# AutoML-Zero: Evolving Machine Learning Algorithms From Scratch

Kseniia Petrushina

MIPT, 2024

February 20, 2024

# Motivation

## AutoML

Automate the design of model structures and learning methods

## Current solutions

- Growing networks neuron-by-neuron
- Bayesian hyperparameter optimization
- Neural architecture search (including fine-grained search)
- Joint neural architecture and hyperparameter search

## Limitations

- Human bias due to the creation of building blocks
- Limited search space

# Proposed approach

## Idea

Evolutionary symbolic search over basic mathematical operations

## Process formulation

1. Given a set of ML tasks $\mathcal{T}$, find optimal algorithm $\alpha^* \in \mathcal{A}$
2. Quality of $\alpha$ is measured on $\mathcal{T}_{search} \subset \mathcal{T}$. Each search experiment produces candidate algorithm
3. The best candidate is chosen on $\mathcal{T}_{select} \subset \mathcal{T}$

## Search space

- Functions (`Setup`, `Predict`, `Learn`)
- Scalar, vector and matrix variables
- Instruction with mathematical operation

# Evaluation

```
# (Setup, Predict, Learn) = input ML algorithm.
# Dtrain / Dvalid = training / validation set.
# sX/vX/mX: scalar/vector/matrix var at address X.
def Evaluate(Setup, Predict, Learn, Dtrain,
Dvalid):
  # Zero-initialize all the variables (sX/vX/mX).
  initialize_memory()

  Setup() # Execute setup instructions.

  for (x, y) in Dtrain:
    v0 = x # x will now be accessible to Predict.
    Predict() # Execute prediction instructions.
    # s1 will now be used as the prediction.
    s1 = Normalize(s1) # Normalize the prediction.
    s0 = y # y will now be accessible to Learn.
    Learn() # Execute learning instructions.

  sum_loss = 0.0
  for (x, y) in Dvalid:
    v0 = x
    Predict() # Only Predict(), not Learn().
    s1 = Normalize(s1)
    sum_loss += Loss(y, s1)

  mean_loss = sum_loss / len(Dvalid)
  # Use validation loss to evaluate the algorithm.
  return mean_loss
```

Figure: Algorithm evaluation on one task.

# Evolutionary algorithm

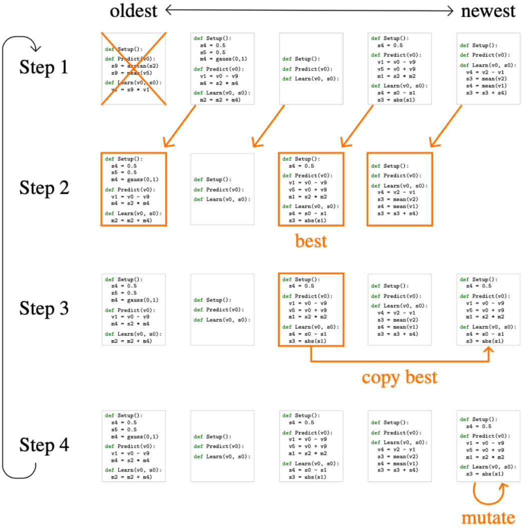



Figure: One cycle of the evolutionary method.

# Evolutionary algorithm

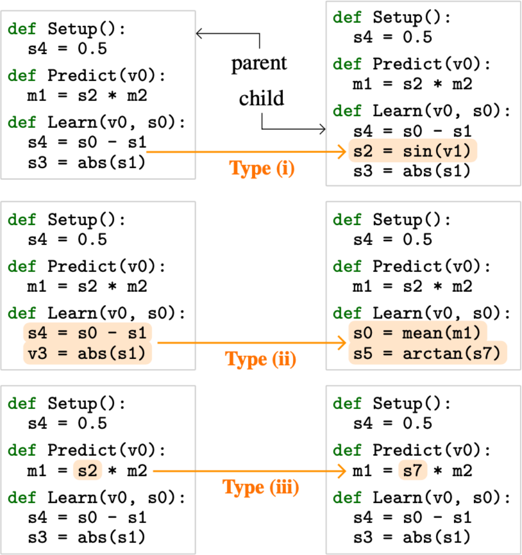



Figure: Mutation examples.

# Results

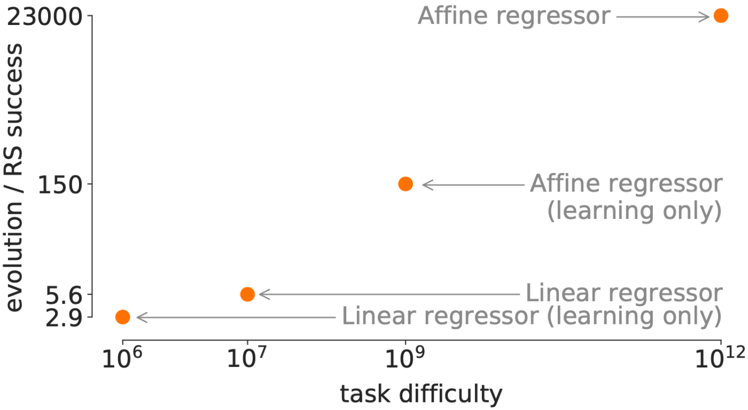



Figure: Relative success rate of evolution and random search (RS).

