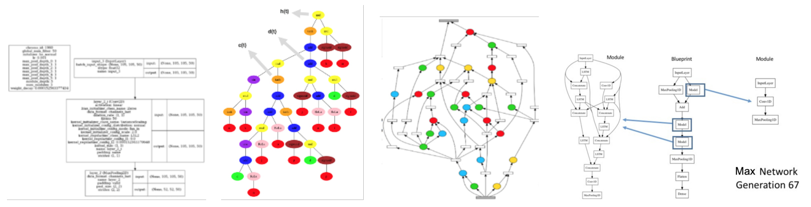
Metalearning in Neuroevolution

Risto Miikkulainen

November 6, 2024

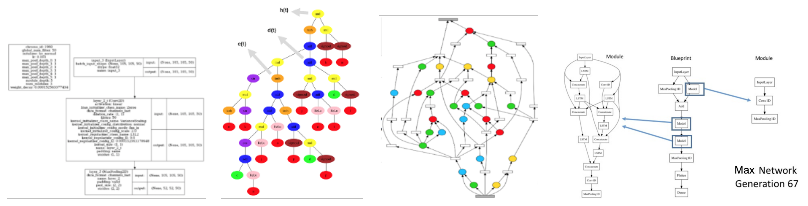
Introduction to Metalearning

- Metalearning means optimizing designs of learning systems.
  - "Learning to learn", configuration, hierarchical adaptation.
  - Too complex to optimize by hand.
- Many techniques can be used for metalearning:
  - Gradient descent; Bayesian optimization; Reinforcement learning; Evolutionary optimization.
  - Why evolution?



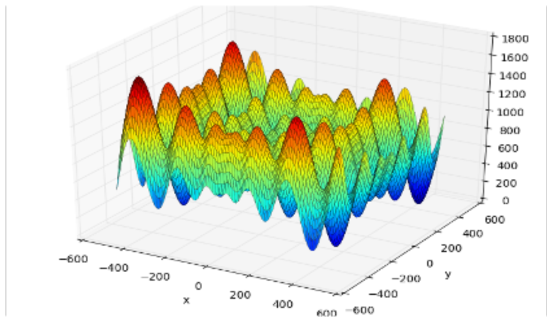
1. Versatility

- Evolution is versatile and comprehensive:
- Can be applied to Continuous values; discrete values; graph structures; design choices.
- All of those are aspects of neural network designs.



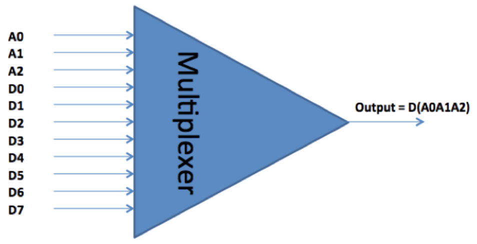
2. Creativity

- Evolution is creative:
- Allows finding solutions in deceptive, multi-peak landscapes.
- Multiobjective search; novelty search; neutral mutations.



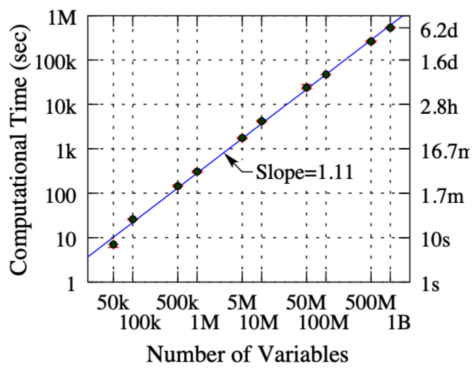
3. Scaling to Large Search Spaces

- Evolution scales to large search spaces.
- Example: 70-bit multiplexer with $2^{2^{70}}$ solutions.
  - The universe is minuscule compared to this search space.
  - Evolution leverages partial solutions as stepping stones.



4. Scaling to High-Dimensional Problems

- Evolution can optimize many variables at once.
- Humans can handle 7+/-2; Bayesian optimization about 10-15.
- Example: Metal casting scheduling process with many objects and many heats.
  - Make desired number of objects with max heat size.
  - Multiple constraint bounded knapsack problem that can be scaled.
  - With special crossover and mutation operators, scales to 1B parameters.



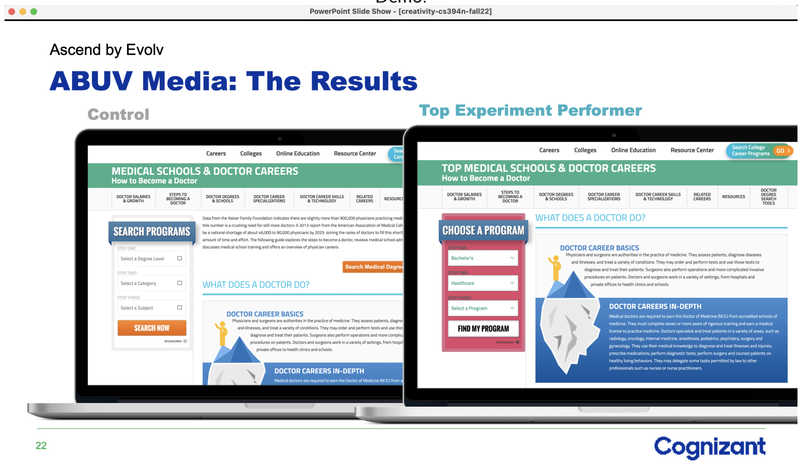
Result: Complexity and Deceptive Interactions

- Many real-world problems involve non-linear and deceptive interactions.
- Example: Shinkansen bullet train nose design.
- Evolutionary search can discover unconventional but effective solutions.

Result: Unlikely and Surprising Solutions

- Evolution discovers counterintuitive solutions in various fields.
- Examples: Improving principled human design by 45% in a webpage optimization task.

