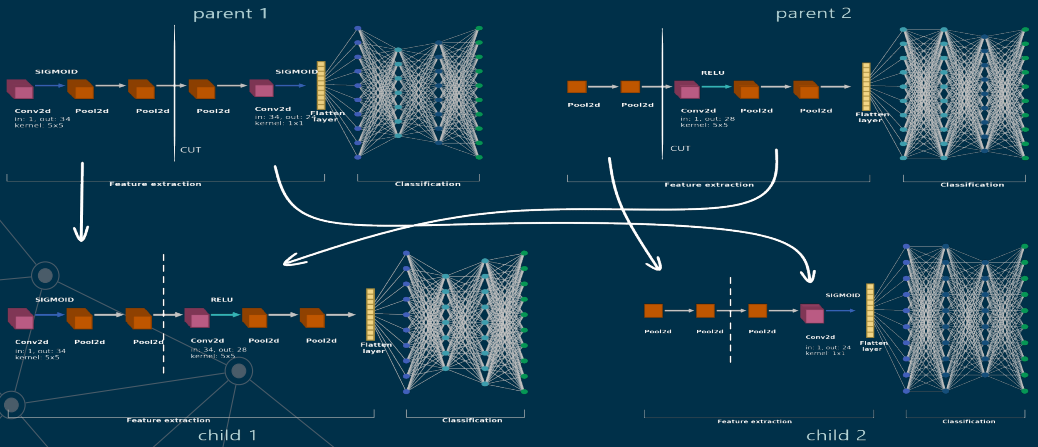
Genetic operators

In order to automatically choose our DNN, we start with a population of random configurations and we try to obtain individuals which perform better at each generation, by repeated combinations of the most suitable candidate solutions.

New individuals are obtained by applying genetic operators, as in GA, these are:

- crossovers
  - one-point
  - bit-mask
- mutations

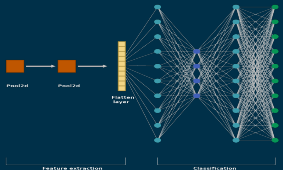
One point crossover



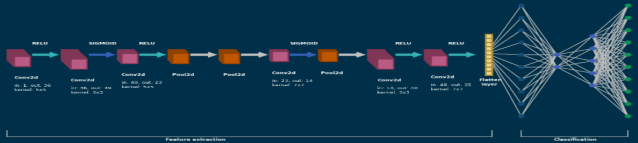
Bit-mask crossover

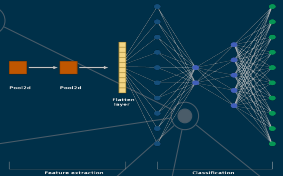
bitmask: 100

parent 1

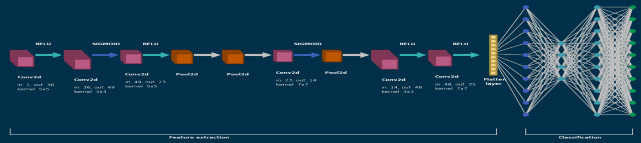


parent 2





child 1



child 2

DENSER

Mutations

Mutations at the GA level manipulate the whole architecture by:

- adding
- replacing
- removing

one evolutionary unit.

At DSGE level, instead, layers' parameters are modified by:

- changing one expansion rule
- changing entirely one block along with its hyperparameters

DENSER

Evaluation

Once a new population is obtained through crossovers and mutations, each individual needs to be evaluated to establish its suitability.

So starting from its GA-DSGE representation: each individual is mapped into a trainable DNN; the corresponding model is trained; its performances are measured to determine its fitness.

The metric used to evaluate the classification models is accuracy.

Implementation

Implementation

Main concepts

An implementation of DENSER needs:

- A class to represent the networks and to handle the genetics operator.
- A class to manage the evolution.
- A way to use PyTorch to train and evaluate the networks.

We decided to implement from scratch the representation and evolution class and then assemble a PyTorch module for each net in order to test and evaluate the individual.

