Machine learning in the string landscape

FABIAN RUEHLE (UNIVERSITY OF OXFORD)

CERN String Theory Seminar
12/09/2017

Based on [1706.07024]

UNIVERSITY OF OXFORD

Data sets in String Theory

- String theorists have produced large sets of data / samples of the string landscape over the years
  - Calabi-Yau manifolds
    - CICYs in 3D and 4D [Candelas,Dale,Lutken,Schimmrigk'88; Gray,Haupt,Lukas'13]
    - Kreuzer-Skarke database [Kreuzer,Skarke'00]
    - Toric bases for F-Theory [Morrison,Taylor'12]
  - String models
    - Type IIA/IIB models [Gmeiner,Blumenhagen,Honecker,Lust,Weigand'06; Davey,Hanany,Pasukonis'09; Franco,Lee,Seong,Vafa'16; …]
    - Heterotic on CY/Orbifolds/Free fermionic [Anderson,Constantin,Gray,Lukas,Palti'13; Nilles,Vaudrevange'14; Abel,Rizos'14; Blaszczyk,Groot Nibbelink,Loukas,FR'15; …]
    - F-Theory [Taylor,Wang'15; Halverson,Tian'16; Halverson,Long,Sung'17; …]

Study String Vacua

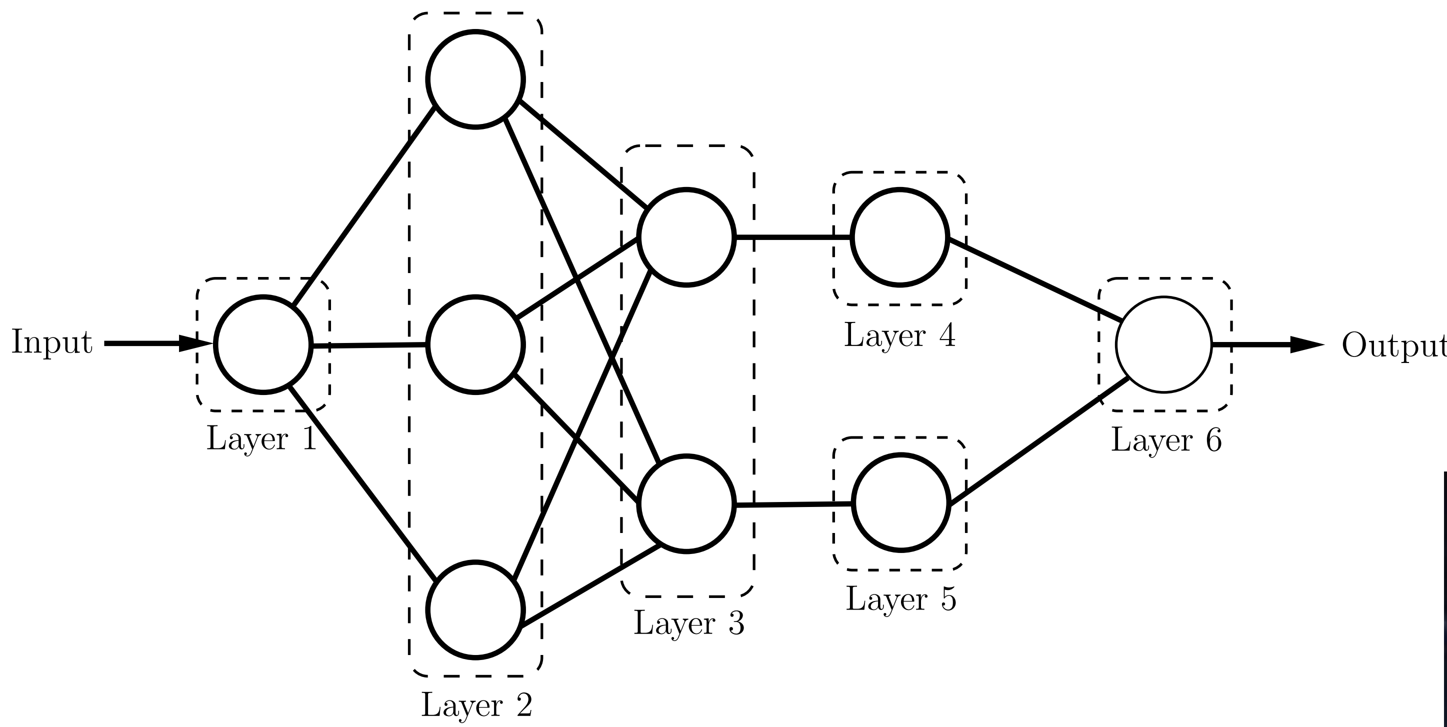
- Currently no selection mechanism for string vacua $\Rightarrow$ vast string landscape
- Long term goal for: Map out the landscape
  - Find string models in the landscape
  - Find generic / common features of string-derived model
  - Extract string theory predictions from the landscape
  - Are low energy manifestations of string vacua linked?
  - Find new relations/mathematical theorems from string theory
- Can we use neural networks (NNs) to answer or study such questions? [He'17; Krefl,Seong'17; FR'17; Carifio,Halverson,Krioukov,Nelson'17]

Study String Vacua

- Starting point: 12D/11D/10D F/M/I-IIA-IIB-HE-HS $\Rightarrow$ (rather) unique
- Phenomenology of the model encoded in discrete (background) choices / data (compactification space, fluxes, …)
- Given this data, can one decide whether a model has
  - SM gauge group
  - three generations with one pair of vector-like Higgs
  - correct Yukawa textures
  - 60 e-folds of slow-roll inflation
  - a Minkowski/de-Sitter vacuum solution
  - …
- In principle possible (computable for a given choice), but I cannot see it directly from the input data $\Rightarrow$ can a NN decide / compute (some of) these?
- If so, how can we find most efficient NNs for the job? $\Rightarrow$ Genetic algorithms

Outline

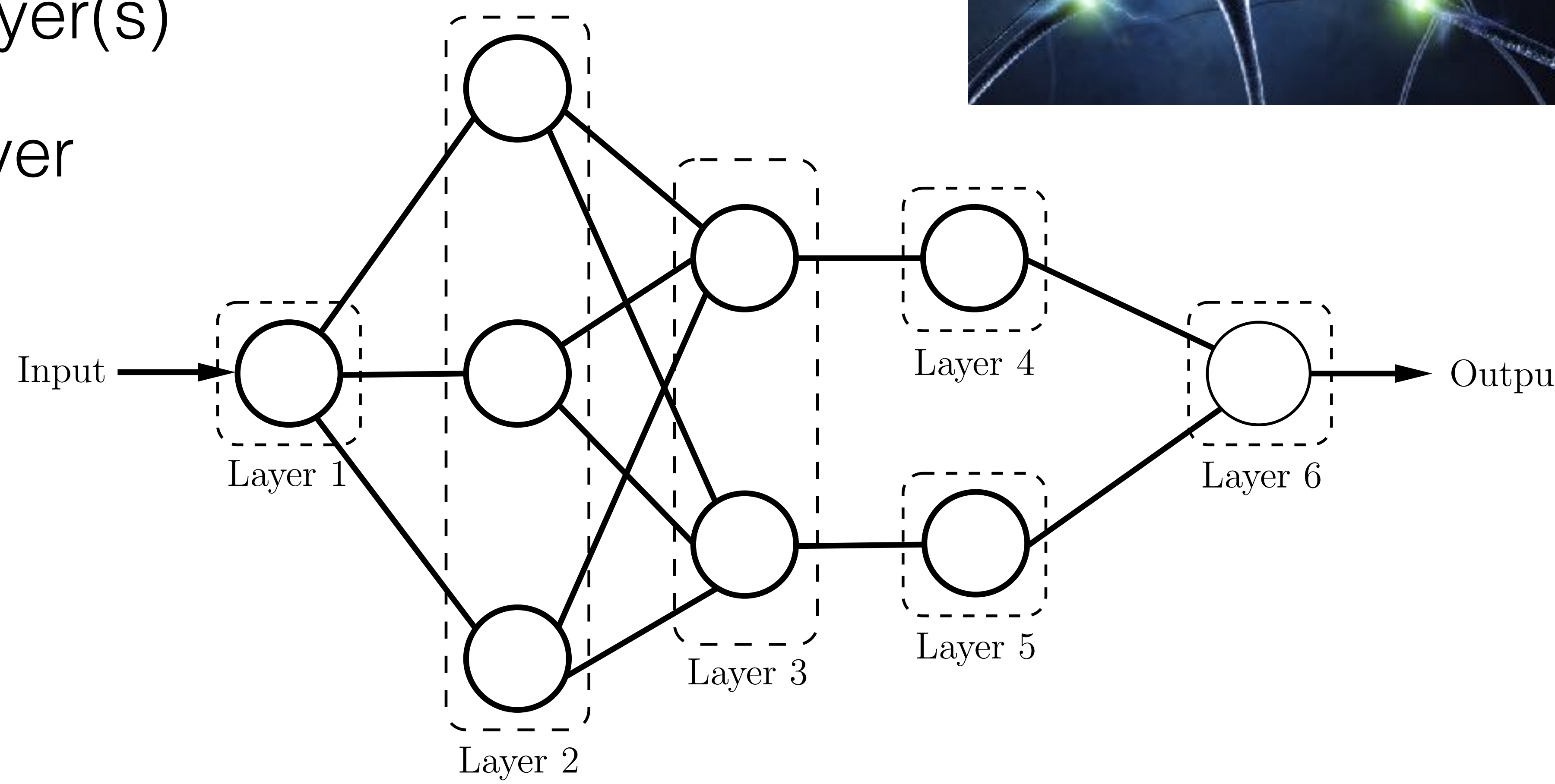
- Introduction to Neural Networks
  - Neural Networks 101 - How/why do they work
  - Where can we apply neural networks
- Genetic Algorithms 101
- Combining both approaches
  - Example: Classifying stable line bundles
  - Example: Computing line bundle cohomology
- Conclusions



Introduction to Neural Networks

Neural Networks 101

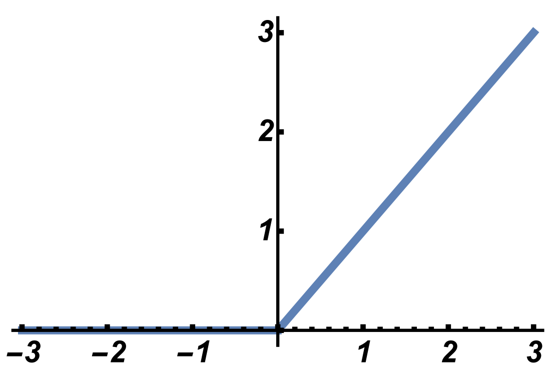
- Copy nature ⇒ modelled after human brain
- Building blocks
  - Input layer
  - Hidden layer(s)
  - Output layer



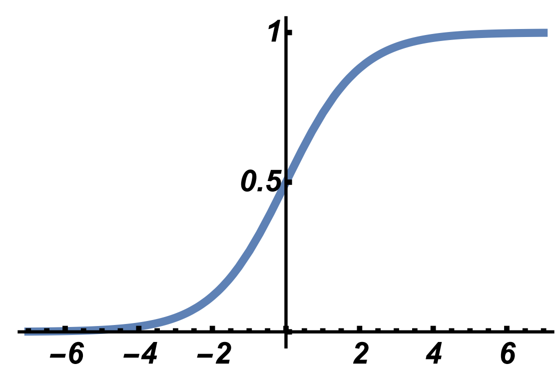
Neural Networks 101

- Connection between layers : Linear transformations L_i:
  Matrix multiplication $v^i_{\text{out}} = A^i v^i_{\text{in}} + b^i$
- Each layer applies a function (activation function) to its input to compute its output. Common choices are

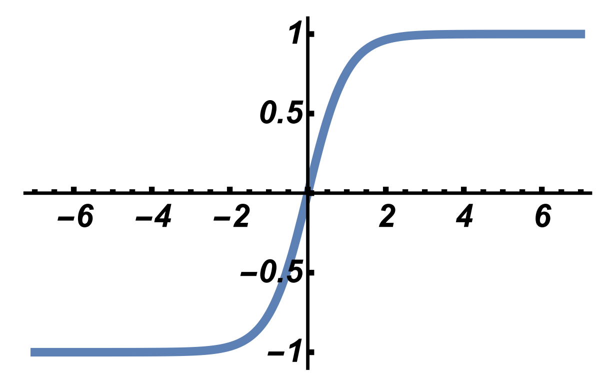
Ramp



Logistic Sigmoid



Tanh

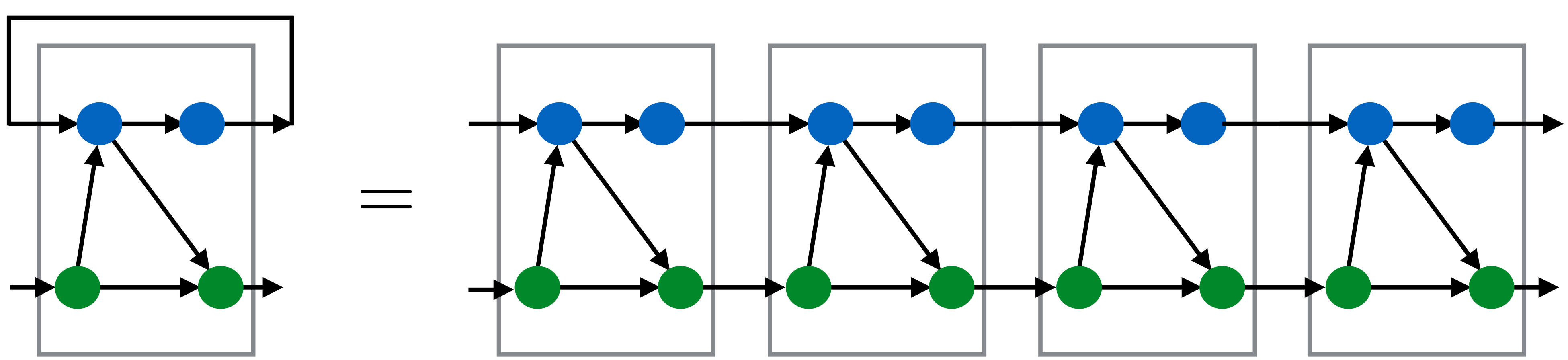


- Typical NN: $\mathbb{R}^M \rightarrow \mathbb{R}^N$

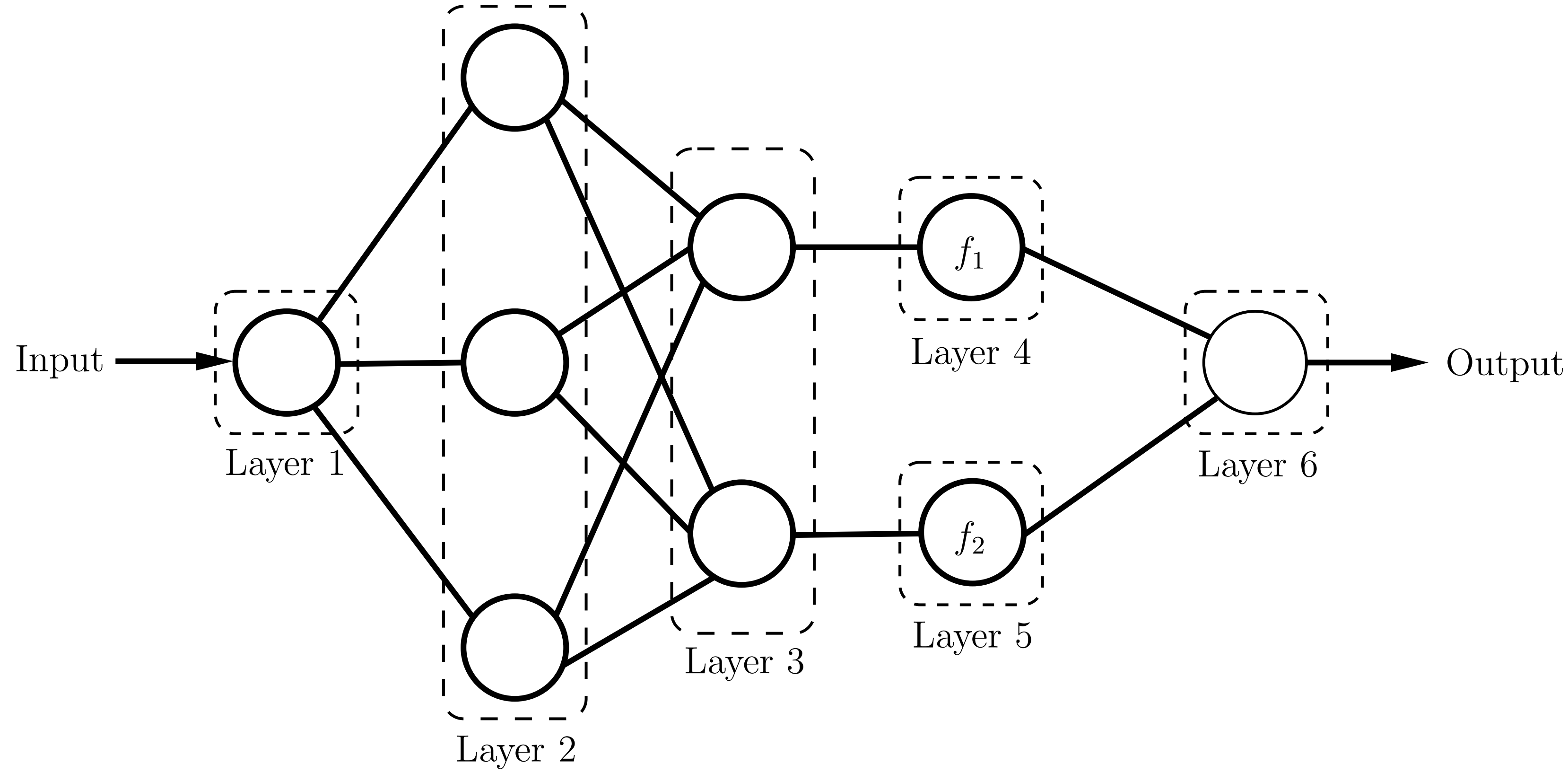
$$v \mapsto f_n \circ L_n \circ \ldots \circ f_0 \circ L_0$$