Demo:


PowerPoint Slide Show - [creativity-cs394n-fall22]

Ascend by Evolv

ABUV Media: The Results

Control

Top Experiment Performer

Physicians and surgeons are authorities in the practice of medicine. They assess patients, diagnose diseases and illnesses, and treat a variety of conditions. They may order and perform tests and use those tests to diagnose and treat their patients. Surgeons also perform operations and more complicated invasive procedures on patients. Doctors and surgeons work in a variety of settings, from hospitals and private offices to health clinics and schools.

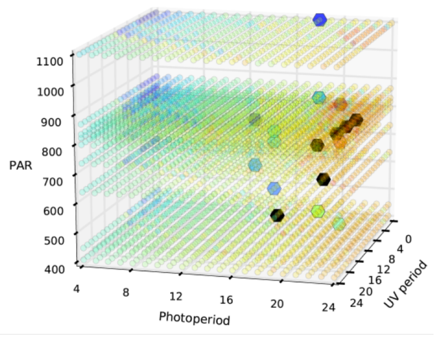
Medical doctors are required to earn the Doctor of Medicine (M.D.) from accredited schools of medicine. They must complete seven or more years of rigorous training and earn a medical license to practice medicine. Doctors specialize and treat patients in a variety of areas, such as radiology, oncology, internal medicine, anesthesia, pediatrics, psychiatry, surgery and gynecology. They use their medical knowledge to diagnose and treat illnesses and injuries; prescribe medications; perform diagnostic tests; perform surgery and counsel patients on healthy living behaviors. They may delegate some tasks permitted by law to other professionals such as nurses or nurse practitioners.

22

Cognizant

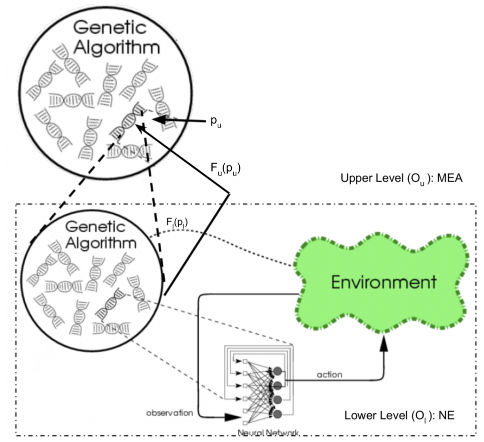
Result: New Scientific Discoveries

- Creativity beyond human-designed boundaries.
- LLMs are unlikely to make such discoveries
  - They cannot tell when they hallucinate.
  - They cannot tell what is new.
- Example: Discovering that basil does not need to sleep.



Putting Metalearning to Work I: Bilevel Neuroevolution

- Optimizes parameters in nested optimization problems.
- Upper level: Finds best parameters for lower-level neuroevolution.
- Goal: Automate parameter search for better neuroevolution results.

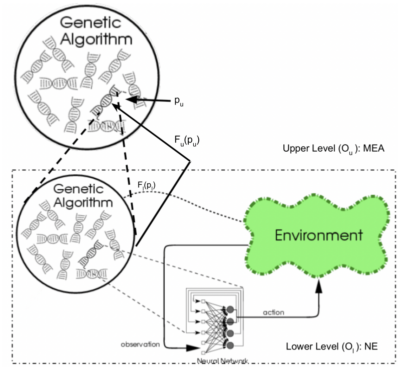


Formalization of Bilevel Optimization

$$\underset{p_u}{\text{maximize}}\, F_u(p_u) = E[F_l(p_l)|(p_u)] \quad (1)$$

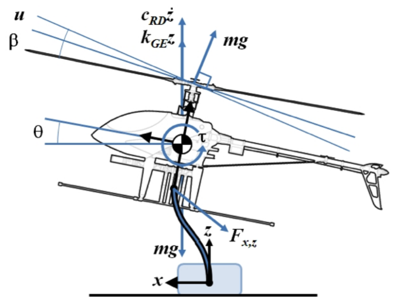
$$\text{subject to } p_l = O_l(p_u) \quad (2)$$

- p_u: Upper-level parameters, p_l: Lower-level parameters.
- O_l: Lower-level optimization, O_u: Upper-level optimization.



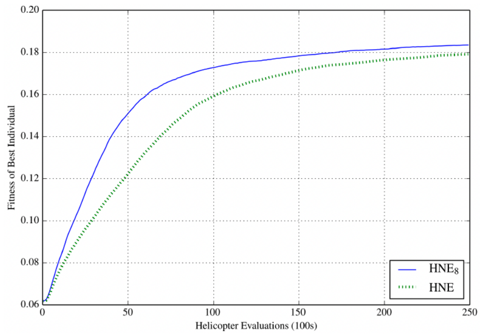
Application in Helicopter Hovering

- Bilevel optimization can improve performance in control tasks.
- Task: Keep helicopter hovering despite disturbances.
- Neuroevolution optimizes parameters for stability and control.



Advantages Over Hand-Tuning

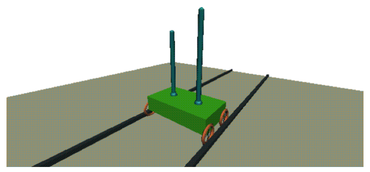
- Eight neuroevolution parameters: mutation probability, rate, amount, replacement rate and fraction, population size, crossover probability, and crossover averaging rate.
- Hand-tuned parameter values in prior work.
- Bilevel optimization finds superior parameter values.