Implementation

DSGE level

```
class Layer:
    def __init__(self, type, c_out, param) # initialise a layer either random or specifying
        self.layer_type = layer_type.CONV # the type and parameters
        self.param = {"kernel_size": 3, "stride":1, "padding": same}
        self.channels = {"in": 3, "out":17}
    def random_init(self)                 # random initialise the layer
    def compute_shape(self,input_shape) # compute the output shape if
                                          # the layer is convolutional or pooling
class Module:
    def __init__(self, M_type, c_out) # initialise the module with a random layers
        self.M_type = M_type.CLASSIFICATION
        self.layers = initialise(self.M_type, c_out)
    def initialise(self, type, c_out) # random add to the module layers
    def compute_shape(self, input_shape) # compute the output shape of the module
```

Implementation

GA level

```
class Net_encoding:
    def __init__(self, len_features, len_class, c_in, c_out, input_shape)
     #initialise the net encoding object with random module and layers
        self.features = []
        self.classification = []
        ...
    def len(self):
    def compute_shape_features(self, input_shape) #compute the output shape of the net
    def update_encoding(self) # remove not valid layers
    def fix_first_classification(self) # adjust the inputs of the first
                                       # classification module
class Net(nn.Module):
    def __init__(self, Net_encod) #initialise a torch network from our encoding
    def forward(self,x)  # forward pass
```

Implementation

Mutations, crossover and evolution

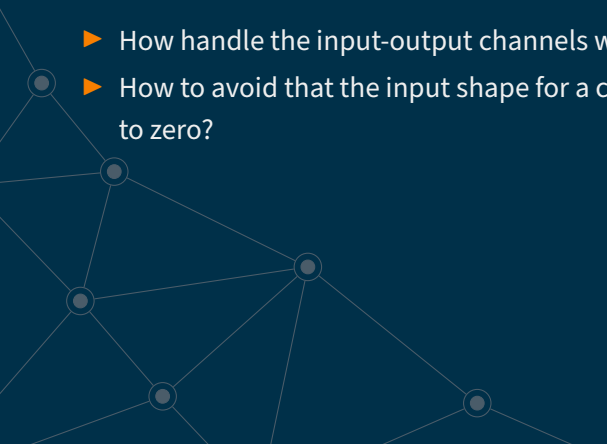
```
def GA_bit_mask(parent1, parent2) # Bitmask crossover
def GA_one_point(parent1, parent2): # One point crossover
def GA_add(offspring, cut, module) # add a module to offspring at position cut
def GA_remove(offspring) # remove a random module of offspring
def GA_replace(offspring) # replace a random module of offspring
                          # with another randomically generated
def dsge_mutation(offspring) # grammatical or integer mutation of a
                             # radom layer of offspring


class evolution():
    def __init__(self, pop_size, holdout, mating): # init the initial population
            # randomically
    def generation(self): # train and eval the population and execute
                          # crossover and mutation
```

Implementation

Two problems, two solutions

During the implementation we faced two main problems:

- How handle the input-output channels with a dynamic structure?
- How to avoid that the input shape for a certain layer is less then kernel size or is equal to zero?



Implementation

Two problems, two solutions

During the implementation we faced two main problems:

- How handle the input-output channels with a dynamic structure?
- How to avoid that the input shape for a certain layer is less then kernel size or is equal to zero?

Solutions:

Implementation

Two problems, two solutions

During the implementation we faced two main problems:

- How handle the input-output channels with a dynamic structure?
- How to avoid that the input shape for a certain layer is less then kernel size or is equal to zero?

Solutions:

1. We record only output channels (random initialised) and we compute input channels only when transform the encoding to a PyTorch net.

Implementation

Two problems, two solutions

During the implementation we faced two main problems:

- How handle the input-output channels with a dynamic structure?
- How to avoid that the input shape for a certain layer is less then kernel size or is equal to zero?

Solutions:

1. We record only output channels (random initialised) and we compute input channels only when transform the encoding to a PyTorch net.
2. When we decode the network we drop all the layers that produce not valid shape.

Implementation

Demo

Let’s look at the code

Implementation

Network representation, generation: 0

Results

Results

Datasets and Hardware

- We tried our model on MNIST[1] and CIFAR10[2].
- Due to the fact that the task was computational demanding we reduced the numbers of channels and we train each networks only using $1/10$ of the size of the dataset. However we find good results with much less computation time.
- We ran the code 4 times.
- We ran the code on ORFEO[1], using as accelerator GPU NVIDIA V100. The computation took about 20 hours.