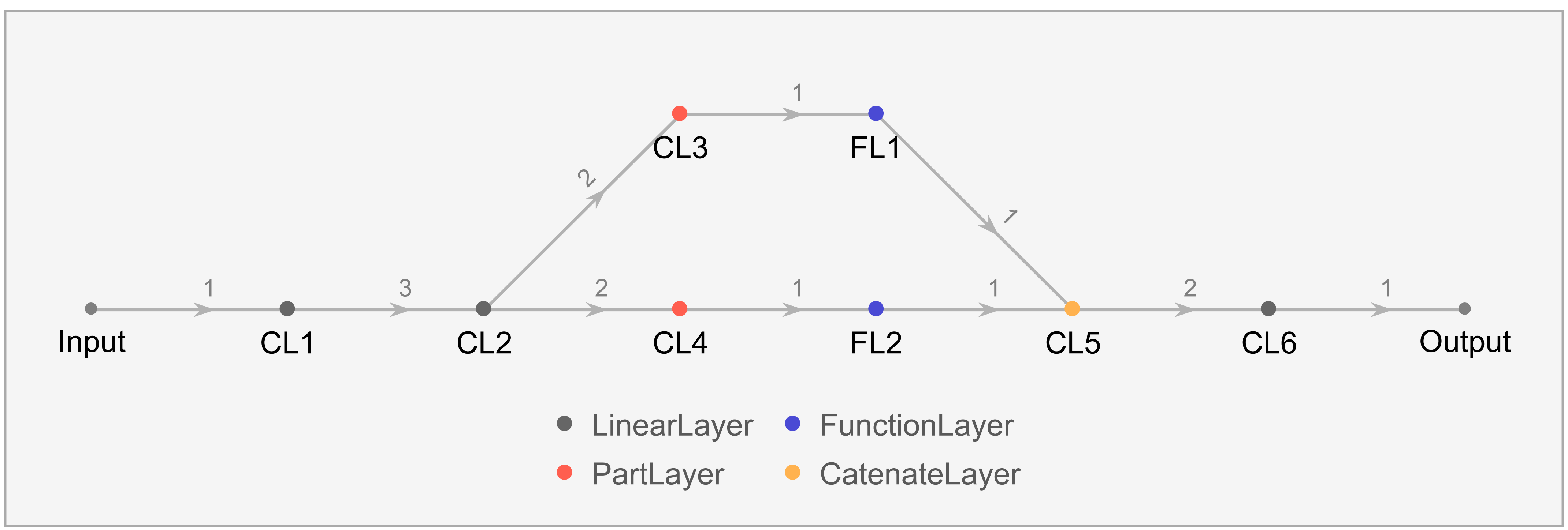
Modifications / Extensions

- Do not connect all outputs of layer i to all inputs of layer $i+1$
  - Add / multiply / concat results of parallel NN streams
- Create loops
  - Feed output of an NN layer back into its input
  - Recurrent NN $\Rightarrow$ Give the network a memory (LSTM layers)



Example NN


Input
Output
f_1
f_2
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Layer 6


Input
CL1
CL2
CL3
CL4
FL1
FL2
CL5
CL6
Output
1
3
2
2
1
1
1
1
2
1
LinearLayer
FunctionLayer
PartLayer
CatenateLayer

Neural Networks - Training

- Precise way in which NN learn active field of research
- In *supervised* ML you show the network the correct results
- In *unsupervised* ML you let the network find common properties (clustering) and identify things that “don’t fit in” by itself
- In this talk: supervised ML:
    - Divide data set into a training set (30% of data) and validation set (70% of data)
    - Randomly initialize the trainable parameters of the NN (e.g. weights and biases of the connections)
    - Let the network look at the entire train set (inputs and outputs) and minimize the difference between its output and the output of the training set (w.r.t. some metric) “back-propagation”
    - Shuffle the training set and repeat until
        - Error is not significantly reduced anymore
        - Each training model has been used a set number of times
        - A certain amount of time has elapsed
    - Cross-check performance of trained NN against the validation set

Neural Networks - Applications

- Three ways of applying neural networks

  (A) To *find & bypass* implementations of algorithms (in combination with genetic algorithms)

  (B) To approximate functions (*predictor*)

  (C) To *classify* outcome of some complicated / unknown mathematical operation based on the structure of the input

- Once we have built a neural network to apply to (A) - (C) we need to train it

- Once trained NNs can perform very efficiently

  - They just apply simple functions to produce some output

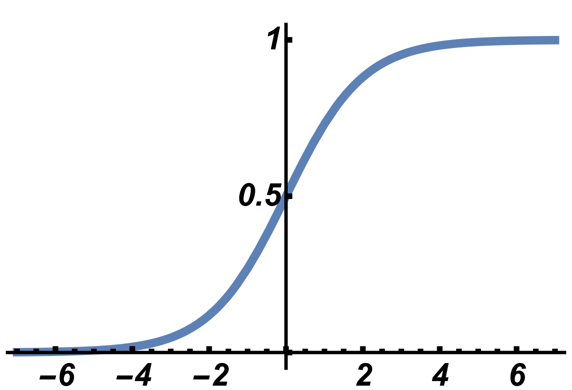
  - Computations are independent $\Rightarrow$ parallelizable

(A) Using NNs to implement algorithms

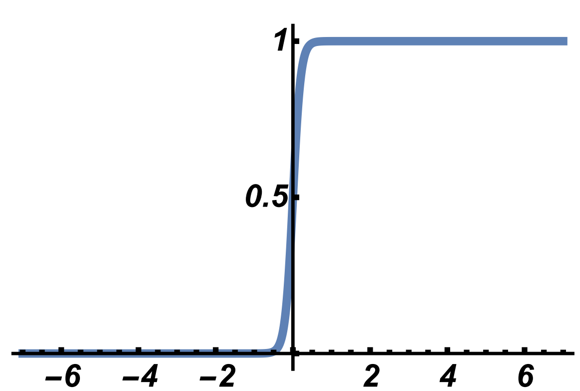
- This is more about abusing the modular nature of NN
- Each layer performs an action / applies a function
- Implement an arbitrary algorithm by
    - choosing the function appropriately
    - including a (possibly trained) NN that performs a specific algorithm (e.g. computes binomial coefficients)
    - emulating a computer using NNs (combine NN layers that perform bit-wise and/xor/not/… operations
- Like playing LEGO

(B) Using NN to approximate functions

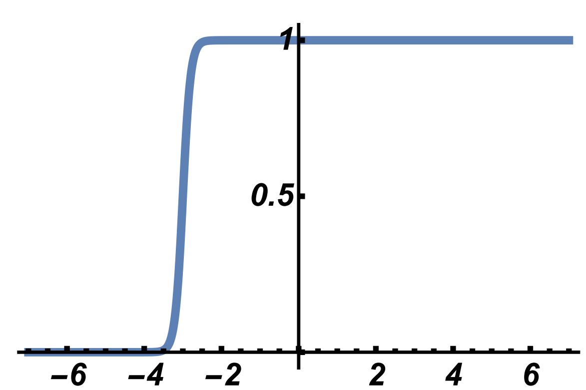
- Simple case: 1 layer, 1 node, logistic sigma function
    - Linear Layer: $x_{\text{int}} = a x_{\text{in}} + b$
    - Activation Function: $x_{\text{out}} = 1/(1 + \exp[x_{\text{int}}])$
      $= 1/(1 + \exp[a x_{\text{in}} + b])$
    - a : Steepness of step (step function for $a \to \infty$)
    - b : Position of step: (intersects y-axis at $y = 1/2$ for $b = 0$)



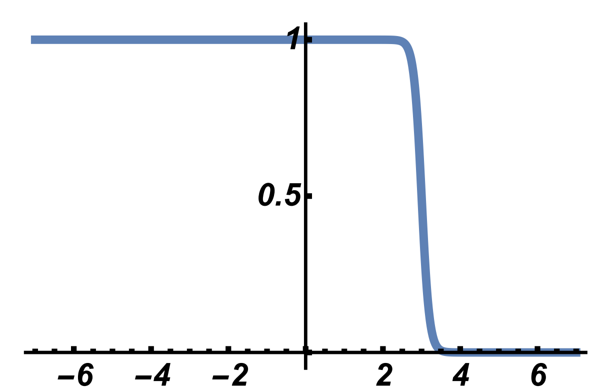
$a = 1, b = 0$



$a = 10, b = 0$

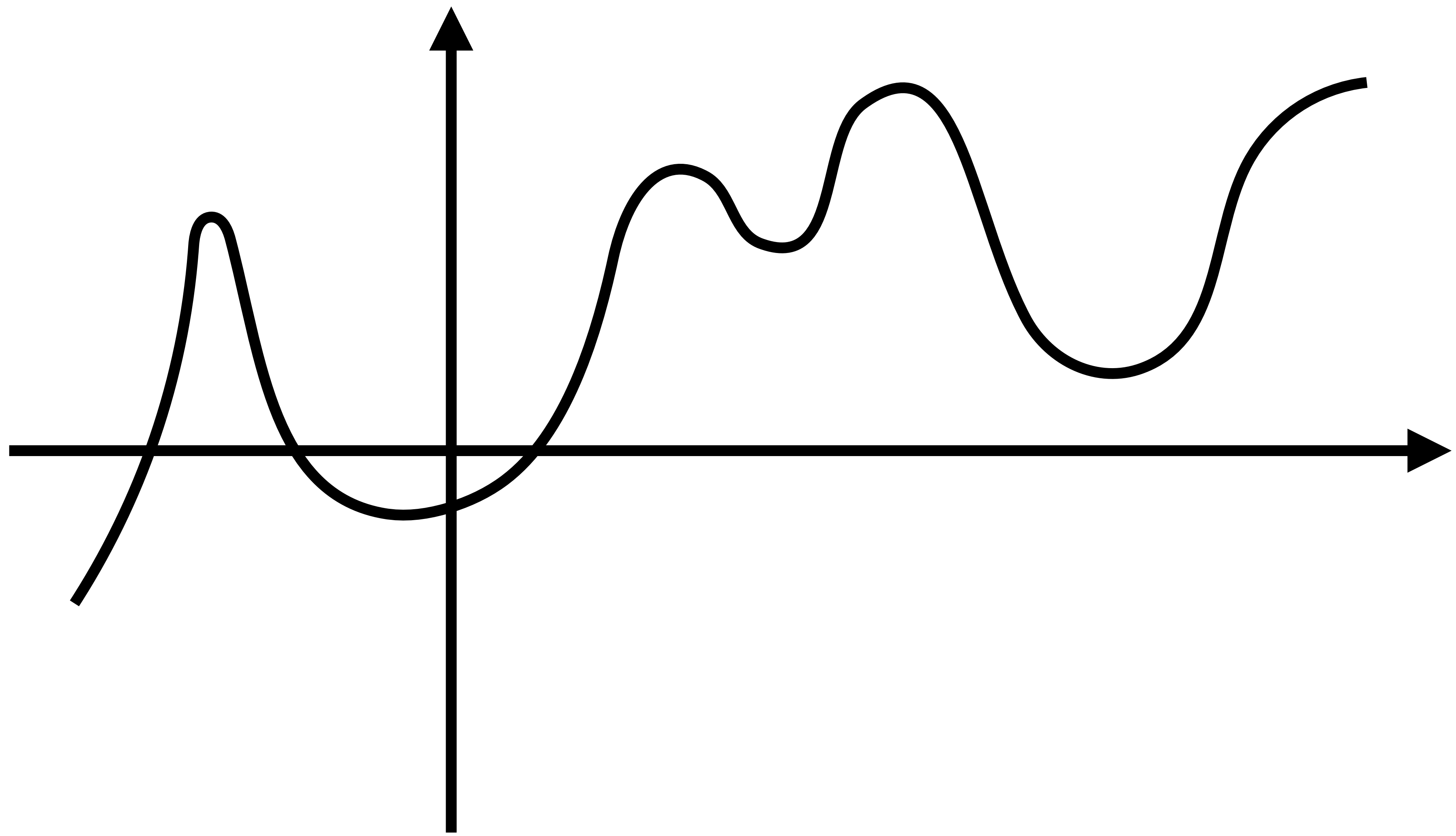


$a = 10, b = -30$

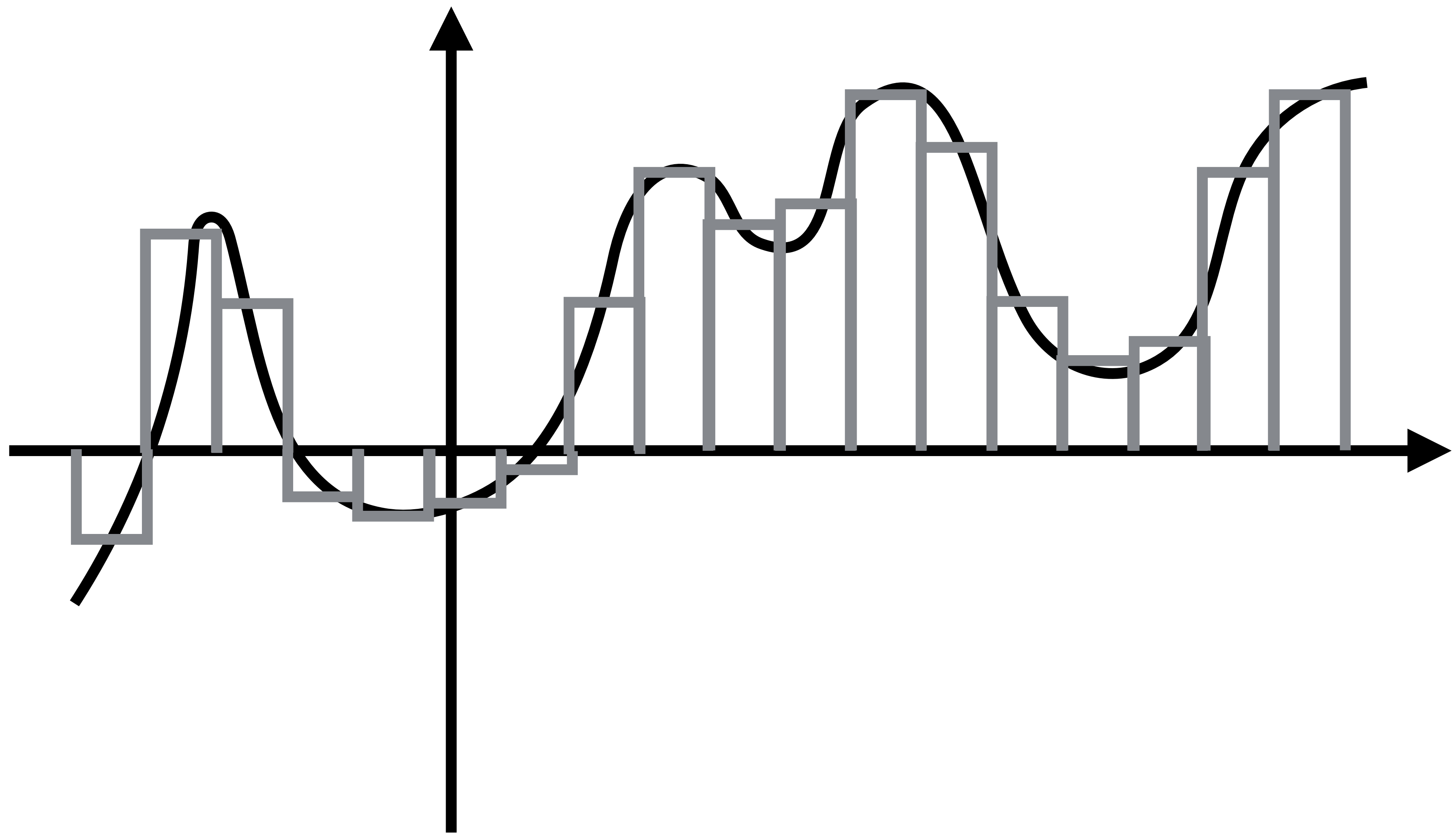


$a = -10, b = 30$

(B) Using NN to approximate functions



(B) Using NN to approximate functions

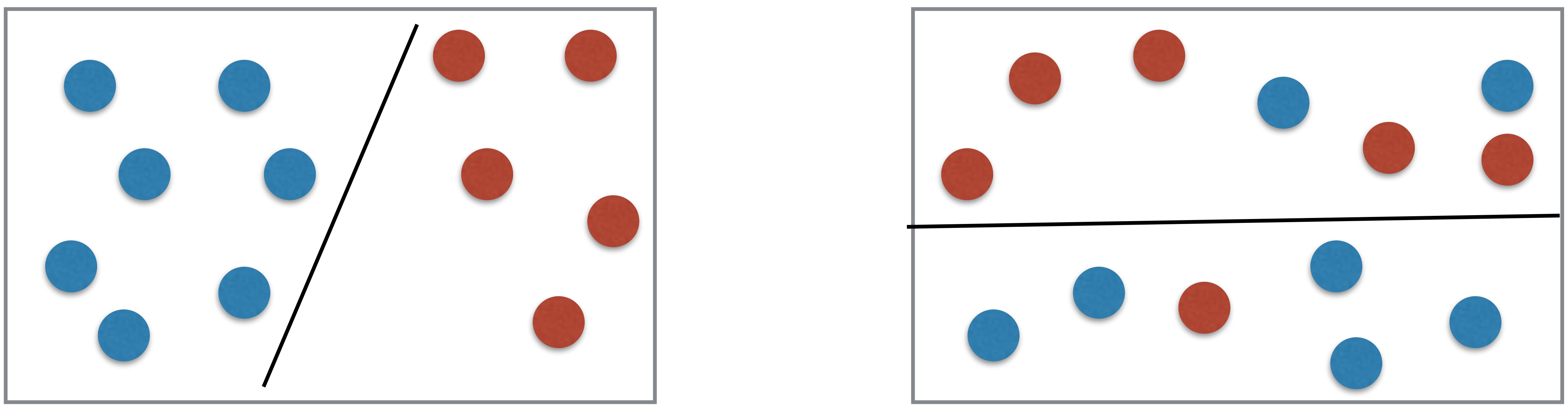


- More nodes $\Rightarrow$ more steps $\Rightarrow$ approximate any function (with one layer) “Universal Approximation Theorem”