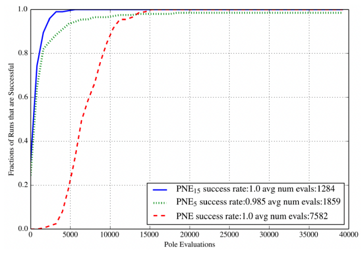


`https://nn.cs.utexas.edu/?liang_gecco_demo`

Increased Complexity is Useful

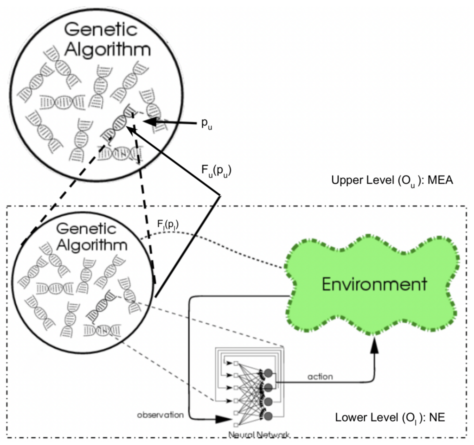
- Makes it possible to optimize more parameters.
- E.g. in the double pole-balancing, 15 instead of 5 or 0.
- Humans are limited to about 7 +/1 2
- Finds better solutions with more parameters.




Fractions of Runs that are Successful
Pole Evaluations
PNE$_{15}$ success rate:1.0 avg num evals:1284
PNE$_5$ success rate:0.985 avg num evals:1859
PNE success rate:1.0 avg num evals:7582

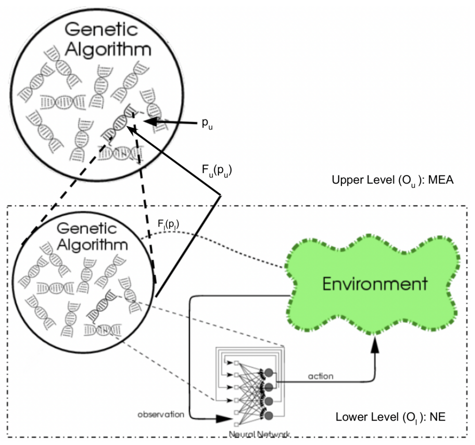
Bilevel Neuroevolution with Surrogate Models

- Bilevel evolution is expensive if fully nested.
- Surrogate models predict fitness without full neuroevolution.
  - Quadratic functions, random forests, and neural networks.
  - Need to obtain enough training data to guide upper-level evolution

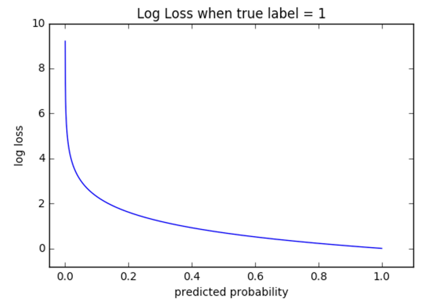


Extending Bilevel Optimization

- Adapting parameters online during the optimization.
- Multiple levels of optimization.



Putting Metalearning to Work II. Loss-Function Optimization



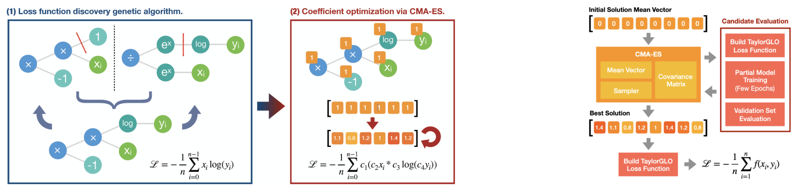
Loss function defines the learning goal in supervised learning

- Need to be amenable to gradient descent
- Need to define desirable minima

Cross-entropy, or log loss is most common, e.g. in classification:

$$\mathcal{L}_{\text{Log}} = -\frac{1}{n}\sum_{i=0}^{n-1} x_i \log(y_i)$$

Evolution of Loss Functions



(1) Loss function discovery genetic algorithm.
(2) Coefficient optimization via CMA-ES.
Initial Solution Mean Vector
CMA-ES
Mean Vector
Covariance Matrix
Sampler
Best Solution
Candidate Evaluation
Build TaylorGLO Loss Function
Partial Model Training (Few Epochs)
Validation Set Evaluation
Build TaylorGLO Loss Function
$\mathscr{L} = -\frac{1}{n}\sum_{i=0}^{n-1} x_i \log(y_i)$
$\mathscr{L} = -\frac{1}{n}\sum_{i=0}^{n-1} c_1(c_2 x_i * c_3 \log(c_4 y_i))$
$\mathscr{L} = -\frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i)$

GLO: Evolve the function through Genetic Programming

- Optimize coefficients with CMA-ES (Gonzalez and Miikkulainen 2020)

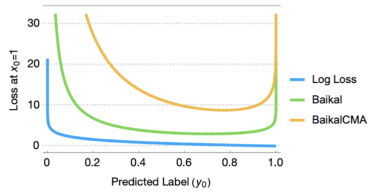
TaylorGLO: Optimize coefficients of a Taylor approximation

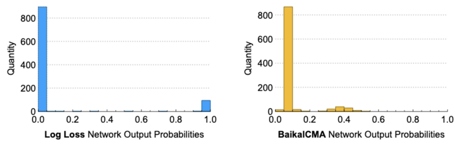
- CMA-ES on 3rd order approximation (Gonzalez and Miikkulainen 2021a)

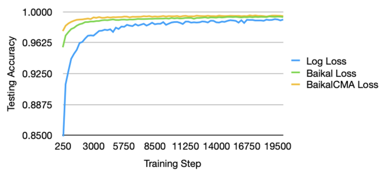
For a given learning task and network architecture

- E.g. MNIST, CIFAR10, SVHN with AllCNN, WRN, PreResNet
- Fitness based on validation accuracy

A Surprising Result: Baikal Loss







Loss increases close to the correct label!

Results in regularization

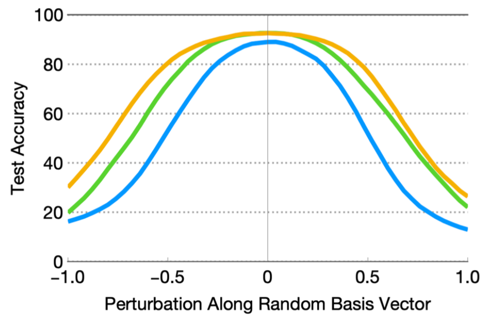
- Prevents overfitting to the correct label
- Still allows for correct class to be identified

Difficult to discover by hand

Enhances performance on unseen data.