[1]https://www.areasciencepark.it/piattaforme-tecnologiche/data-center-orfeo/

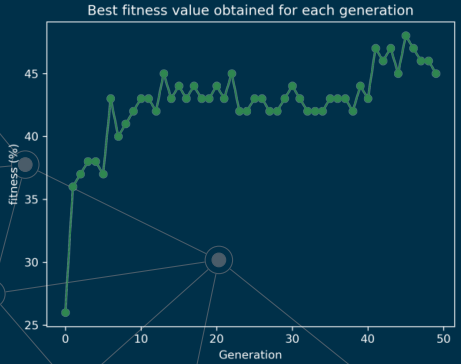
Results

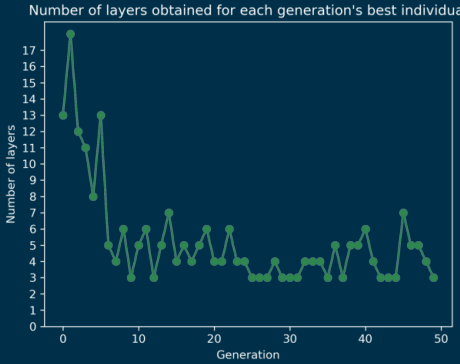
Parameters

Parameters	MNIST	CIFAR10
Max number of features block	10	30
Max number of classification block	2	10
Range of channels for features block	[9, 50]	[32, 256]
Range of channels for classification block	[64, 1024]	[128, 2048]
Range Kernel size	[1, 8]	[1, 5]
Size of train set for each individual	5000	4000
Mutation rate	33%	33%
Crossover rate	70%	70%

Results

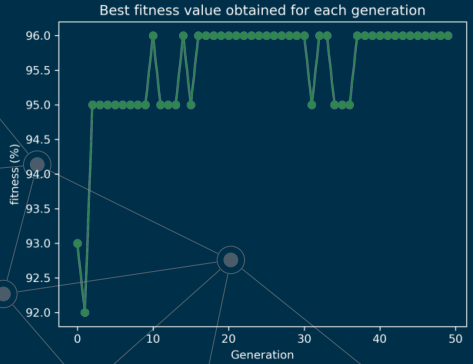
CIFAR-10

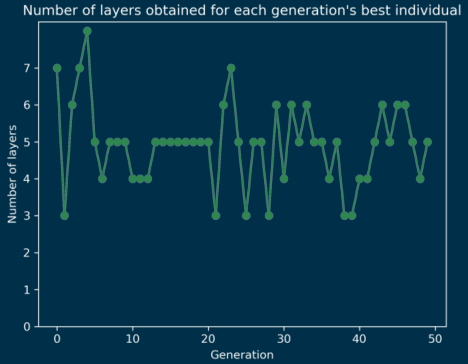




Results

MNIST



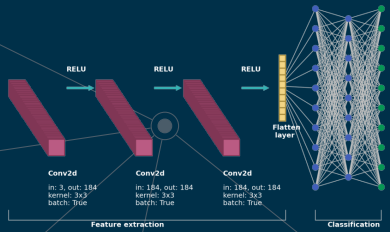


Results

Best individual

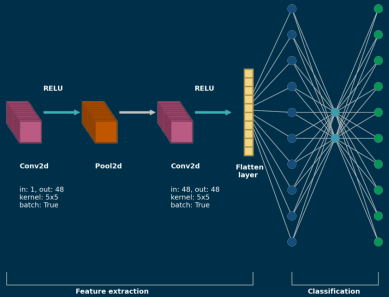
CIFAR-10, Acc: 77%

Network representation, generation: 49



MNIST, Acc: 99.39%

Network representation, generation: 49



Conclusion

Conclusion

Increasing complexity in task and very heterogeneous data landscape often demands deep ANN.

The challenge with these nets regards how to find adequate architecture and parametrization

DENSER try to tackle the issue optimizing both the topology and parameters of each layers through an evolutionary approach.

Conclusion

The innovative employment of both a **GA** and **DSGE** level for encoding solutions has a two-fold gain:

- the genetic material is encapsulated, which facilitates the application of the genetic operators
- the grammatical nature of the method makes it easy to evolve solutions to different problems, or different network structures

Conclusion

After applying DENSER to different datasets there seems to emerge some recurrent patterns in both structure and parameters of better performing nets Evolution seems to favour individual that employ batch normalization, *ReLu* is generally selected as activation function for *convolutional* layers, while *sigmoid* for *linear* layers

Possible future developments

- Add skip layer in order to avoid the vanishing gradient.
- Extend the code to works with different grammars.
- Parallelize the code, in particular the for loop inside each generation.

References I

L. Deng.
The mnist database of handwritten digit images for machine learning research.
IEEE Signal Processing Magazine, 29(6):141–142, 2012.

A. Krizhevsky, V. Nair, and G. Hinton.
Cifar-10 (canadian institute for advanced research).