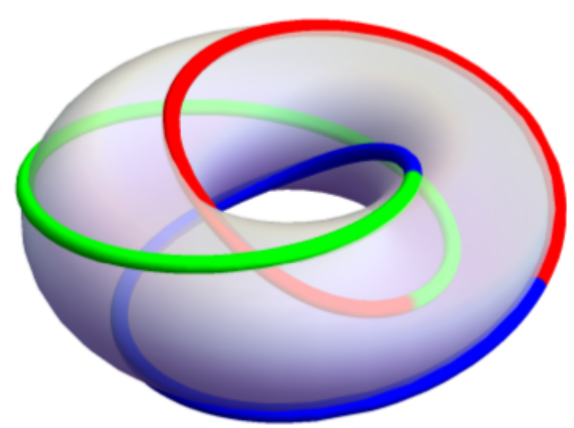
[Cybenko ’89; Hornik ’91; Nielsen‘15]

(C) Using NN to classify data

- Simple (feed-forward) NNs can classify data that is linearly separable, i.e. their convex hulls are disjoint



- When is data (linearly) separable? E.g. is the (3,2) torus knot linearly separable?

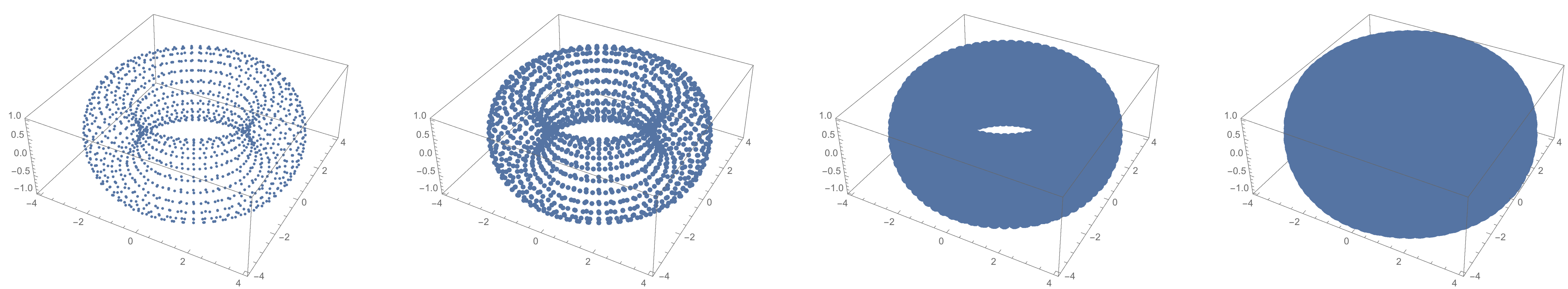


(C) Using NN to classify data

- Several ways to make data "linearly" separable
  - Go to higher dimensions (an n-dimensional knot can be disentangled in $2n+2$ dimensions)
  - Change / warp the geometry by applying non-linear functions (away from Euclidean, a "straight" line looks different)
  - Deform the data to make the error (i.e. the line that cuts through the entangled data) as small as possible

(C) Using NN to classify data

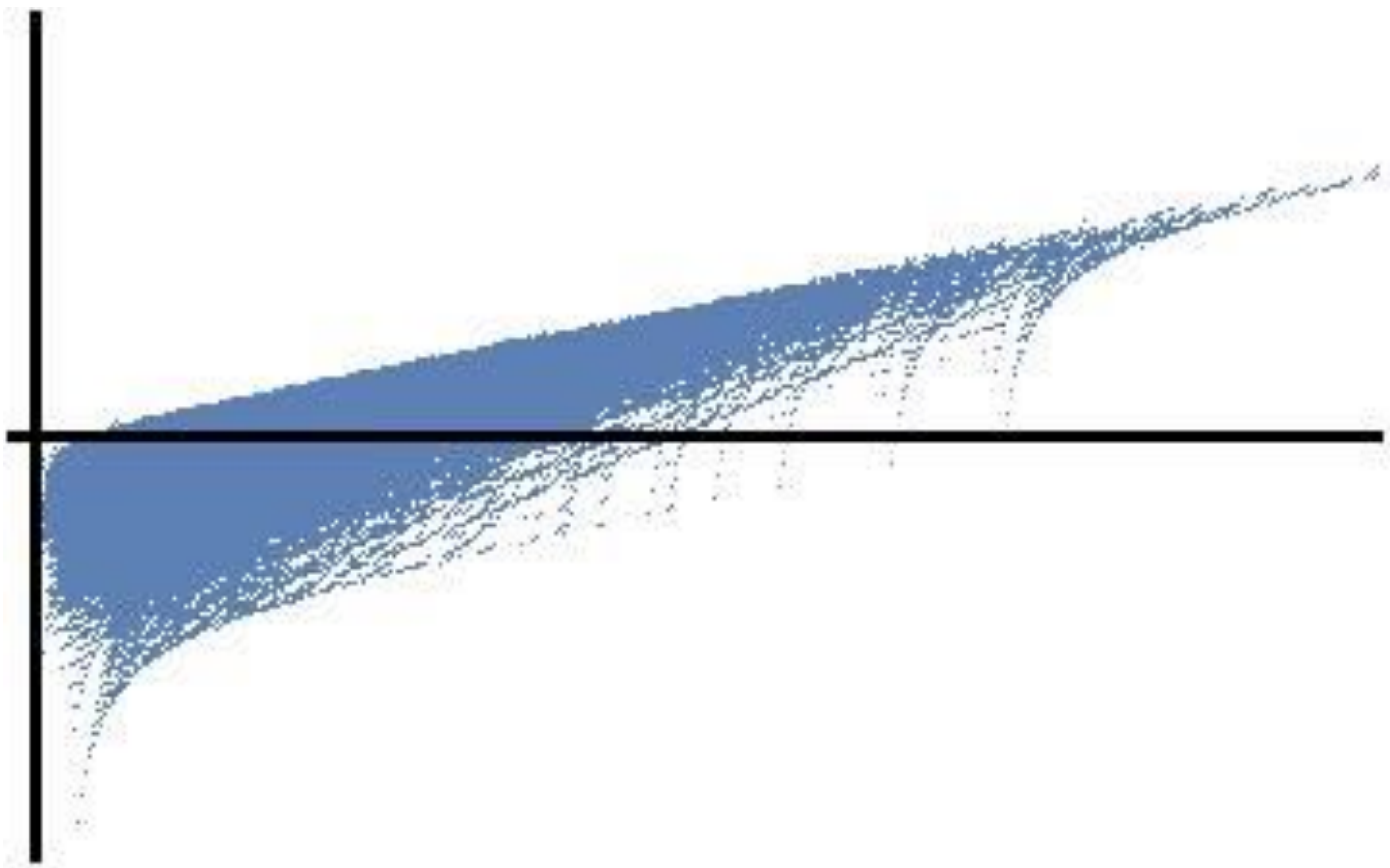
- Ways to identify "topology" of point set: Persistent homology
  - Has been applied to string vacua in [Cirafici '15]
  - Idea:
    - Replace data points by balls (several disconnected components)
    - As radius of points grow, components connect / form cycles / …
    - When radius grows further, cycles can disappear again



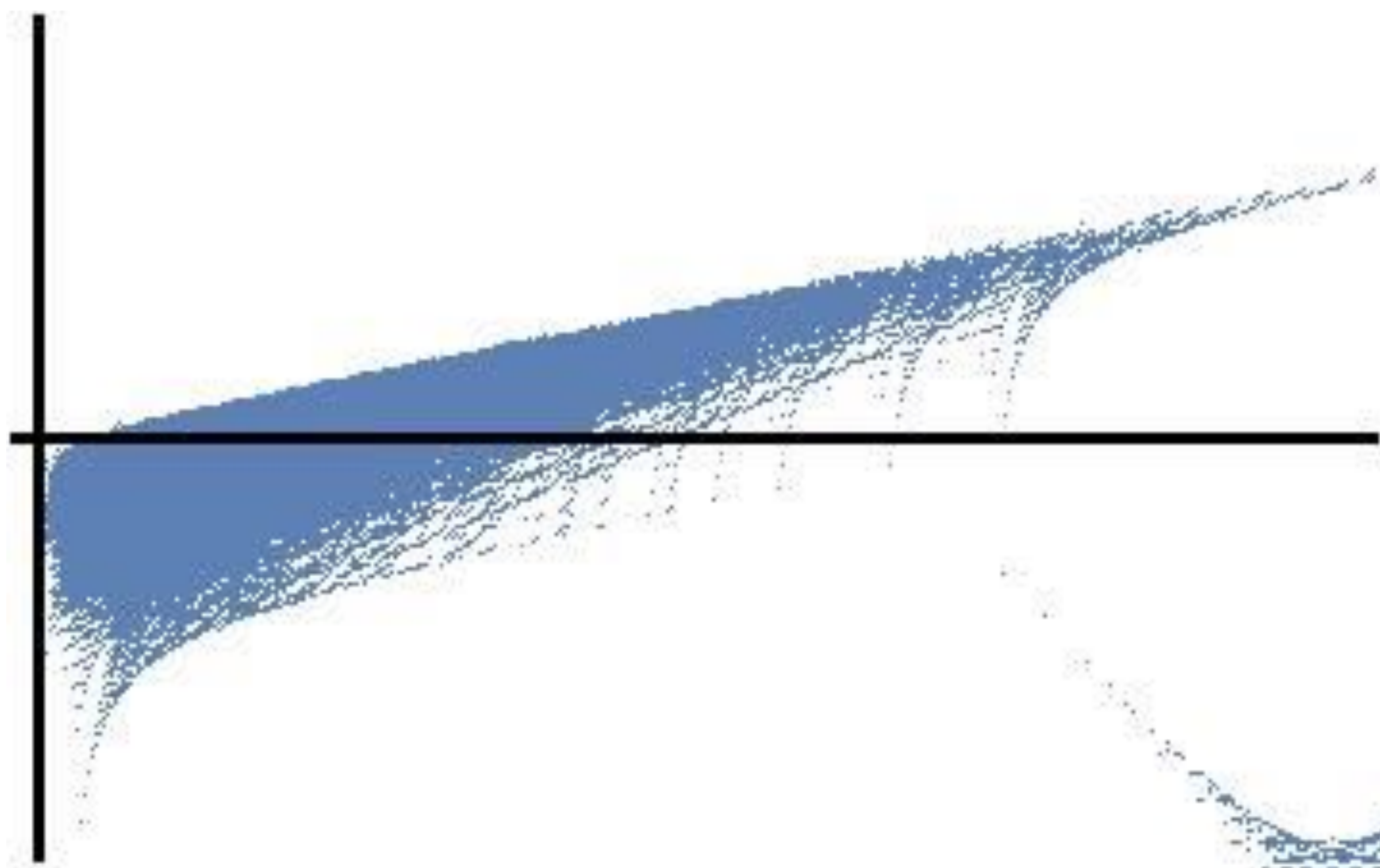
(C) Using NN to classify data

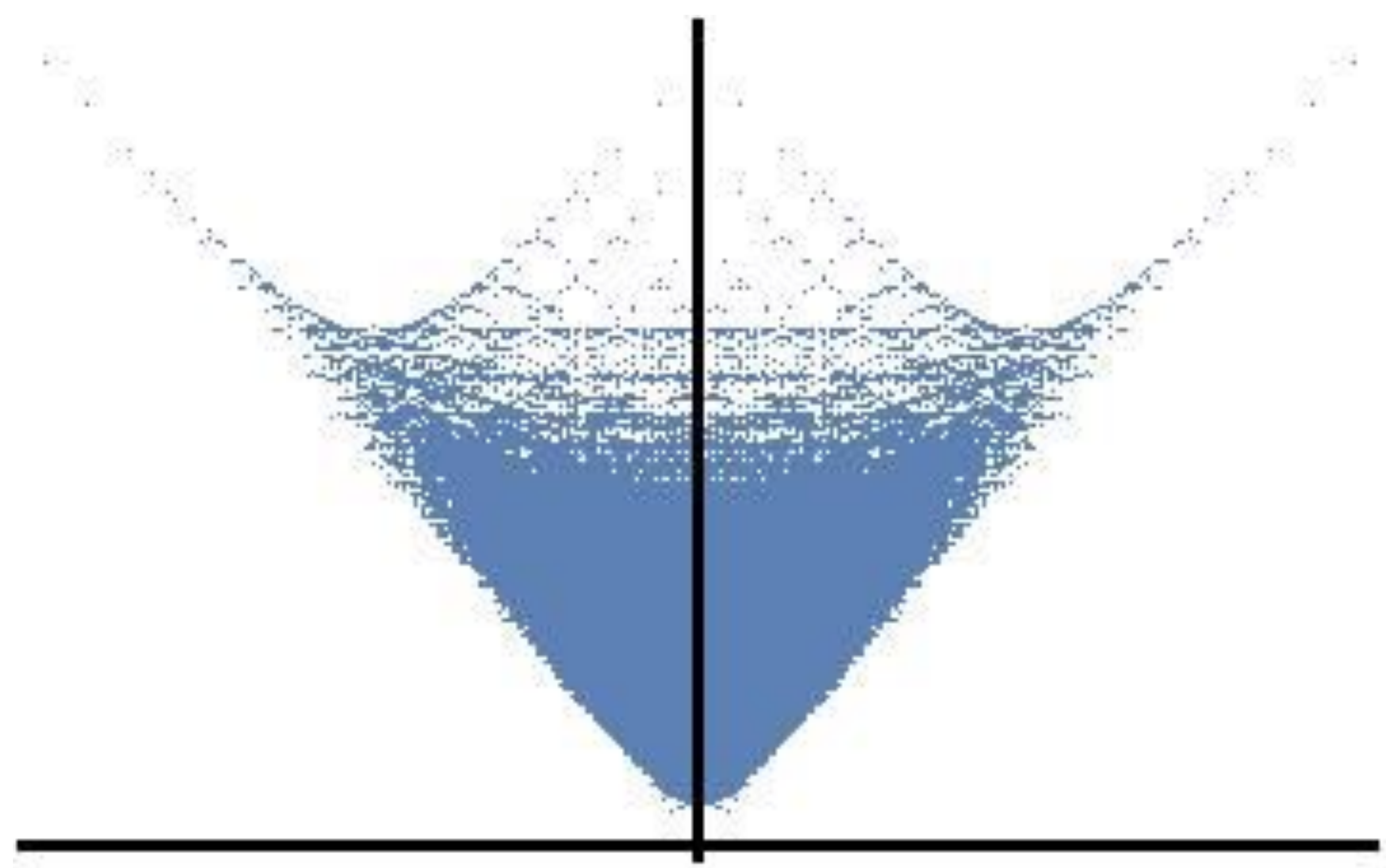
- For each k-cycle determine how long it exists as a function of the sphere radius $\Rightarrow$ barcode (Betti number vs radius)
- The longer a cycle exists the more likely it is to be a true feature
- In this talk we want to follow a different approach
  - The bar codes you obtain depend on the way you plot the data