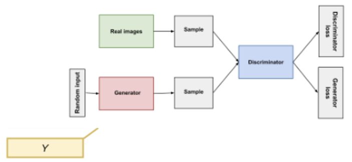
Regularization and Robustness with Baikal Loss

- Baikal loss improves network robustness against perturbations.
- Helps maintain performance with adversarial inputs.
- Enables better generalization in low-data environments.



Generative Adversarial Networks (GANs)

- (Goodfellow et al. 2014)
- A generator creates images
- A discriminator decides whether generated or real
- Indirect training of generator through the discriminator



- Difficult to train: stability, mode collapse, slow
- Can produce impressive images
- Can be directed through conditioning with additional input

Generalization: Loss Functions for GANs

Formulation	Loss D (real) $\mathbb{E}_{x\sim\mathbb{P}_{\text{data}}}$	Loss D (fake) $\mathbb{E}_{z\sim\mathbb{P}_z}$	Loss G (fake) $\mathbb{E}_{z\sim\mathbb{P}_z}$
GAN (minimax) [9]	$-\log D(x)$	$-\log(1-D(G(z)))$	$\log(1-D(G(z)))$
GAN (non-saturating) [9]	$-\log D(x)$	$-\log(1-D(G(z)))$	$-\log D(G(z))$
WGAN [2]	$-D(x)$	$D(G(z))$	$-D(G(z))$
LSGAN [21]	$\frac{1}{2}(D(x)-1)^2$	$\frac{1}{2}(D(G(z)))^2$	$\frac{1}{2}(D(G(z))-1)^2$

GANs are difficult to train
- Mode collapse and instabilities
- Many proposals for loss functions

Optimize D(real), D(fake), and G(fake) loss functions separately
- 3rd order TaylorGLO (Gonzalez et al. 2021b)

Loss Functions for GANs

$$\mathcal{L}_{D_{\text{real}}} = 5.6484\ (D(\boldsymbol{x}) - 8.3399) + 9.4935\ (D(\boldsymbol{x}) - 8.3399)^2 + 8.2695\ (D(\boldsymbol{x}) - 8.3399)^3$$

$$\mathcal{L}_{D_{\text{fake}}} = 6.7549\ (D(G(\boldsymbol{z})) - 8.6177) + 2.4328\ (D(G(\boldsymbol{z})) - 8.6177)^2 + 8.0006\ (D(G(\boldsymbol{z})) - 8.6177)^3$$

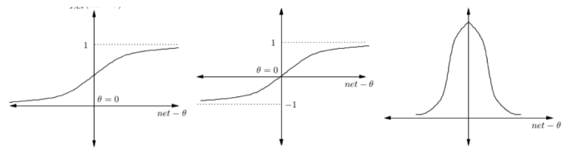
$$\mathcal{L}_{G_{\text{fake}}} = 0.0000\ (D(G(\boldsymbol{z})) - 5.2232) + 5.2849\ (D(G(\boldsymbol{z})) - 5.2232)^2 + 0.0000\ (D(G(\boldsymbol{z})) - 5.2232)^3$$

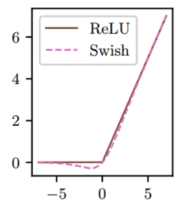
CMP Façade dataset, pix2pix-HD model, 2 metrics:
- Structural similarity index (SSIM) btw G and ground truth
- Perceptual distance: L1 btw VGG-16 ImageNet embeddings

TaylorGAN improves on both metrics against Wasserstein
- 11.66 vs. 9.43; 2029 vs. 2040
- Better color; better boundaries/sky; better detail

Synergy of evolution and learning:
- Optimize the goal of learning

Putting Metalearning to Work III. Activation Function Optimization



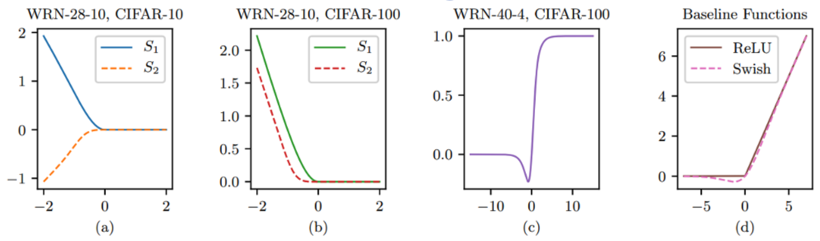


Originally sigmoid, tanh, Gaussian
- Representation theorems, effective on shallow networks

ReLU became essential for Deep Learning (Nair & Hinton 2010)
- Helped with vanishing gradients; faster to compute

Even small modifications can have large effects:
- E.g. Swish improves over ReLU in many tasks (Ramachandran et al. 2018)

Activation Function Metalearning



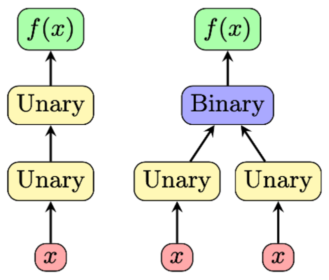
Can we find better general functions?
Can we specialize functions to tasks and architectures?
A possible synergy:

- Evolve function shapes
- Optimize parameters with gradient descent

PANGAEA (Bingham & Miikkulainen 2020)

Search Space

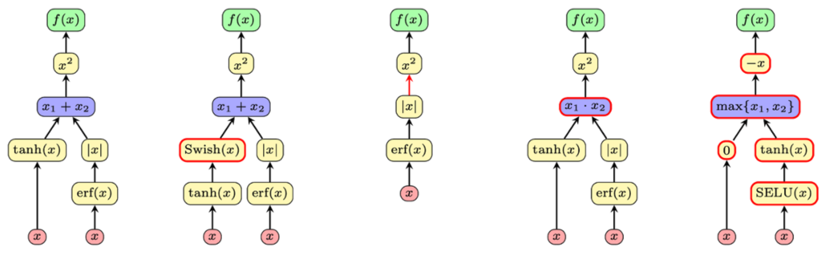
- Two kinds of computation graphs
- Randomly initialized with unary and binary operators