# Results

## Non-linear data

*Teacher* neural networks generates regression tasks.

- $\mathcal{T}_{search}$ consists of 1 task - the algorithm hard-codes *teacher* weights
- $\mathcal{T}_{search}$ consists of 100 task - evolution discovers the forward pass and "invents" back-propagation code

```
# sX/vX/mX = scalar/vector/matrix at address X.
# "gaussian" produces Gaussian IID random numbers.
def Setup():
  # Initialize variables.
  m1 = gaussian(-1e-10, 9e-09) # 1st layer weights
  s3 = 4.1 # Set learning rate
  v4 = gaussian(-0.033, 0.01) # 2nd layer weights
def Predict(): # v0=features
  v6 = dot(m1, v0) # Apply 1st layer weights
  v7 = maximum(0, v6) # Apply ReLU
  s1 = dot(v7, v4) # Compute prediction
def Learn(): # s0=label
  v3 = heaviside(v6, 1.0) # ReLU gradient
  s1 = s0 - s1 # Compute error
  s2 = s1 * s3 # Scale by learning rate
  v2 = s2 * v3 # Approx. 2nd layer weight delta
  v3 = v2 * v4 # Gradient w.r.t. activations
  m0 = outer(v3, v0) # 1st layer weight delta
  m1 = m1 + m0 # Update 1st layer weights
  v4 = v2 + v4 # Update 2nd layer weights
```

Figure: Relative success rate of evolution and random search (RS).

# Results

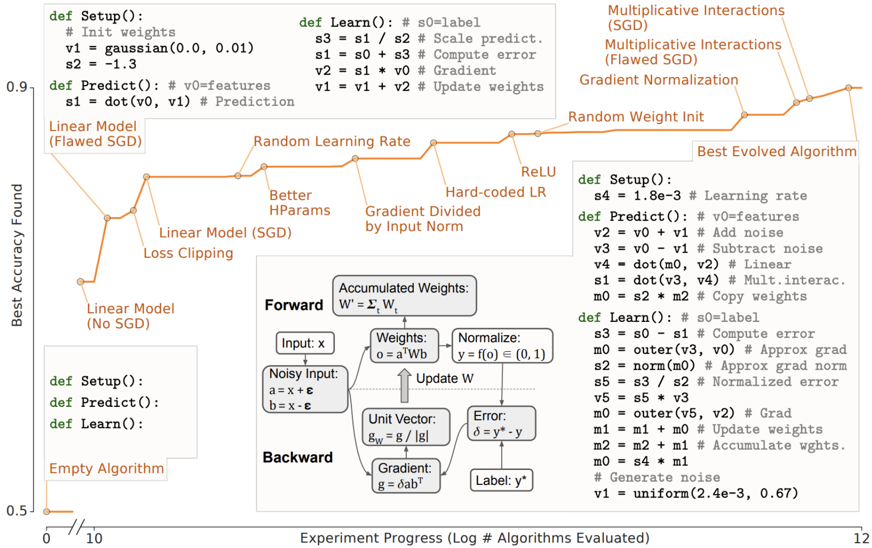



Figure: Progress of one evolution experiment on projected binary CIFAR-10.

# Results

## Emerging techniques

- Regularization

$$\mathbf{a} = \mathbf{x} + \mathbf{u}, \quad \mathbf{b} = \mathbf{x} - \mathbf{u}, \quad \mathbf{u} = \mathbf{U}(\alpha, \beta)$$

- Multiplicative interactions

$$\mathbf{o} = \mathbf{a}^T \mathbf{W} \mathbf{b}$$

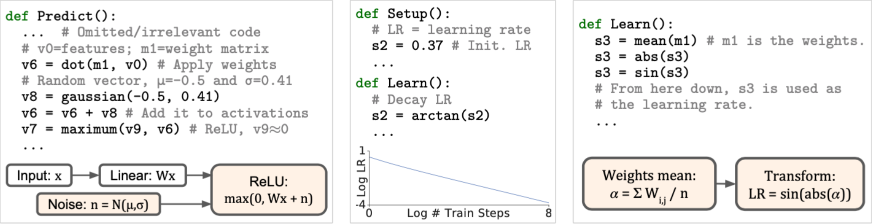
- Gradient normalization

$$\mathbf{g_w} = \frac{\mathbf{g}}{|\mathbf{g}|}; \quad \mathbf{g} = \delta \mathbf{a} \mathbf{b}^T; \quad \delta = \mathbf{y}^* - \mathbf{y}$$

- Weight averaging

$$\mathbf{W}' = \sum_{\mathbf{t}} \mathbf{W_t}$$

# Results




(a) Adaptation to few examples.

(b) Adaptation to fast training.

(c) Adaptation to multiple classes.

Figure: Adaptations to different task types.

# Conclusion

## Discussion

- The search method scalability
- Evaluating evolved algorithms
- Interpreting evolved algorithms
- Search space enhancements

# Literature

1. **Main article** AutoML-Zero: Evolving Machine Learning Algorithms From Scratch.