(C) Using NN to classify data



(C) Using NN to classify data



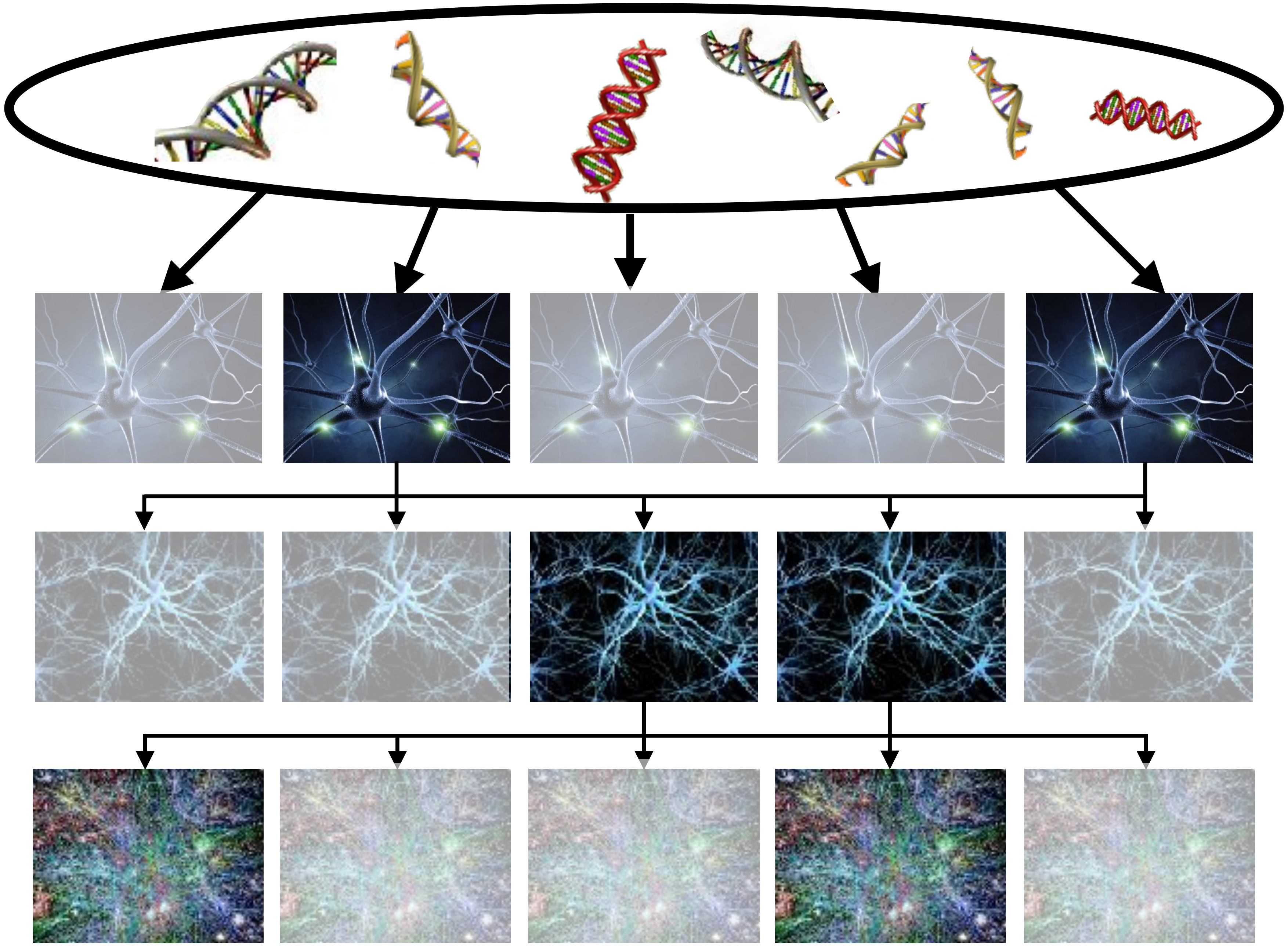


(C) Using NN to classify data

- For each k-cycle determine how long it exists as a function of the sphere radius $\Rightarrow$ barcode (Betti number vs radius)
- The longer a cycle exists the more likely it is to be a true feature
- In this talk we want to follow a different approach
    - The bar codes you obtain depend on the way you plot the data
    - For some applications we are only interested in a NN that works best

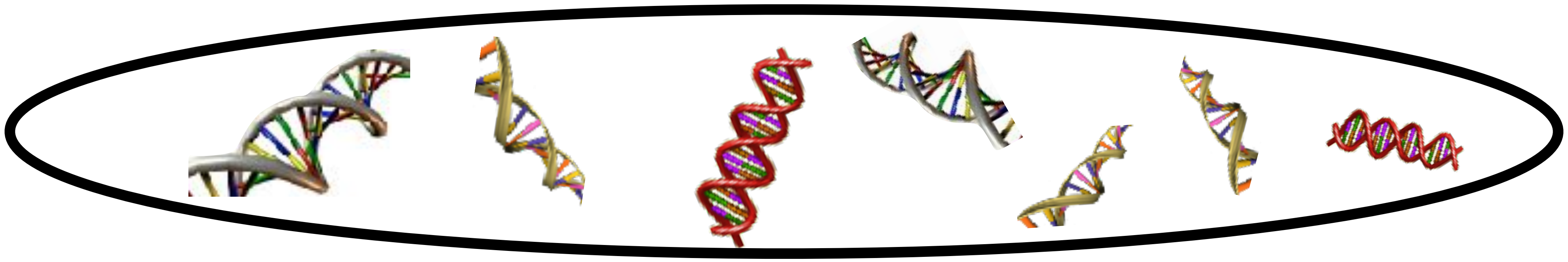
(C) Using NN to classify data

- For each k-cycle determine how long it exists as a function of the sphere radius $\Rightarrow$ barcode (Betti number vs radius)
- The longer a cycle exists the more likely it is to be a true feature
- In this talk we want to follow a different approach
    - The bar codes you obtain depend on the way you plot the data
    - For some applications we are only interested in a NN that works best
- Instead of analyzing the data to decide the necessary complexity of the NN: Simply evolve a NN that works best