Unary							Binary	
0	$\lvert x\rvert$	erf(x)	tanh(x)	arcsinh(x)	ReLU(x)	Softplus(x)	$x_1 + x_2$	$x_1^{x_2}$
1	x^{-1}	erfc(x)	$e^x - 1$	arctanh(x)	ELU(x)	Softsign(x)	$x_1 - x_2$	$\max\{x_1, x_2\}$
x	x^2	sinh(x)	$\sigma(x)$	bessel_i0e(x)	SELU(x)	HardSigmoid(x)	$x_1 \cdot x_2$	$\min\{x_1, x_2\}$
$-x$	e^x	cosh(x)	$\log(\sigma(x))$	bessel_i1e(x)	Swish(x)		x_1/x_2	

Evolution of Function Shapes

Given a parent activation function, one of four mutations is randomly applied tc produce a child activation function.

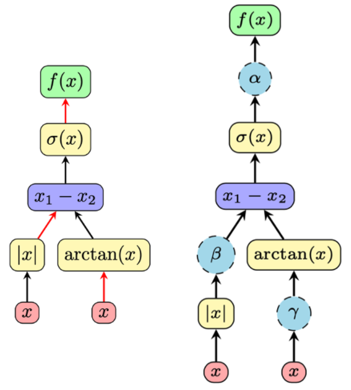


(a) Parent (b) Insert (c) Remove (d) Change (e) Regenerate

Learning of Parameters

$\alpha\sigma(\beta|x| - \arctan(\gamma x))$

- Parameterization allows fine-tuning during backpropagation
- Per-channel learnable parameters are inserted at up to three random edges



Evolutionary Progress

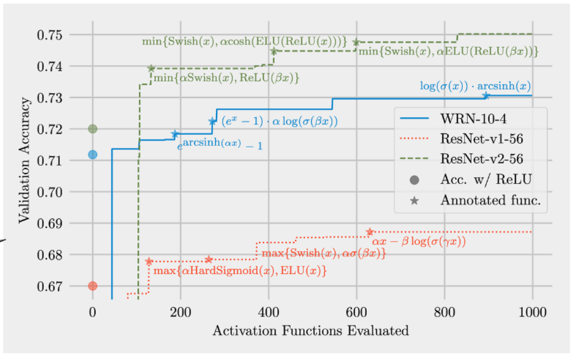
Architectures

- Wide ResNet (WRN-10-4)
- ResNet (ResNet-v1-56)
- Preactivation ResNet (ResNet-v2-56)

Dataset: CIFAR-100

Plot shows best accuracy with all functions evaluated so far.

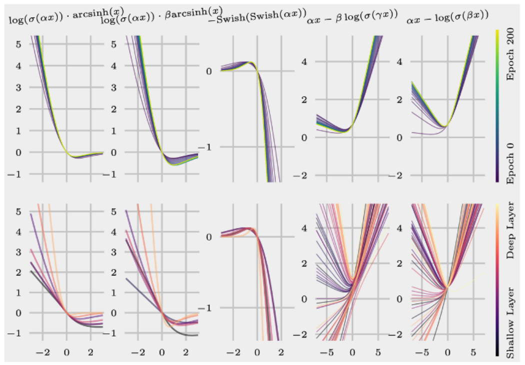
3 independent runs → selective pressure to discover activation functions specialized to each model



Parametric Activation Functions across Space and Time

Parameters change during training, resulting in different activation functions in the early and late stages.

Parameters are separate across channels, inducing different activation functions at different locations in a network.



Performance

Specialized functions

- Achieve the highest performance with architecture for which they were evolved
- May not transfer across architectures

General functions

- Achieve good performance on multiple architectures
- Outperform most baselines, but not the best specialized functions

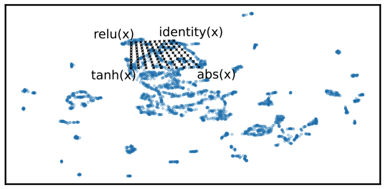
Synergy of evolution and learning:
Functional form and fine-tuning over space and time

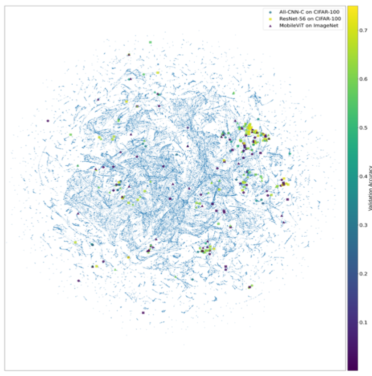
	WRN-10-4	ResNet-v1-56	ResNet-v2-56
Specialized for WRN-10-4			
$\log(\sigma(\alpha x)) \cdot \text{arcsinh}(x)$	**73.23** (73.16 ± 0.41) *** +++	11.15 (19.34 ± 20.14)	72.05 (64.30 ± 21.32)
$\log(\sigma(\alpha x)) \cdot \beta\text{arcsinh}(x)$	73.22 (73.20 ± 0.37) *** +++	05.78 (18.63 ± 21.04)	55.40 (45.88 ± 30.70)
$-\text{Swish}(\text{Swish}(\alpha x))$	72.38 (72.49 ± 0.55) ***	59.61 (58.86 ± 2.88)	74.70 (74.71 ± 0.20) *
Specialized for ResNet-v1-56			
$\alpha x - \beta\log(\sigma(\gamma x))$	70.35 (70.28 ± 0.37)	**70.82** (71.01 ± 0.64) *** ++	74.41 (74.35 ± 0.45)
$\alpha x - \log(\sigma(\beta x))$	70.62 (70.47 ± 0.53)	70.30 (70.30 ± 0.58) *	74.73 (74.70 ± 0.23) *
$\max\{\text{Swish}(x), 0\}$	71.96 (72.10 ± 0.33) **	69.46 (69.43 ± 0.69)	74.97 (74.97 ± 0.25) **
Specialized for ResNet-v2-56			
$\text{Softplus}(\text{ELU}(x))$	71.51 (71.36 ± 0.34)	69.94 (69.96 ± 0.39)	**75.60** (75.61 ± 0.42) ***
$\min\{\log(\sigma(x)), \alpha\log(\sigma(\beta x))\}$	72.05 (72.04 ± 0.34) **	69.63 (69.56 ± 0.48)	75.20 (75.19 ± 0.39) ***
$\text{SELU}(\text{Swish}(x))$	01.00 (01.00 ± 0.00)	01.00 (01.00 ± 0.00)	75.06 (75.02 ± 0.35) **
General Activation Functions			
$\max\{\text{Swish}(x), \alpha\log(\sigma(\text{ReLU}(x)))\}$	72.50 (72.54 ± 0.26) ***	69.97 (69.91 ± 0.37)	75.21 (75.20 ± 0.41) ***
$\min\{\text{Swish}(x), \alpha\text{ELU}(\text{ReLU}(\beta x))\}$	72.44 (72.39 ± 0.29) ***	69.90 (69.82 ± 0.40)	75.20 (75.27 ± 0.38) ***
$\log(\sigma(x))$	72.38 (72.33 ± 0.32) ***	69.49 (69.58 ± 0.35)	75.45 (75.53 ± 0.37) ***
Baseline Activation Functions			
ReLU	71.44 (71.46 ± 0.50)	69.78 (69.64 ± 0.65)	74.43 (74.39 ± 0.44)
ELiSH	01.00 (01.00 ± 0.00)	01.00 (01.00 ± 0.00)	75.16 (75.20 ± 0.31) ***
ELU	72.41 (72.30 ± 0.32) ***	69.59 (69.67 ± 0.46)	74.86 (74.95 ± 0.30) **
GELU	72.00 (71.95 ± 0.35) *	70.16 (70.19 ± 0.40) *	74.84 (74.86 ± 0.33) **
HardSigmoid	55.55 (54.99 ± 1.00)	33.31 (32.55 ± 4.06)	65.03 (64.90 ± 0.69)
Leaky ReLU	71.76 (71.73 ± 0.33)	69.77 (69.78 ± 0.33)	74.75 (74.73 ± 0.35) *
Mish	72.02 (71.95 ± 0.41) *	70.03 (69.88 ± 0.54)	75.33 (75.32 ± 0.29) ***
SELU	70.55 (70.53 ± 0.42)	68.51 (68.52 ± 0.29)	73.86 (73.79 ± 0.36)
sigmoid	56.45 (56.10 ± 0.98)	37.07 (36.47 ± 3.32)	66.72 (66.45 ± 0.92)
Softplus	72.25 (72.27 ± 0.26) ***	69.71 (69.71 ± 0.36)	75.47 (75.46 ± 0.52) ***
Softsign	56.72 (56.30 ± 2.16)	58.33 (58.38 ± 0.96)	69.31 (69.33 ± 0.39)
Swish	72.27 (72.26 ± 0.28) ***	69.60 (69.68 ± 0.38)	75.17 (75.08 ± 0.36) ***
tanh	56.29 (56.52 ± 1.53)	63.89 (63.88 ± 0.38)	70.53 (70.44 ± 0.40)
Parametric Baseline Functions			
PReLU	72.25 (72.23 ± 0.37) ***	69.67 (69.77 ± 0.40)	74.99 (75.10 ± 0.53) **
$\text{PSwish} = x \cdot \sigma(\beta x)$	72.46 (72.40 ± 0.31) ***	70.19 (70.16 ± 0.46) *	75.37 (75.39 ± 0.28) ***

Statistical significance over ReLU (*, **, ***) and over all baselines (+, ++, +++)

Taking advantage of Benchmarks and Surrogates

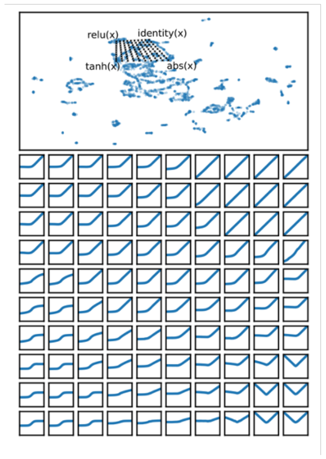
- Construct benchmark datasets by exploring large search spaces.
- Use insights from benchmarks to design surrogate models.
- Surrogates help search in larger spaces without full training.





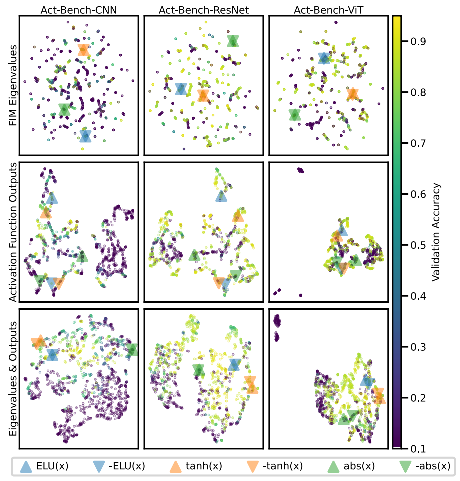
AQuaSurF Approach for Activation Function Discovery

- Exhaustive search with 2913 activation functions on different tasks.
- Activation functions tested across All-CNN, ResNet, and MobileViT architectures.
- Results: only a small subset performs well, with task-specific and general functions.