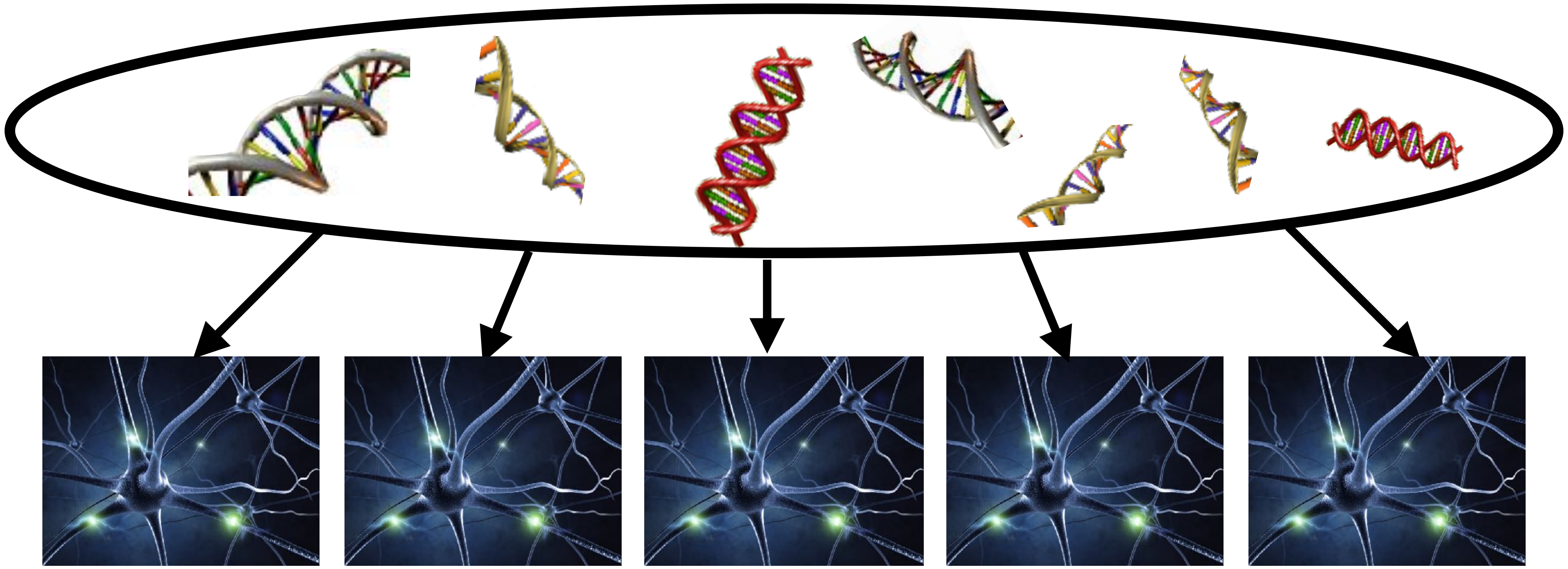


Introduction to Genetic Algorithms

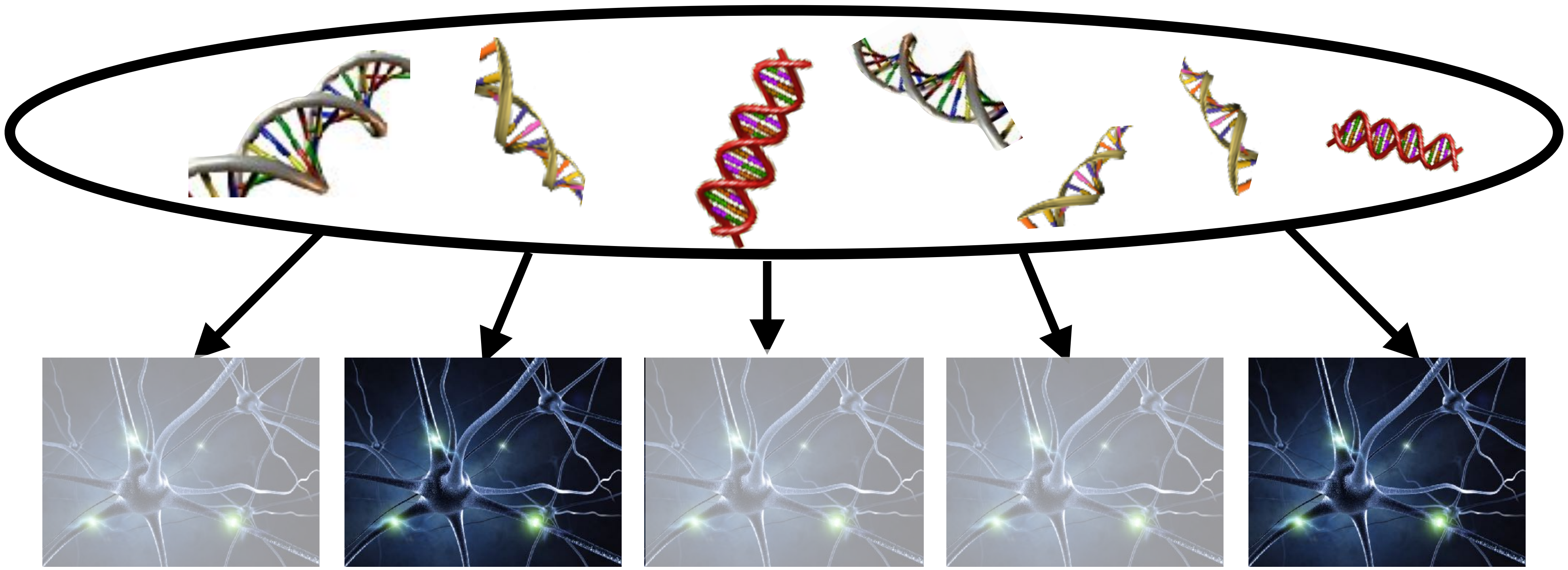
Genetic Algorithms 101



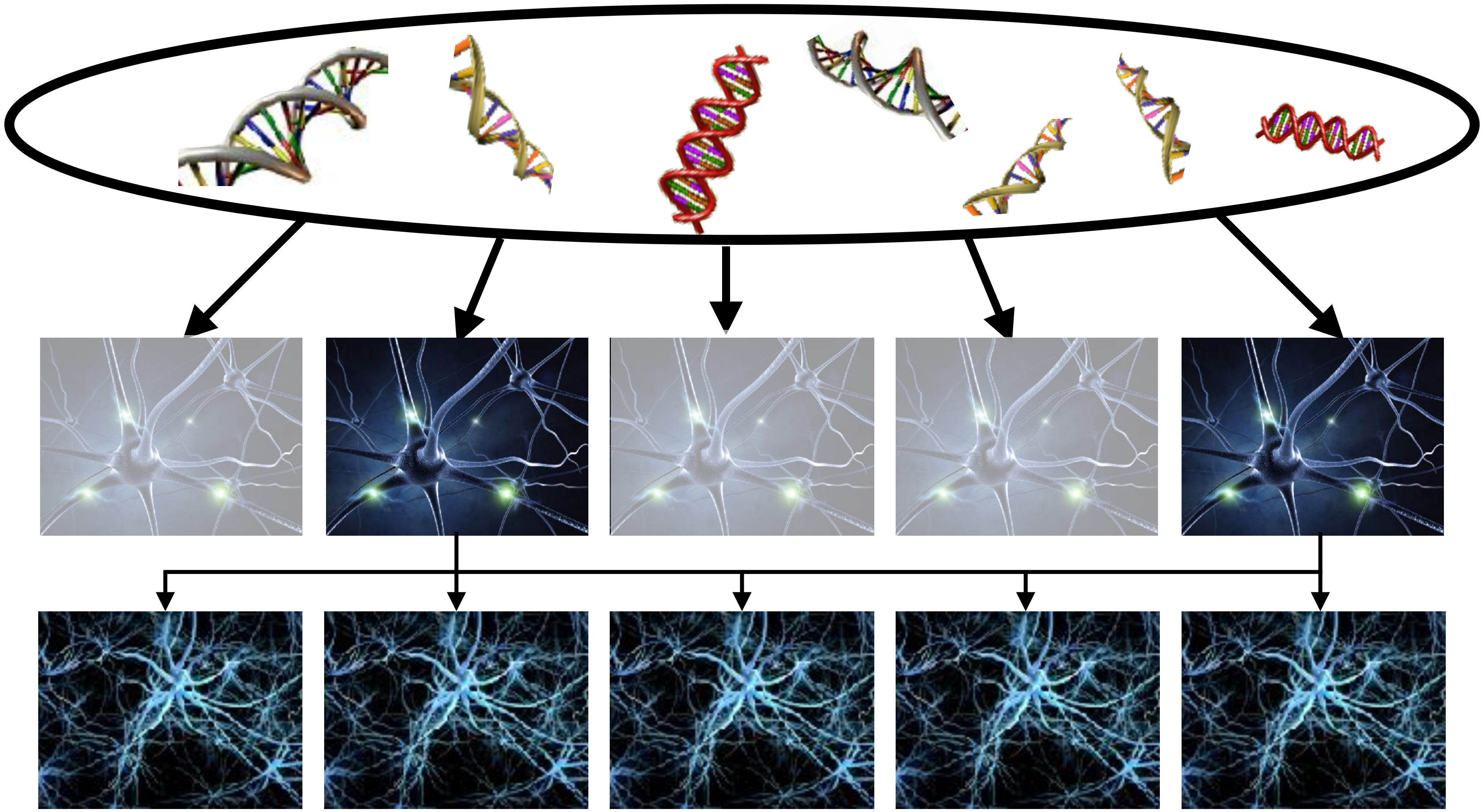
Genetic Algorithms 101



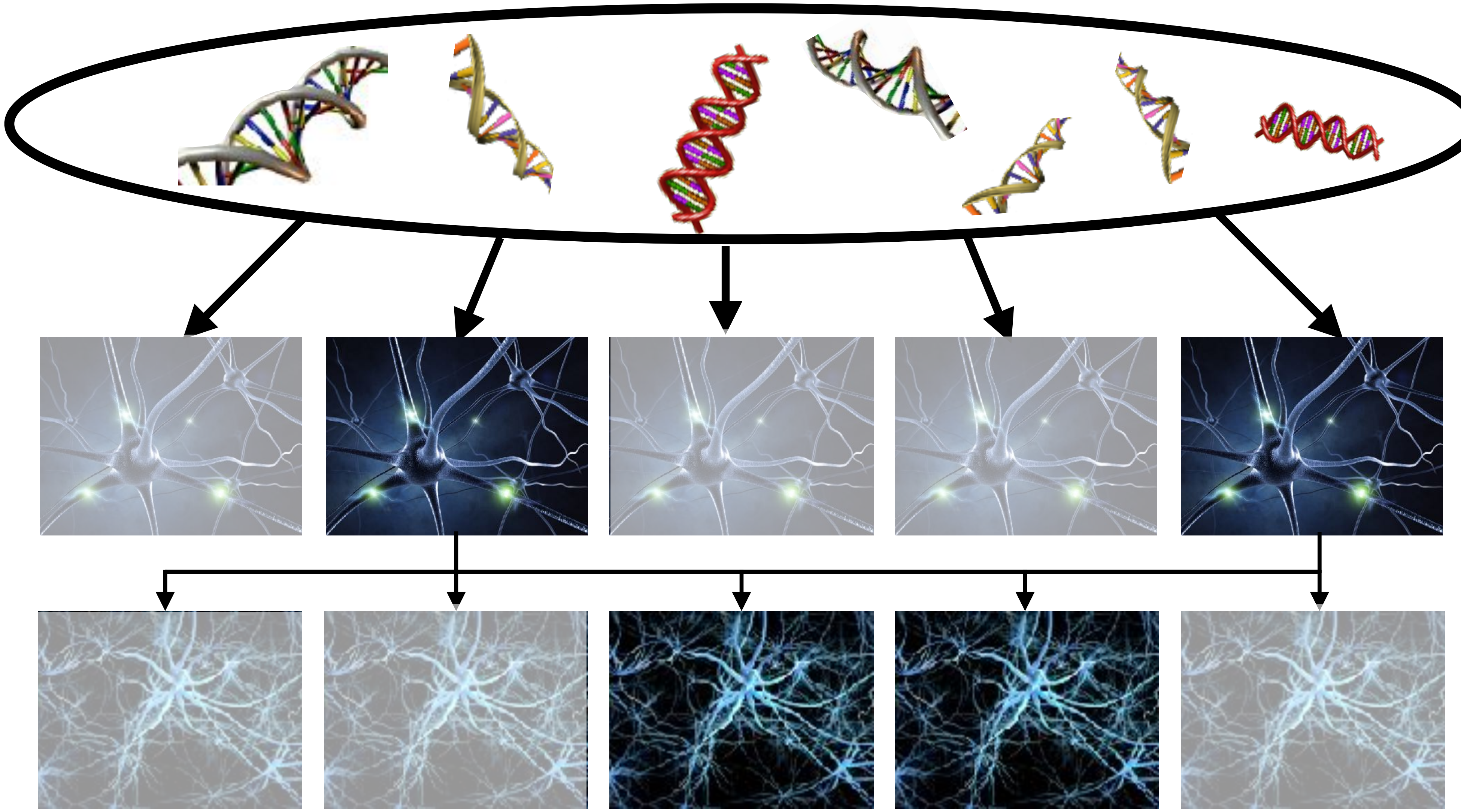
Genetic Algorithms 101



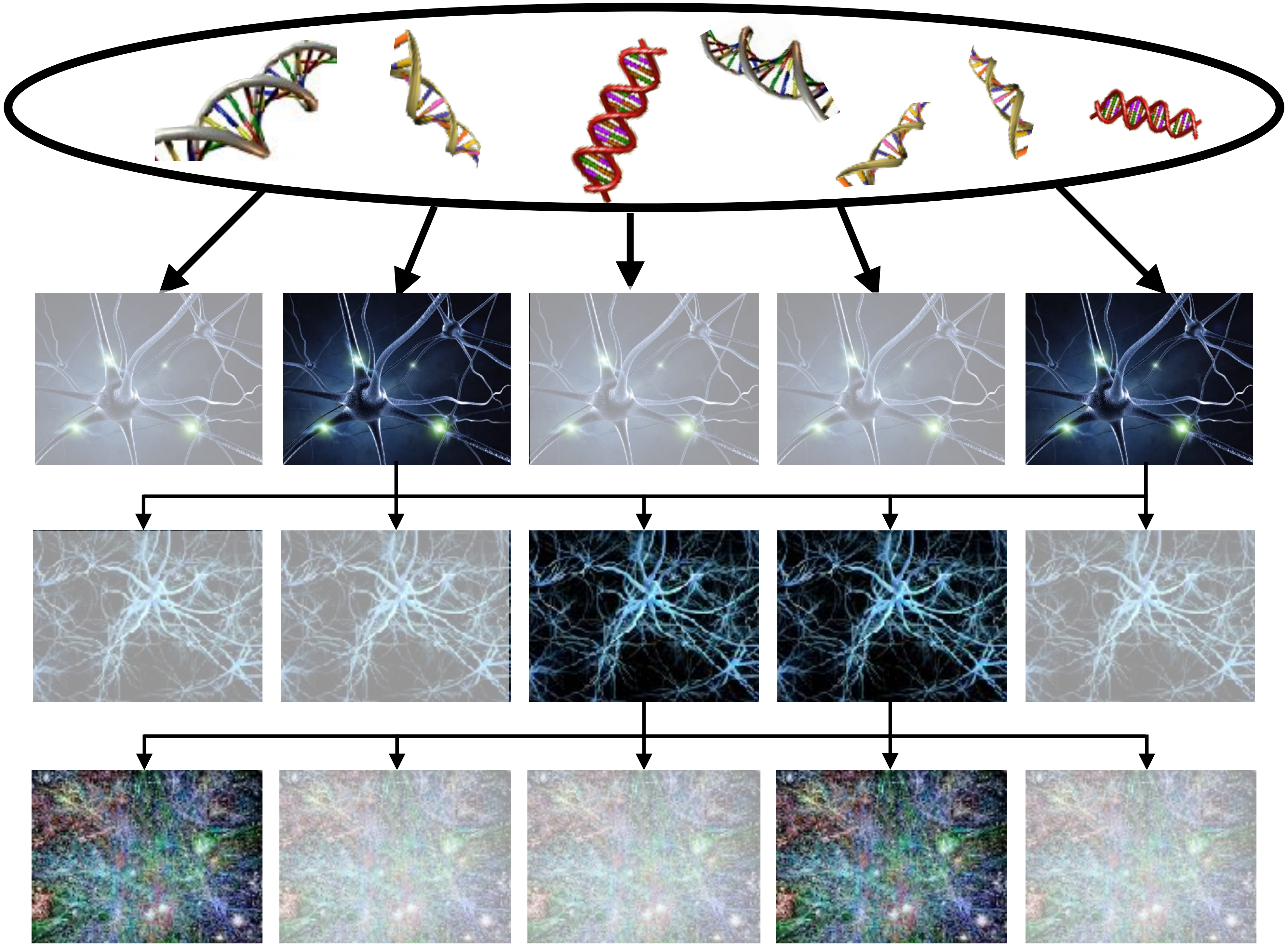
Genetic Algorithms 101



Genetic Algorithms 101



Genetic Algorithms 101

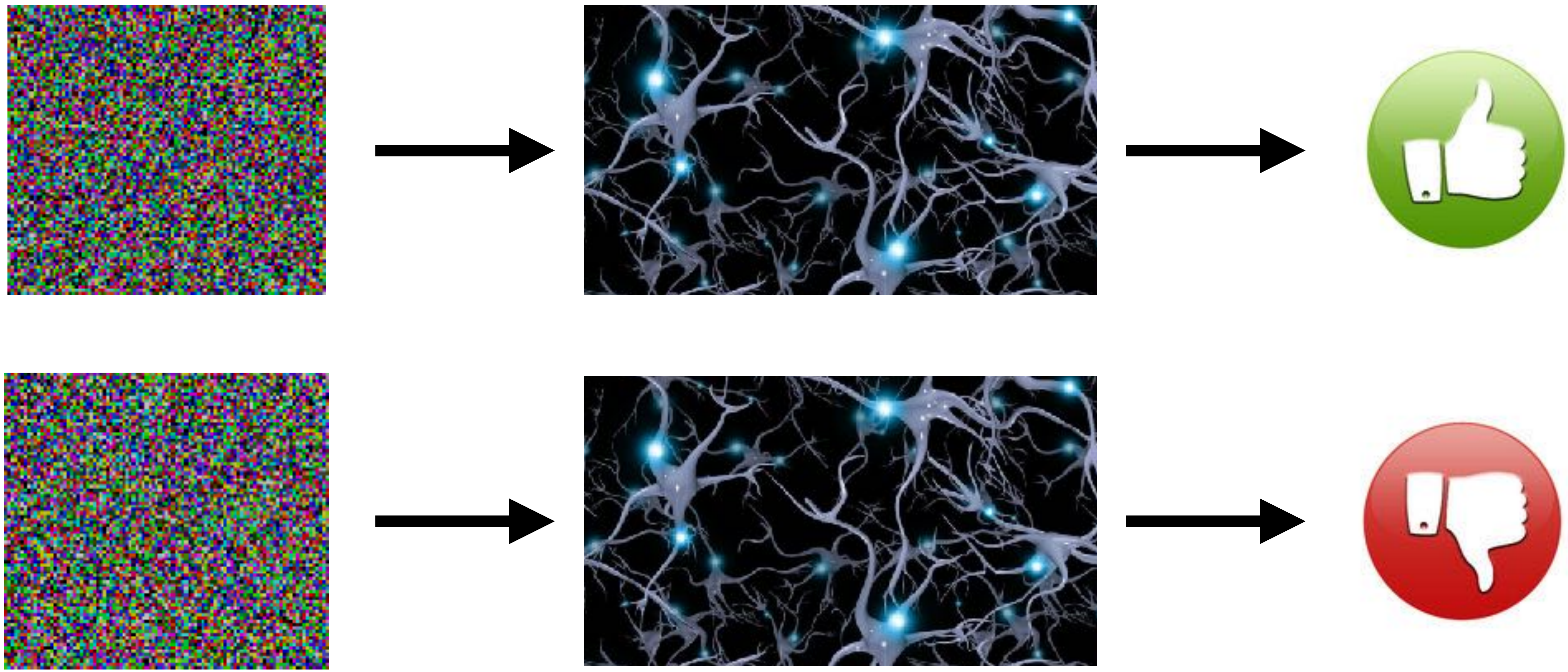


Genetic Algorithms 101

- Idea: Copy nature again $\Rightarrow$ dynamically evolve models [Darwin 1859]
- Applied in string theory to find models [Allanach,Grellscheid,Quevedo'04; Abel,Rizos'14]
- Pros:
  - Evolve / improve themselves 24/7 (automated trial & error)
  - Evolution/fitness evaluation parallelizable within a generation
- Possible applications:
  - Evolve connections rather than weighting them by training - similar to evolution of nerve connections between synapses in the human brain
  - Evolve training/validation set (important if the set cannot be easily randomized: the train set might accidentally have a feature which is picked up by the NN)
  - Evolve entire NNs (topology, activation function, no of layers, no of nodes per layer,…) - similar to evolving entire species in a computer

Genetic Algorithms - Modifications

- Adjust *fitness*
  - accuracy of prediction
  - computation time
- Change *reproduction*
  - cell division or cloning
  - *n* fittest get to reproduce via mating
  - all get to reproduce weighted by their fitness
  - mixture of cell division and mating depending on complexity of evolved species
- Change *mutation*
  - change rate
  - adjust complexity of genes that can mutate
  - change gene properties instead of exchanging entire genes
- Change *complexity* of genes in the *gene pool*
  - include higher level NNs
  - include trained NNs

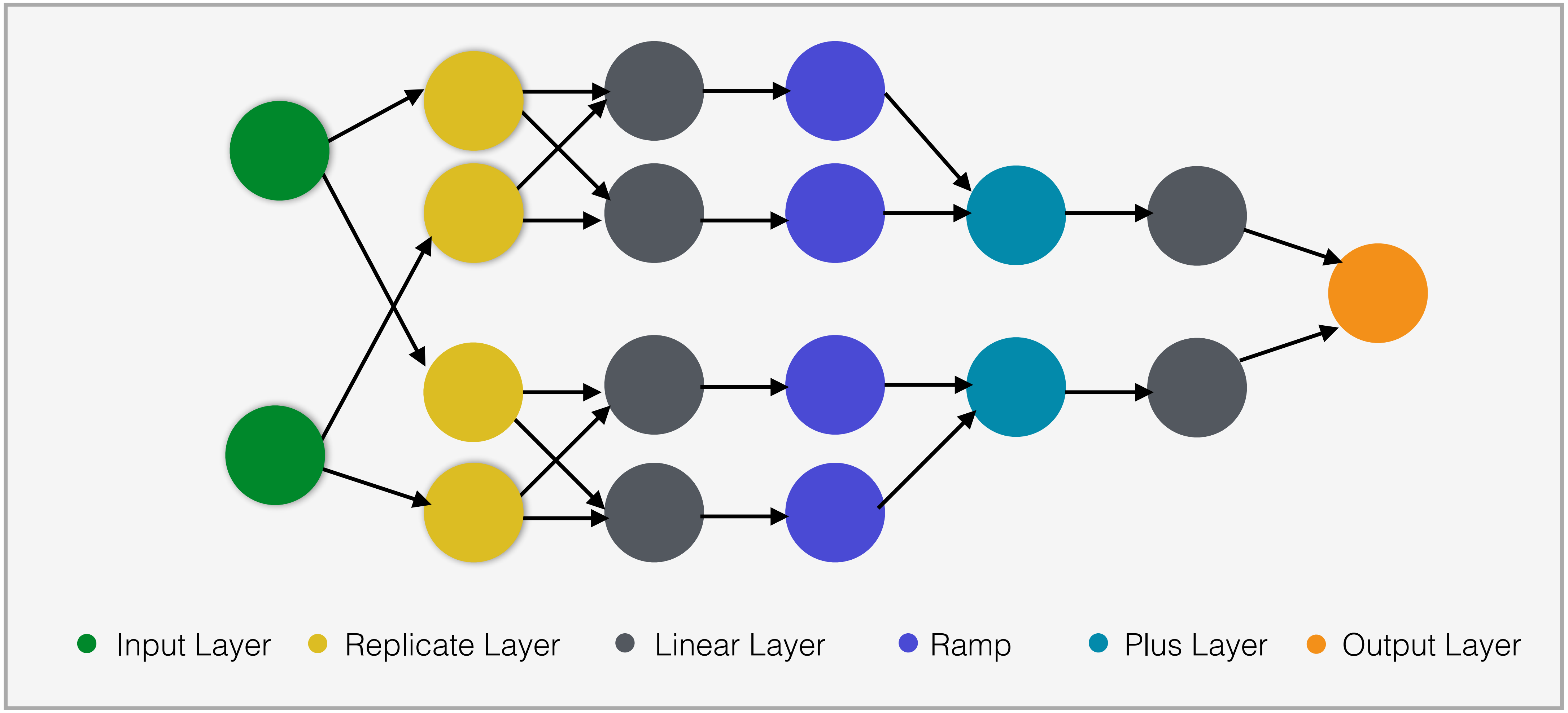


Combining both approaches

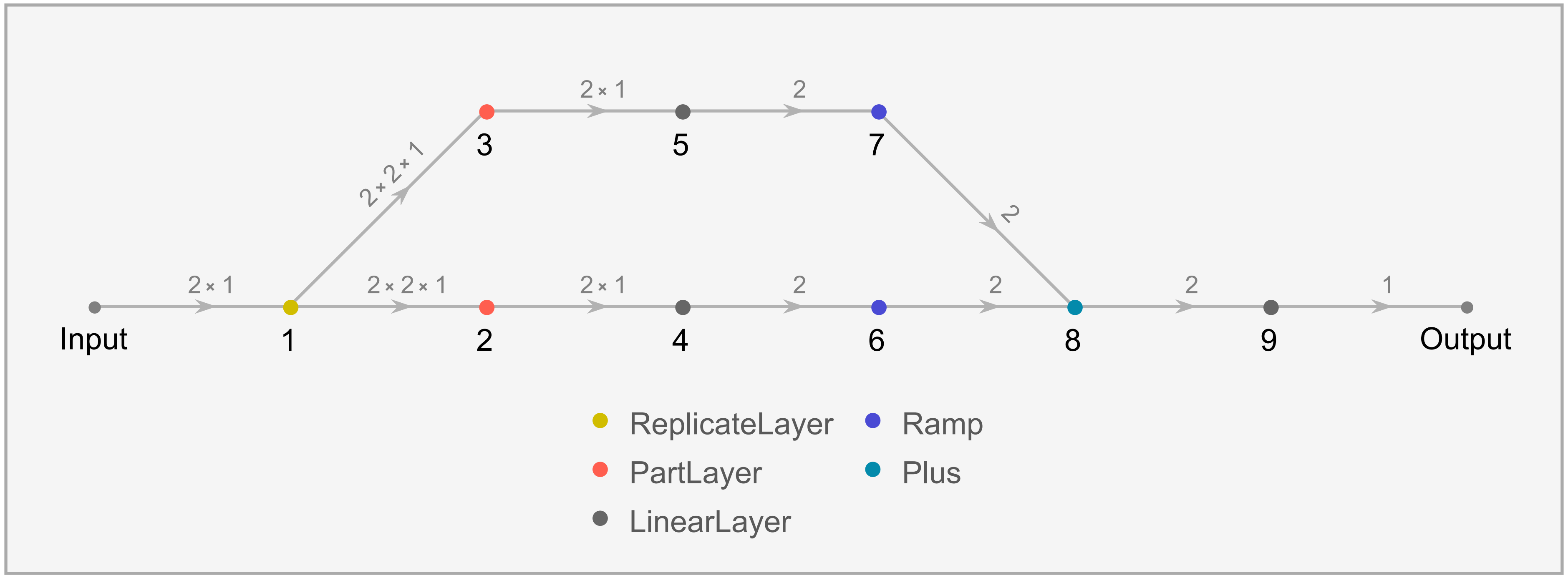
Example: Classify stable bundles

- Line bundle D-flat if $\int_X c_1(\mathcal{L}) \wedge J \wedge J = \kappa_{ijk} k^i t^j t^k = 0$
- Stability restricts Kahler cone to sub-region
- Still bounded by hyperplanes $\Rightarrow$ well-suited for NNs
- Simple example: CICY on $\mathbb{P}^2 \times \mathbb{P}^2$:
  - $h^{1,1} = 2$ (from the two $\mathbb{P}^2$ factors)
  - Kahler cone: $t^1, t^2 > 0$
  - Line bundle: $c_1(\mathcal{L}) = \mathcal{O}_X(k_1, k_2)$
  - Intersection numbers: $\kappa_{112} = \kappa_{122} = 3$, $\kappa_{111} = \kappa_{222} = 0$
  - Stable iff $k_1 > 0, k_2 < 0$ or $k_1 < 0, k_2 > 0$

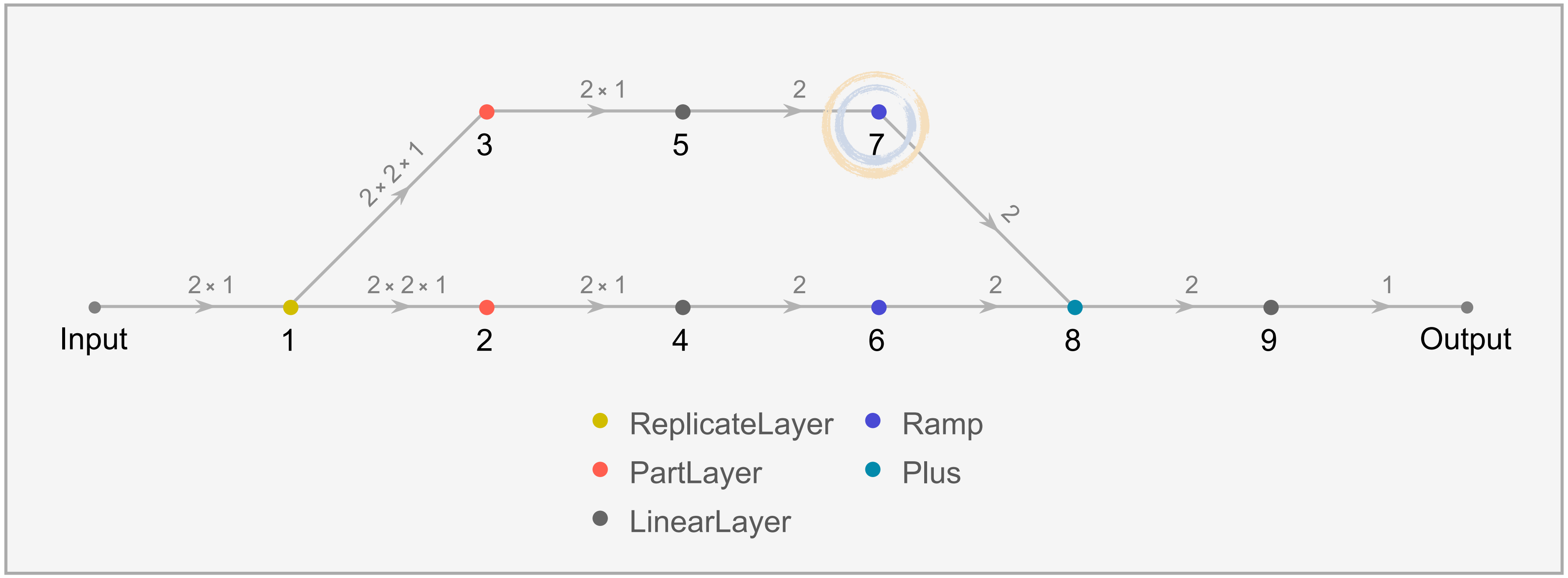
Example: Classify stable bundles

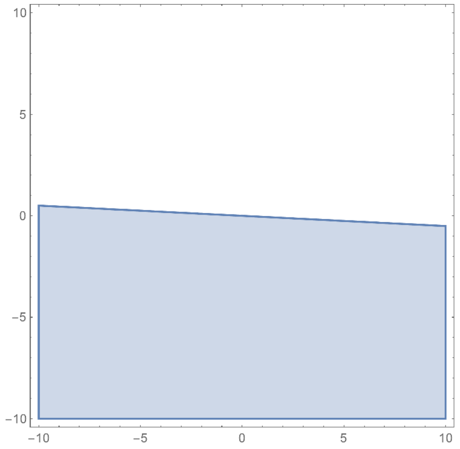


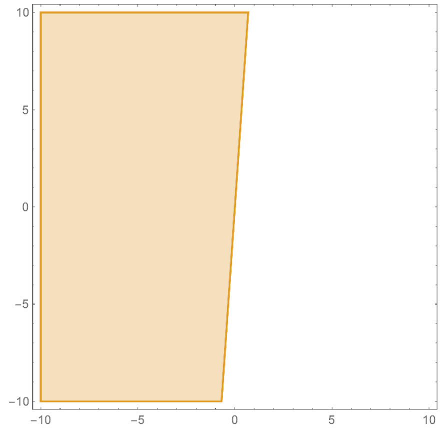
Example: Classify stable bundles



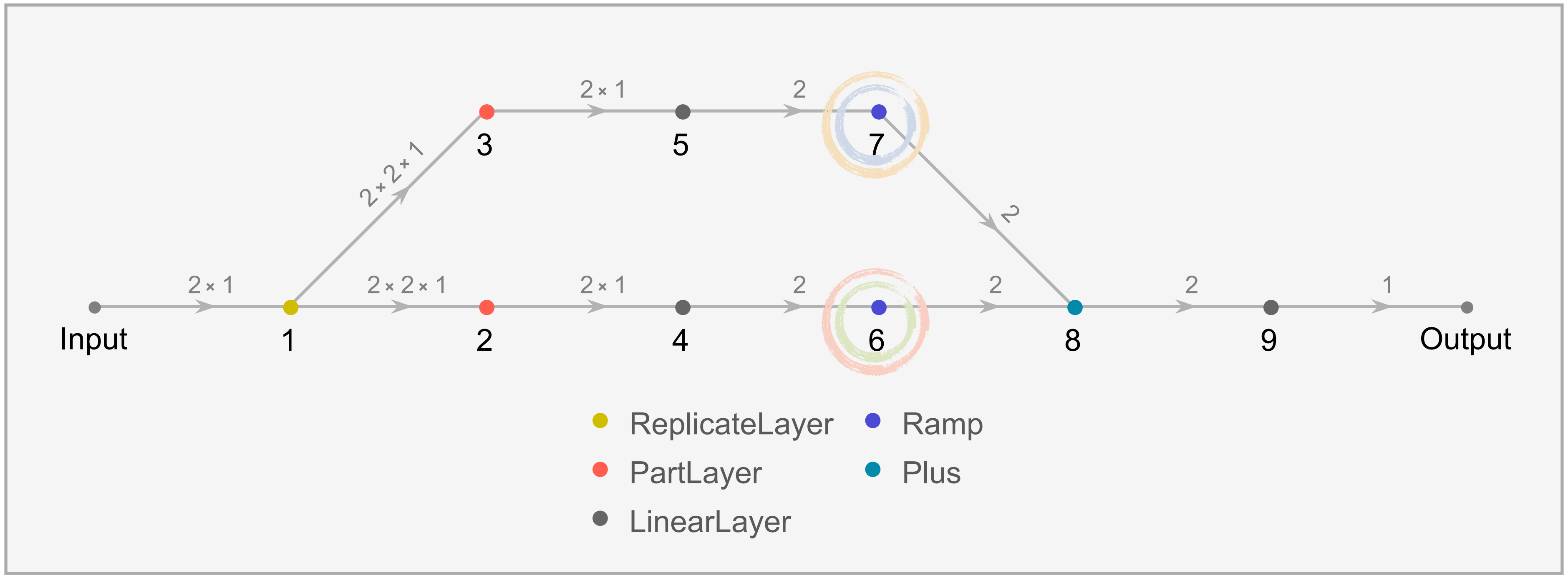
Example: Classify stable bundles

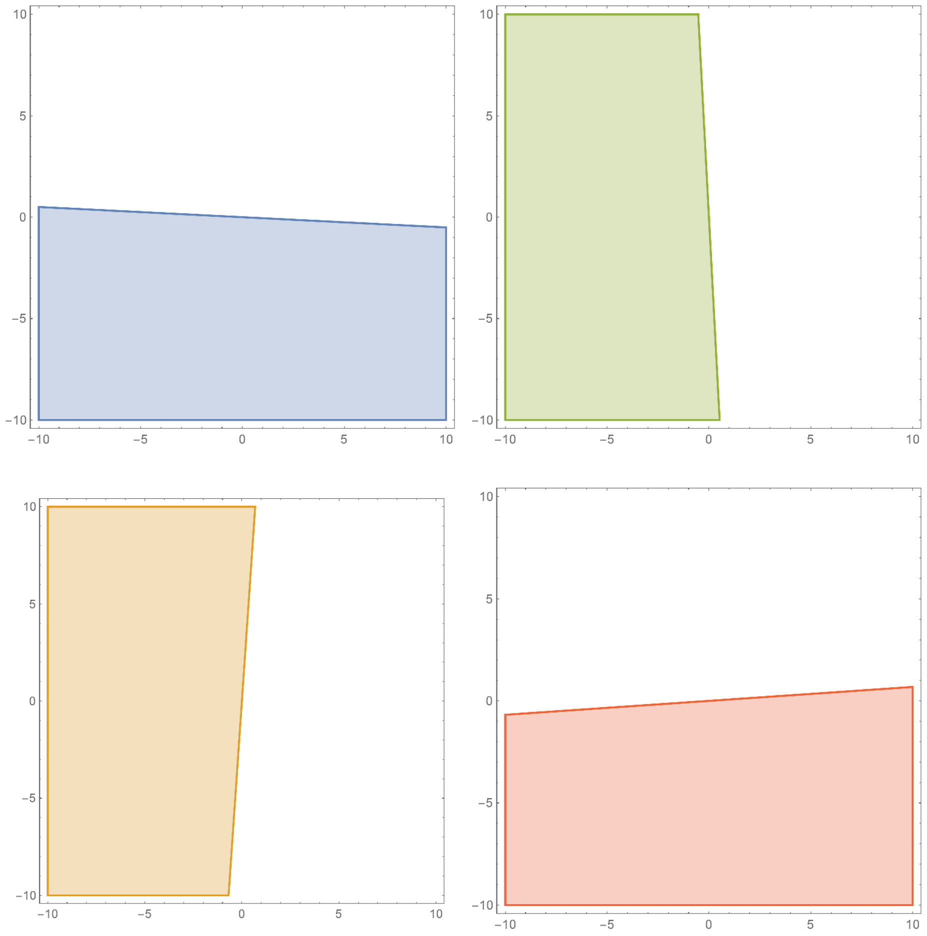




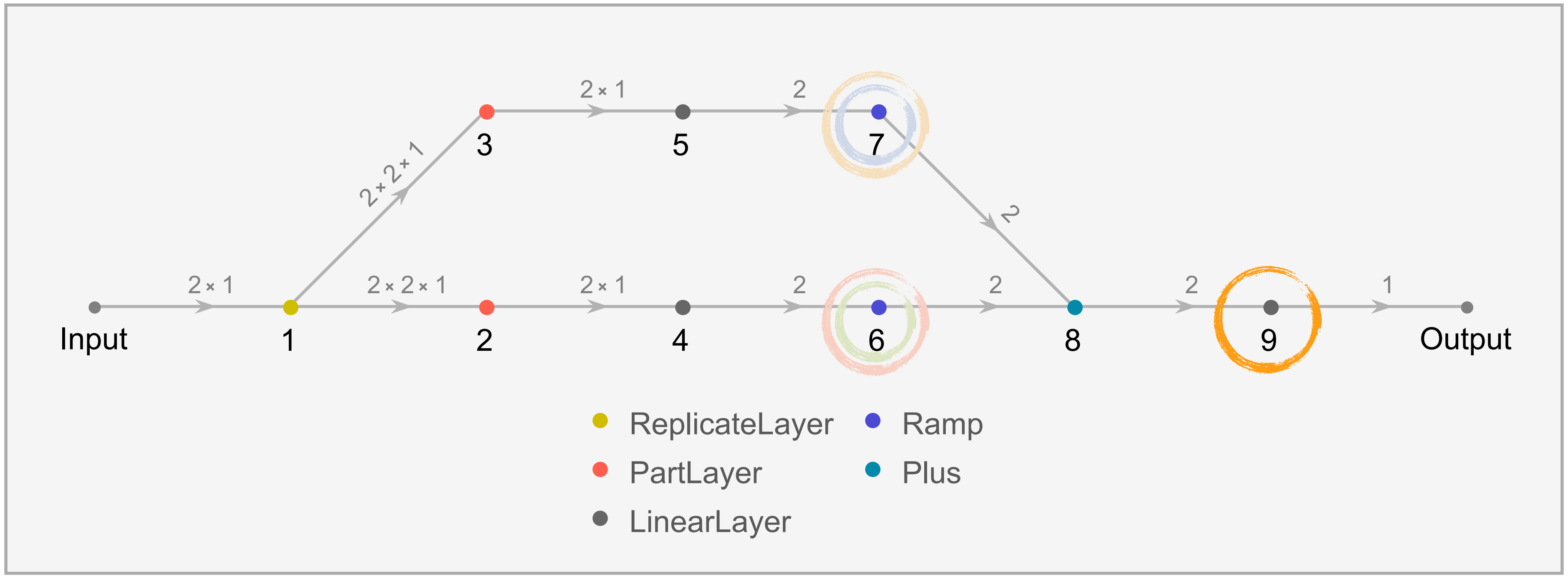


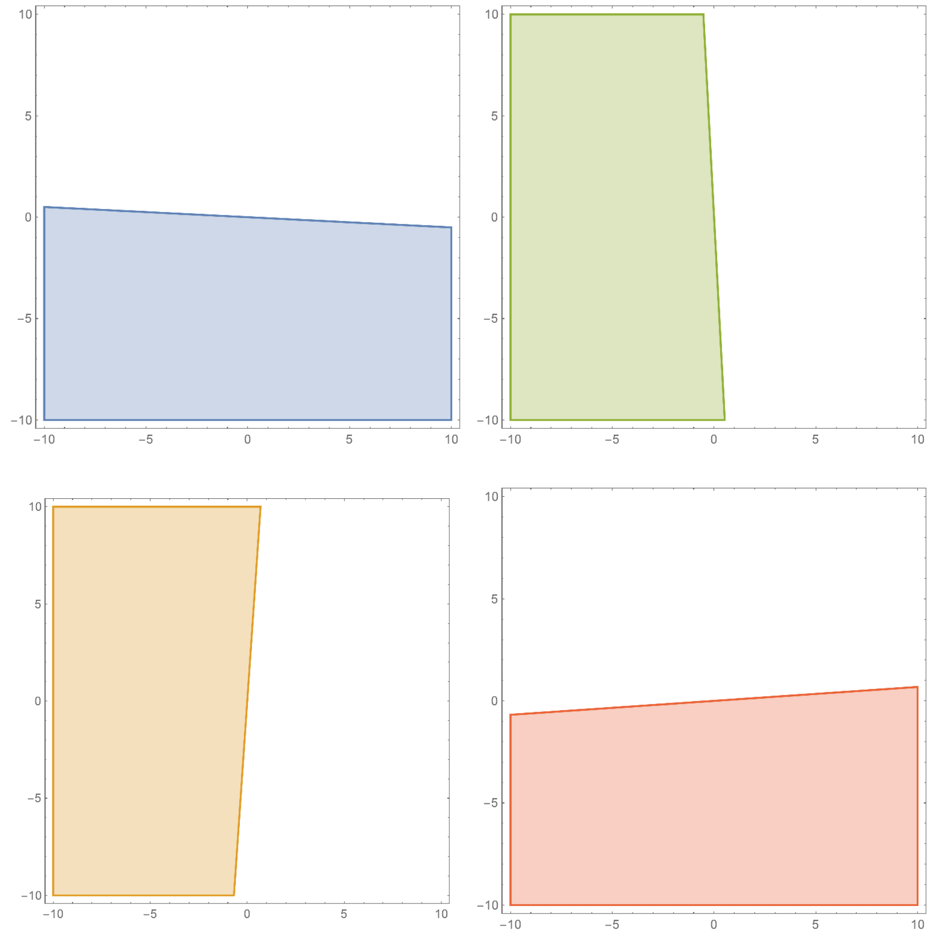
Example: Classify stable bundles