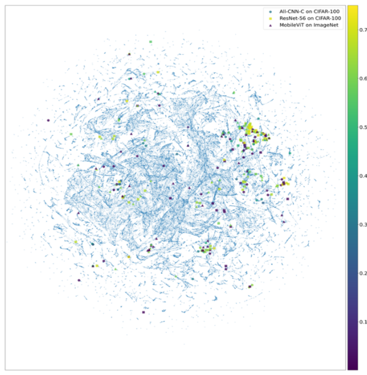
Surrogate Design

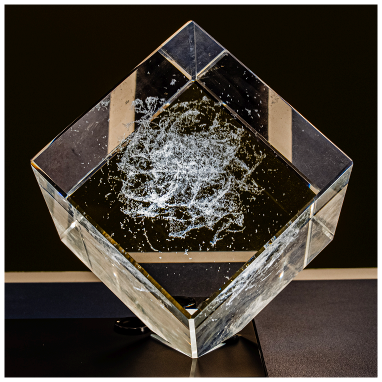
- Fisher Information Matrix (FIM) eigenvalues represent network behavior.
- Activation function shape offers a different characterization.
- Combined FIM and function shape create a powerful surrogate space.
- Similar functions cluster together, simplifying search for effective designs.



Scaling Up With Surrogates

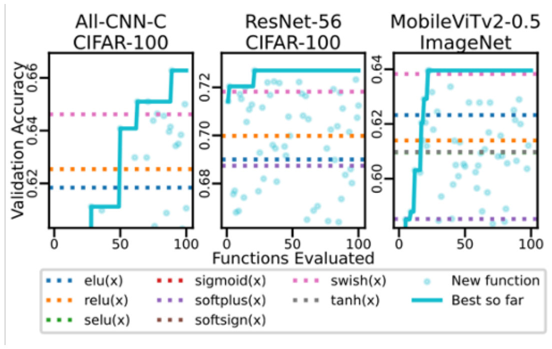
- With surrogates, can search a much larger space of 400,000 candidate functions.
- Original benchmark functions embedded as a small fraction.





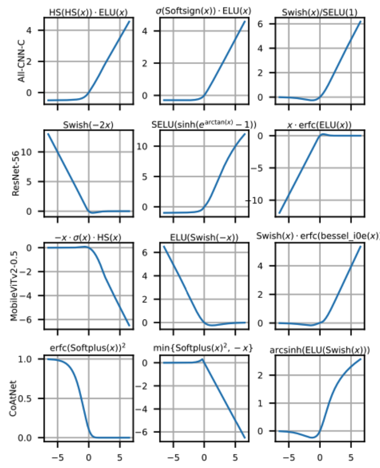
Discoveries and Transfer

- Discovers new functions that outperform all benchmark activation functions.
- These functions transfer effectively to new datasets and architectures.

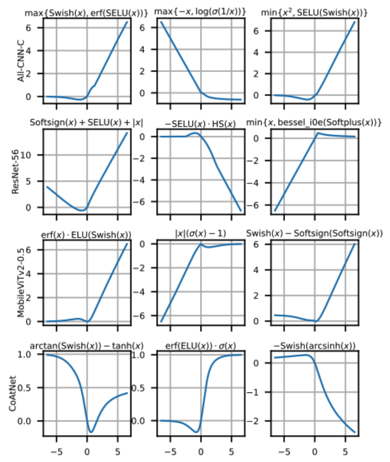


Balancing Refinement and Novelty

- AQuaSurF finds both refined and novel activation functions.
- Customizations of known functions (ELU, Swish) and entirely new ones.
- Functions with unique derivatives and saturation patterns.



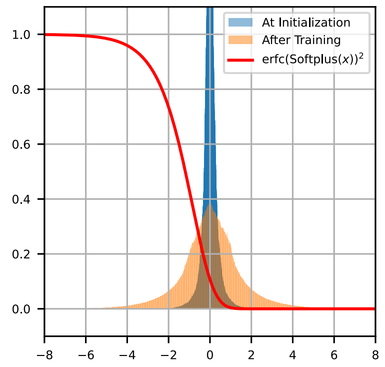
Top-3



Novel exceeding ReLU

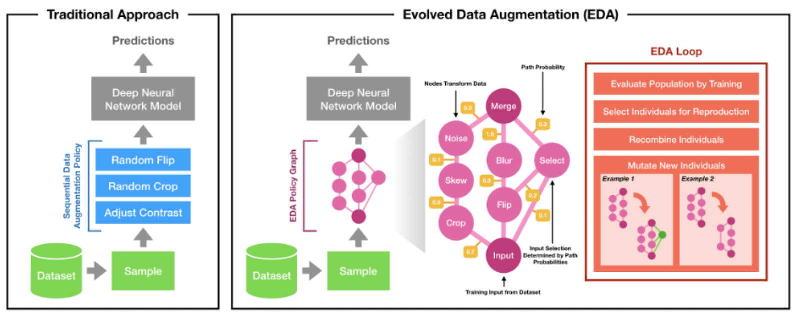
Surprise: Sigmoid Rediscovery in AQuaSurF

- Rediscovered sigmoid activation in CoAtNet for specific tasks.
- Input values utilize it like a ReLU in early training.
- Input values utilize it like a sigmoid (that saturates) in later training.
- Demonstrates potential of metalearning to challenge deep learning conventions.



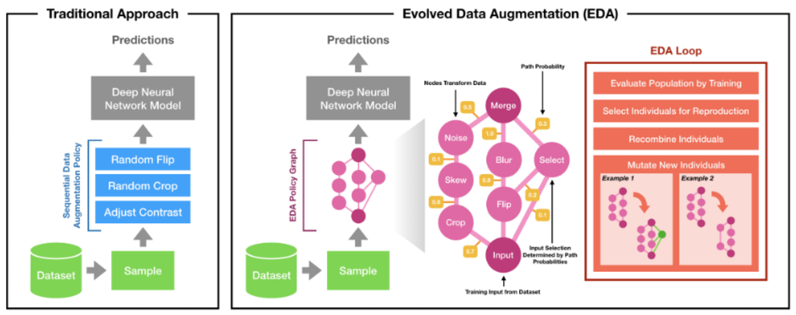
Putting Metalearning to Work IV: Evolving Data Augmentation Strategies

- Data augmentation optimizes the coverage of training data.
- Usually random modifications of existing samples.
- Ensures comprehensive interpolation between samples.



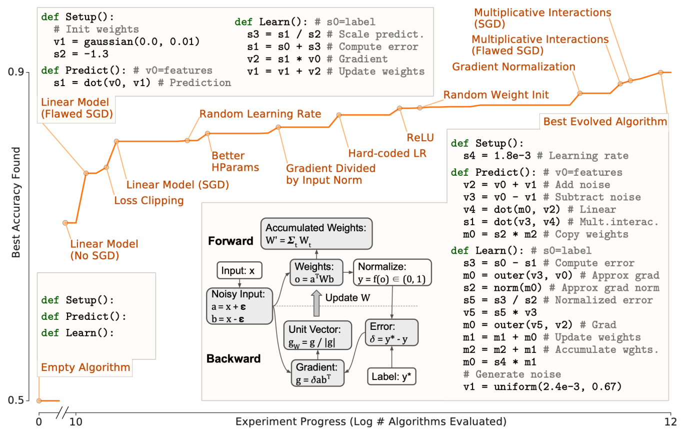
Extending to Evolutionary Data Augmentation

- Evolution can optimize augmentation techniques based on task requirements.
- Generate difficult examples that result in better learning.
- Can also be used to optimize secondary objectives.
- Mitigate bias, optimize fairness, and increase sample diversity.



Putting Metalearning to Work V: Evolving Learning Methods

- AutoML-Zero evolves learning methods from fundamental operations.
- Methods for setup, predict, and learn, using basic mathematical operations.
- Genetic programming of code for CIFAR-10 object classification.



```
def Setup():
def Predict():
def Learn():
```

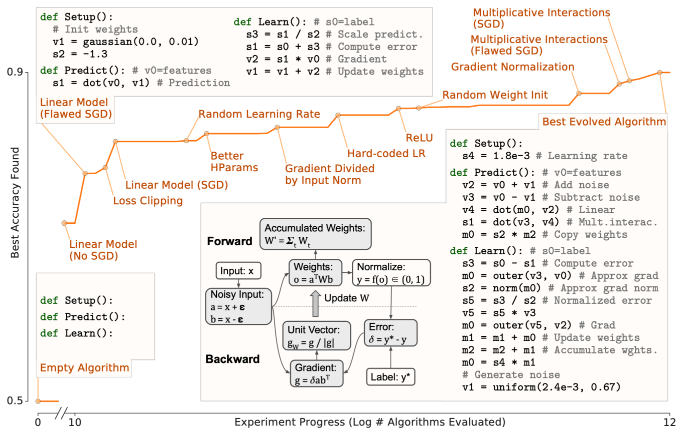
```
def Setup():
  # Init weights
  v1 = gaussian(0.0, 0.01)
  s2 = -1.3
def Predict(): # v0=features
  s1 = dot(v0, v1) # Prediction
```

```
def Learn(): # s0=label
  s3 = s1 / s2 # Scale predict.
  s1 = s0 + s3 # Compute error
  v2 = s1 * v0 # Gradient
  v1 = v1 + v2 # Update weights
```

```
def Setup():
  s4 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```

Discoveries in AutoML-Zero

- AutoML-Zero discovered linear models and gradient descent from scratch.
- Improvement features like regularization, learning-rate decay, ReLU.
- Demonstrates potential for evolving complex learning mechanisms.


def Setup():
def Predict():
def Learn():

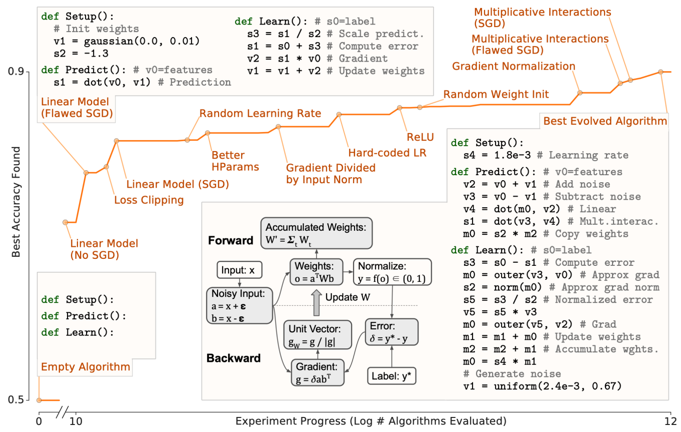
def Setup():
  # Init weights
  v1 = gaussian(0.0, 0.01)
  s2 = -1.3
def Predict(): # v0=features
  s1 = dot(v0, v1) # Prediction

def Learn(): # s0=label
  s3 = s1 / s2 # Scale predict.
  s1 = s0 + s3 # Compute error
  v2 = s1 * v0 # Gradient
  v1 = v1 + v2 # Update weights

def Setup():
  s4 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)

Potential of Evolved Learning Algorithms

- Evolution must be guided within large search spaces for meaningful solutions.
- Potential in customizing methods for specific domains.
- Future work: balancing freedom of creation with solution relevance.


def Setup():
def Predict():
def Learn():

def Setup():
  # Init weights
  v1 = gaussian(0.0, 0.01)
  s2 = -1.3
def Predict(): # v0=features
  s1 = dot(v0, v1) # Prediction

def Learn(): # s0=label
  s3 = s1 / s2 # Scale predict.
  s1 = s0 + s3 # Compute error
  v2 = s1 * v0 # Gradient
  v1 = v1 + v2 # Update weights

def Setup():
  s4 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)