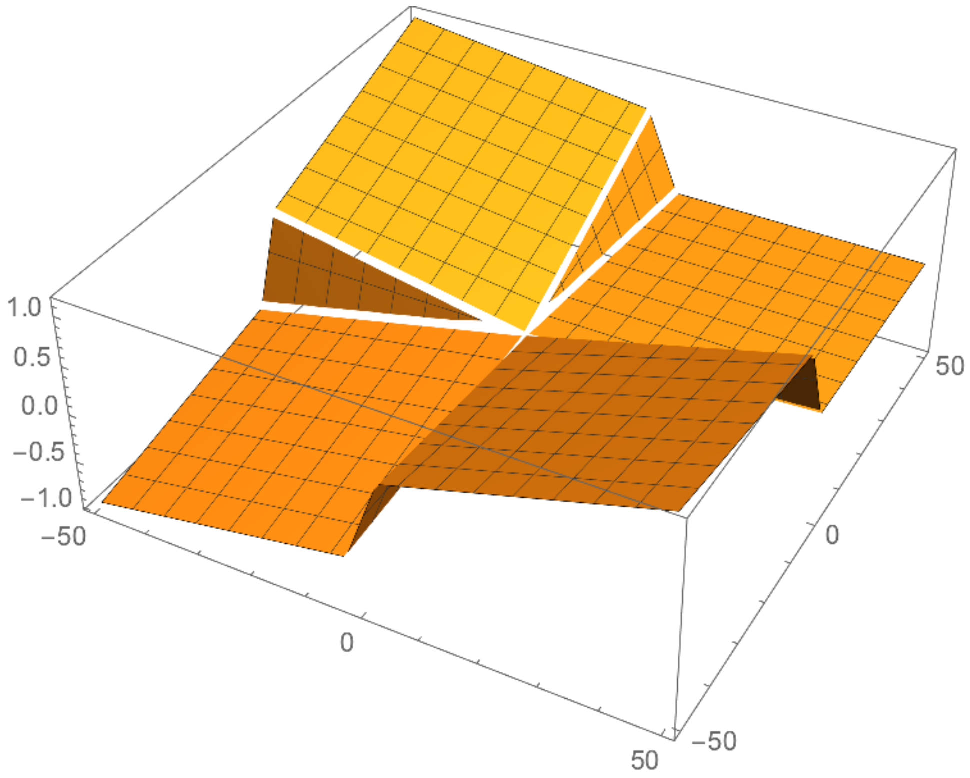




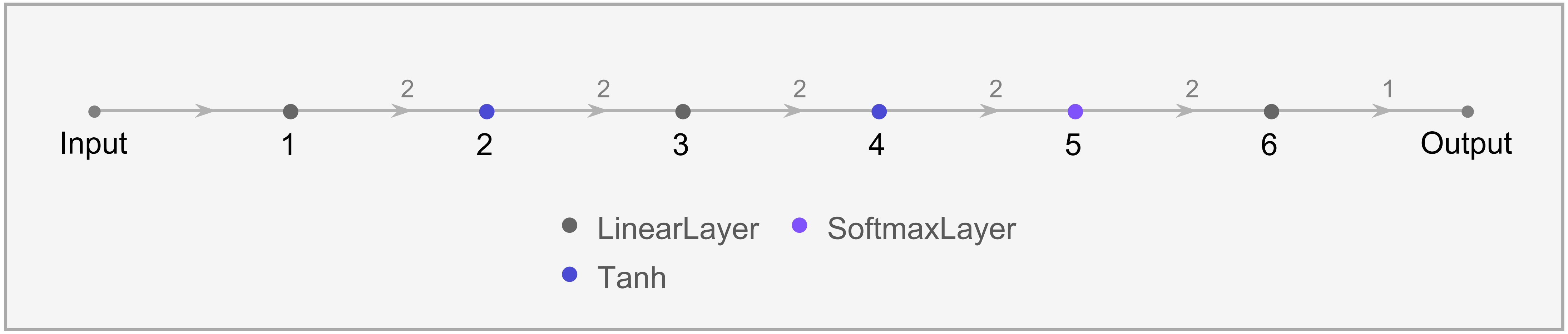
Example: Classify stable bundles



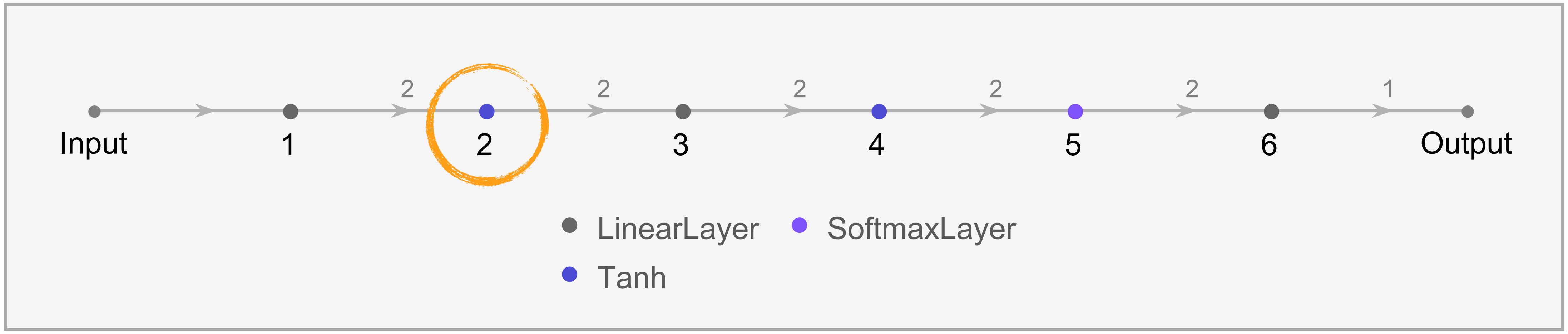


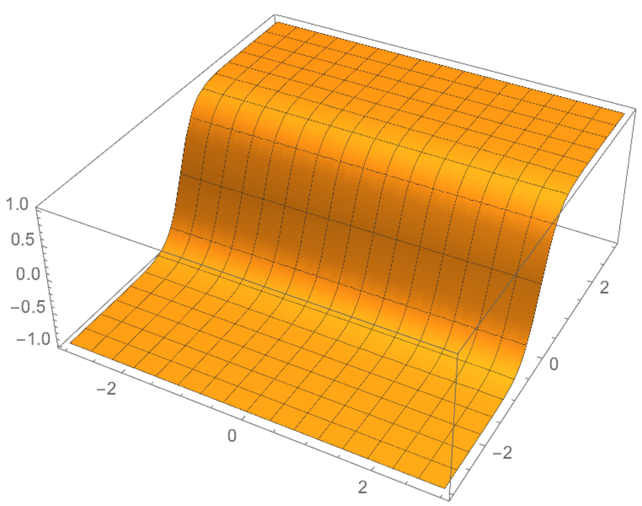


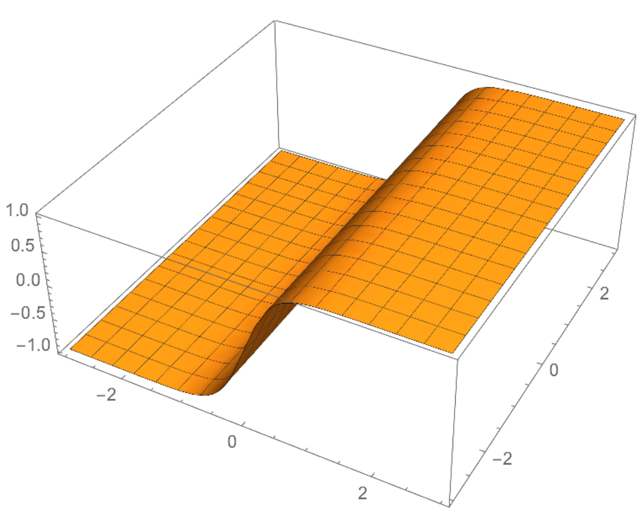
Example: Classify stable bundles



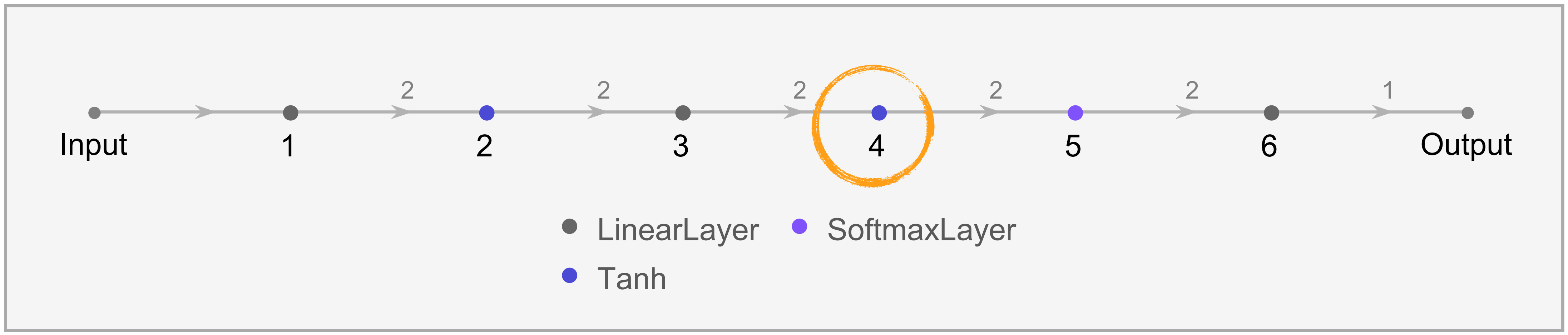
Example: Classify stable bundles

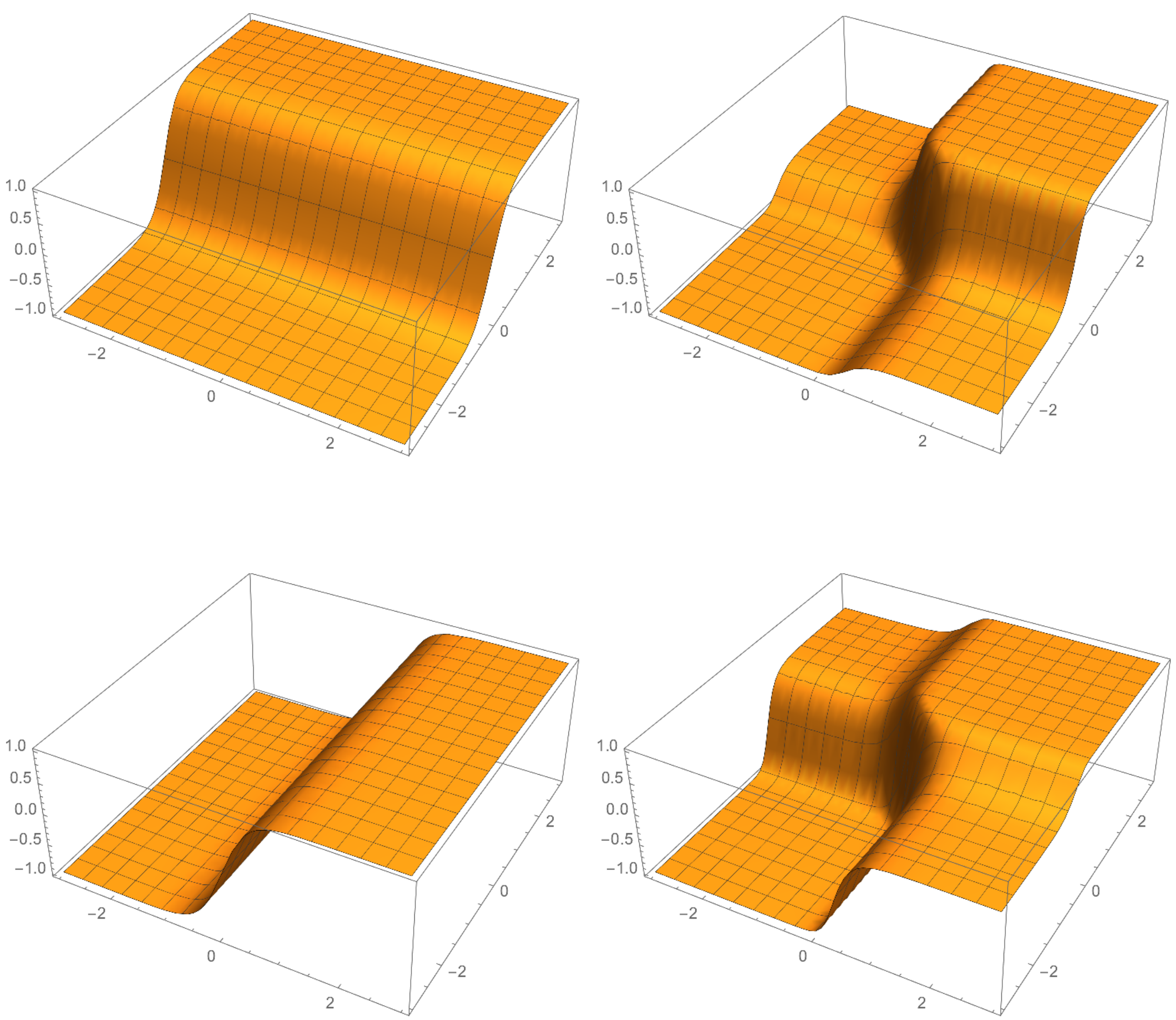




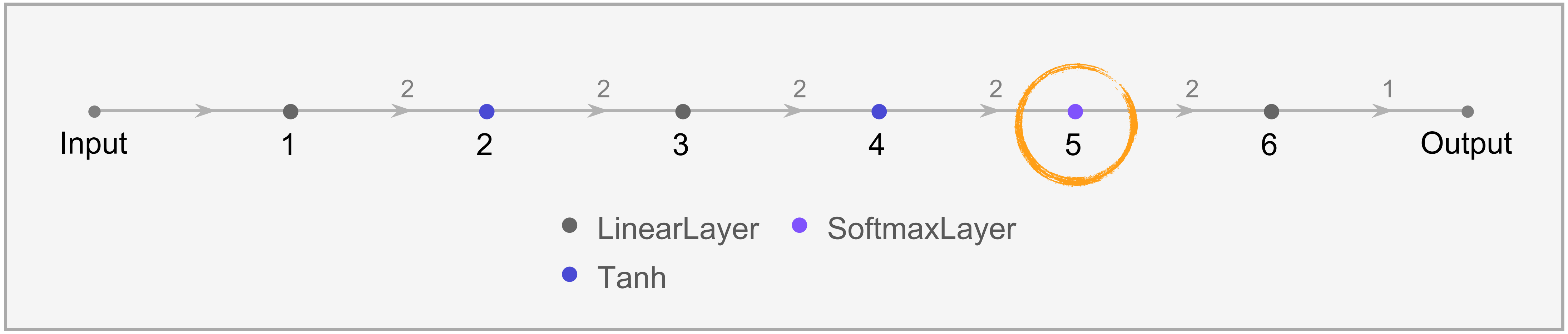


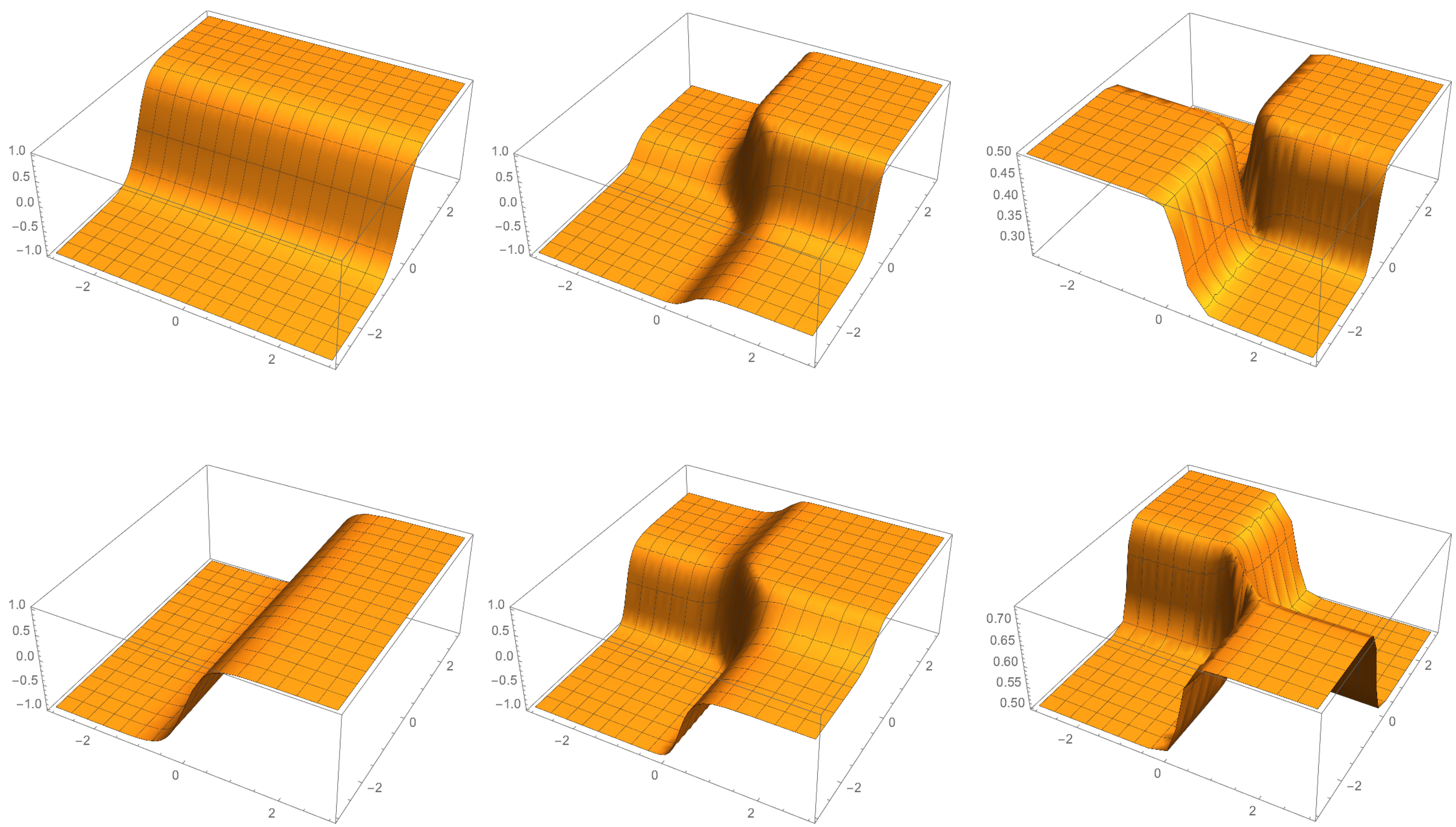
Example: Classify stable bundles



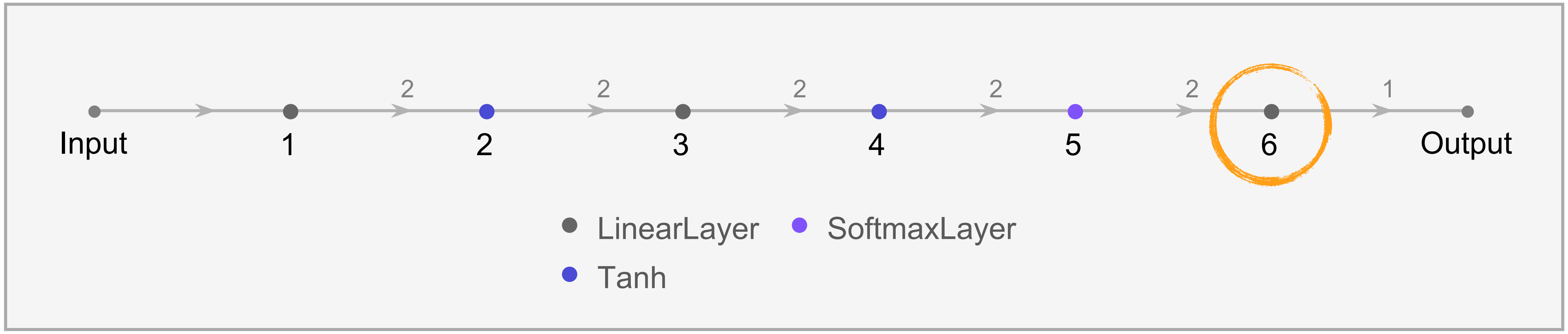


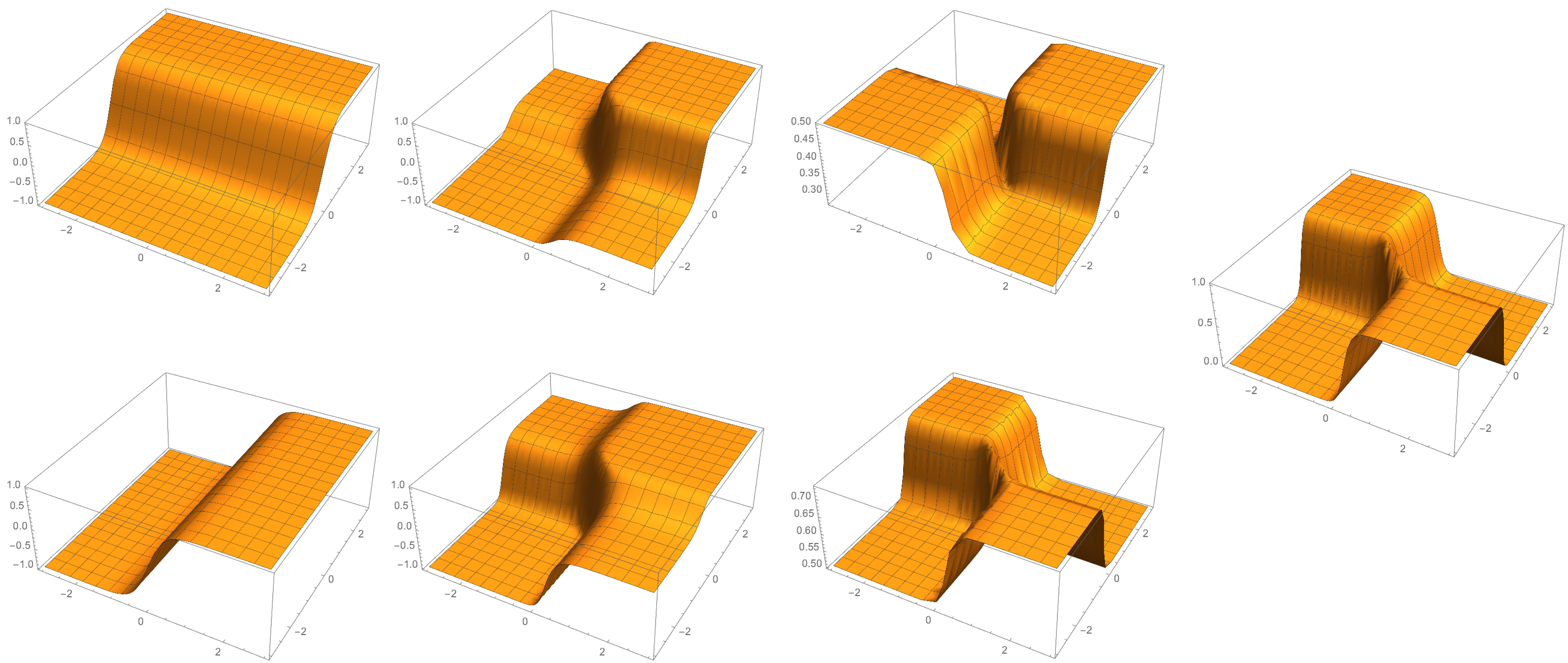
Example: Classify stable bundles



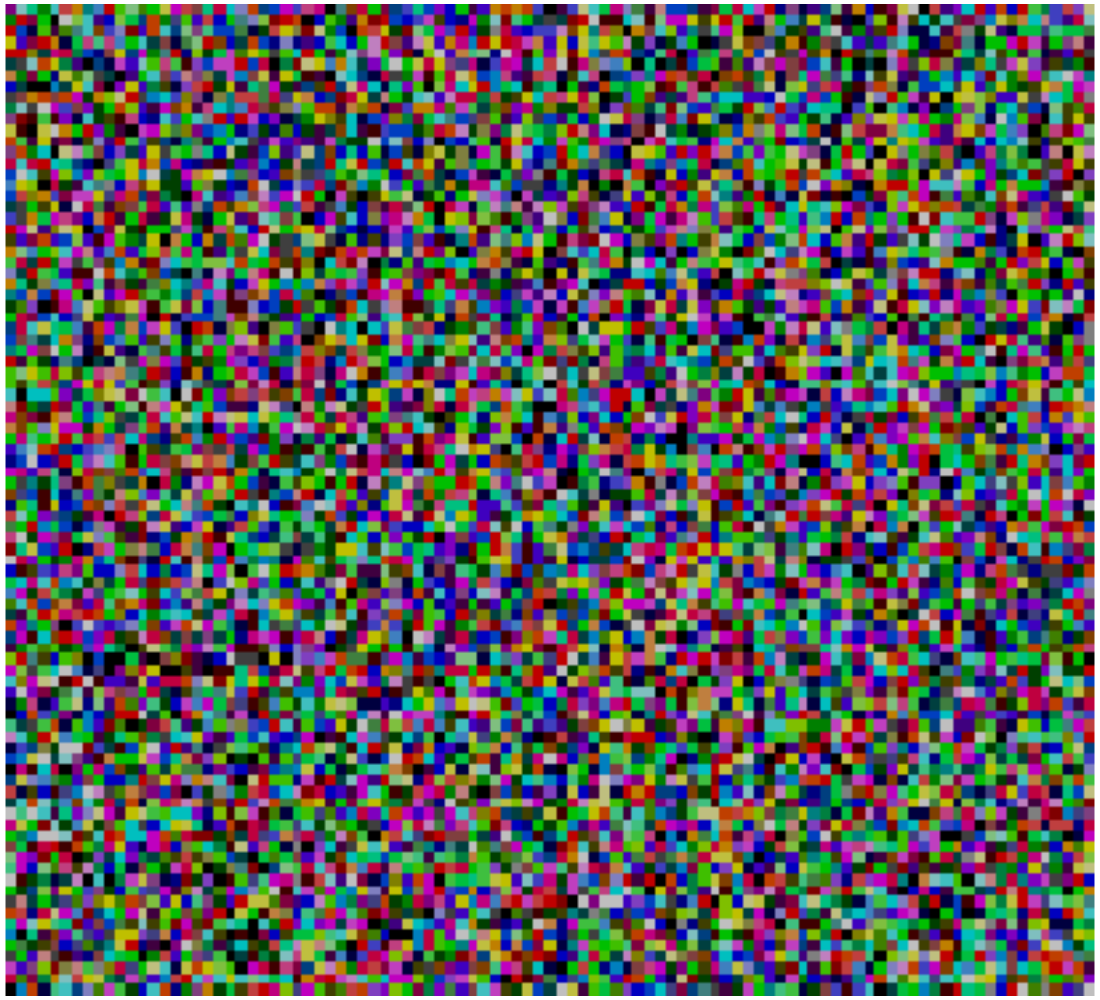


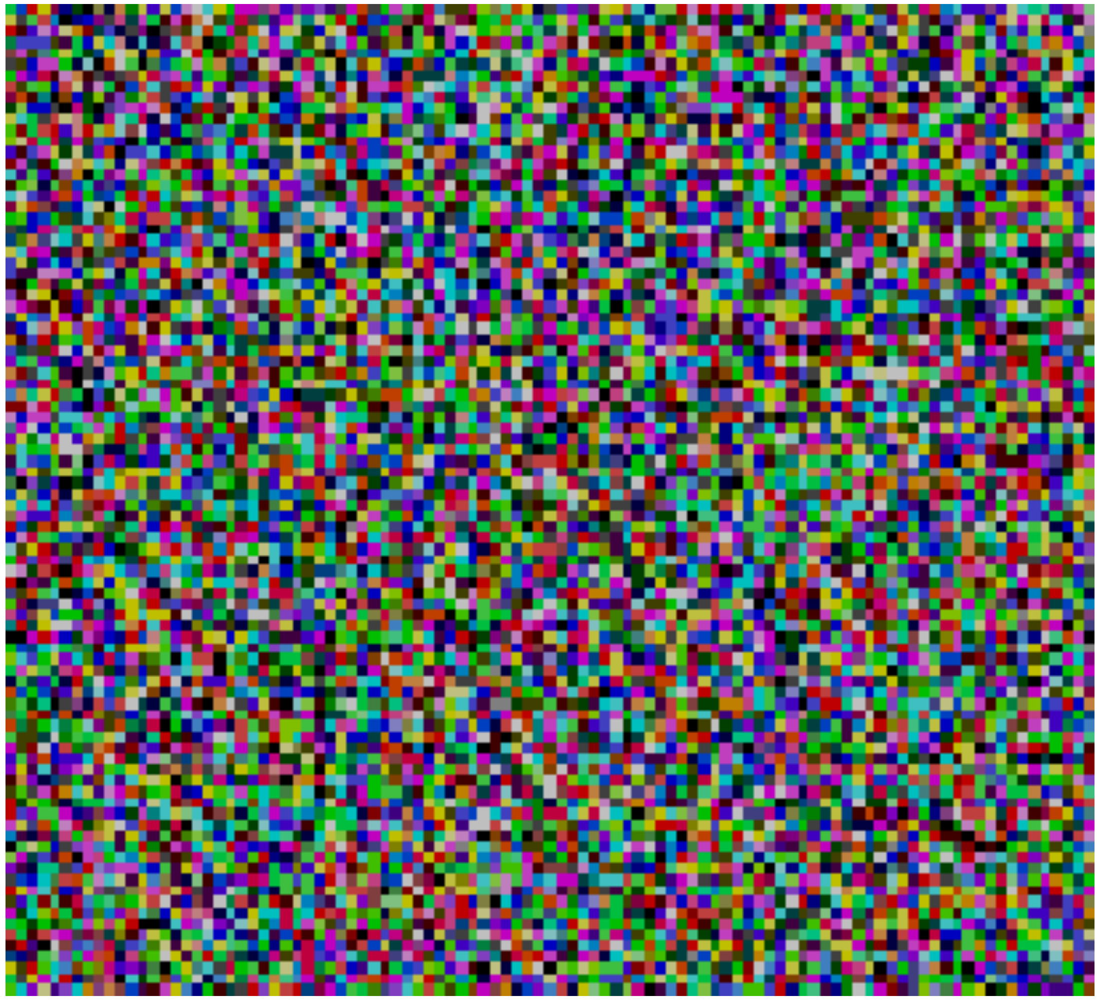
Example: Classify stable bundles



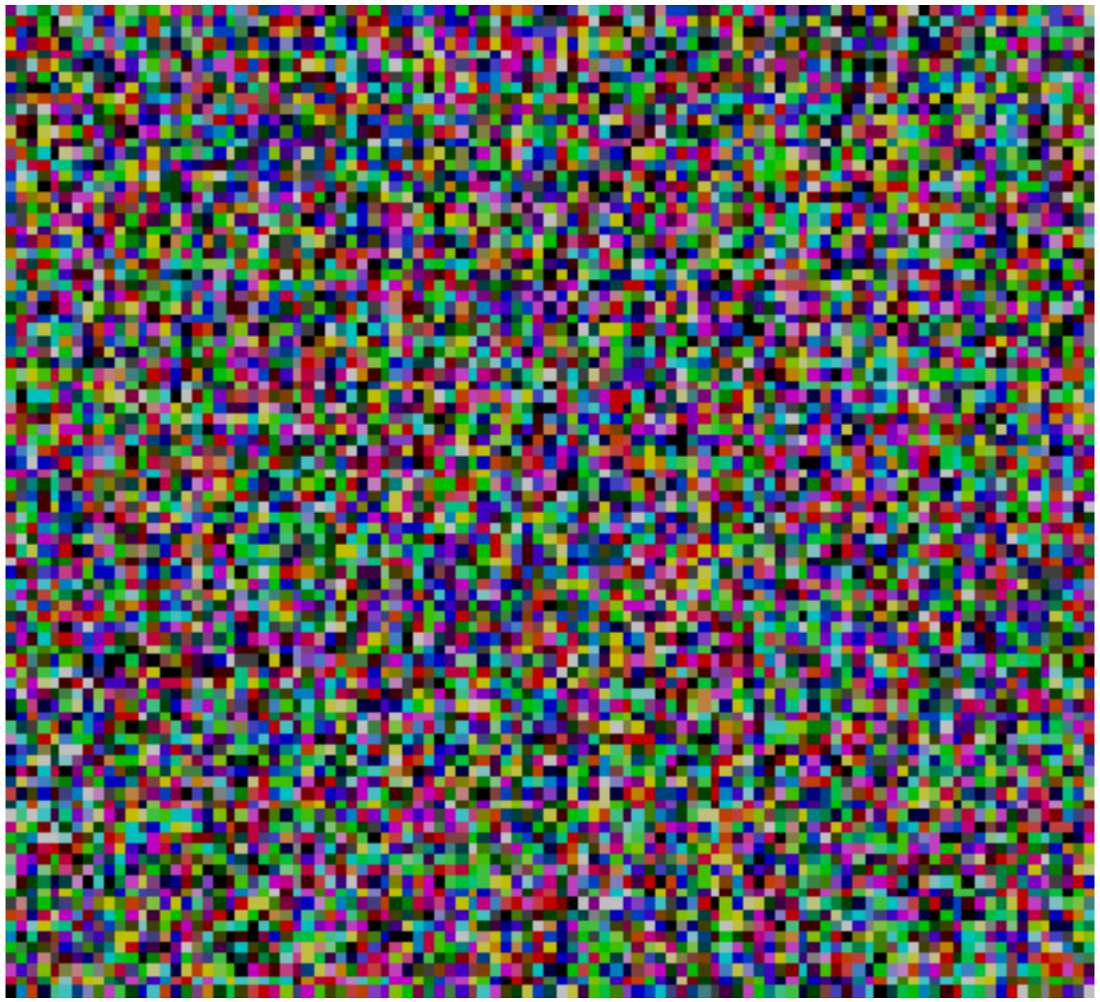


Example: Classify stable bundles

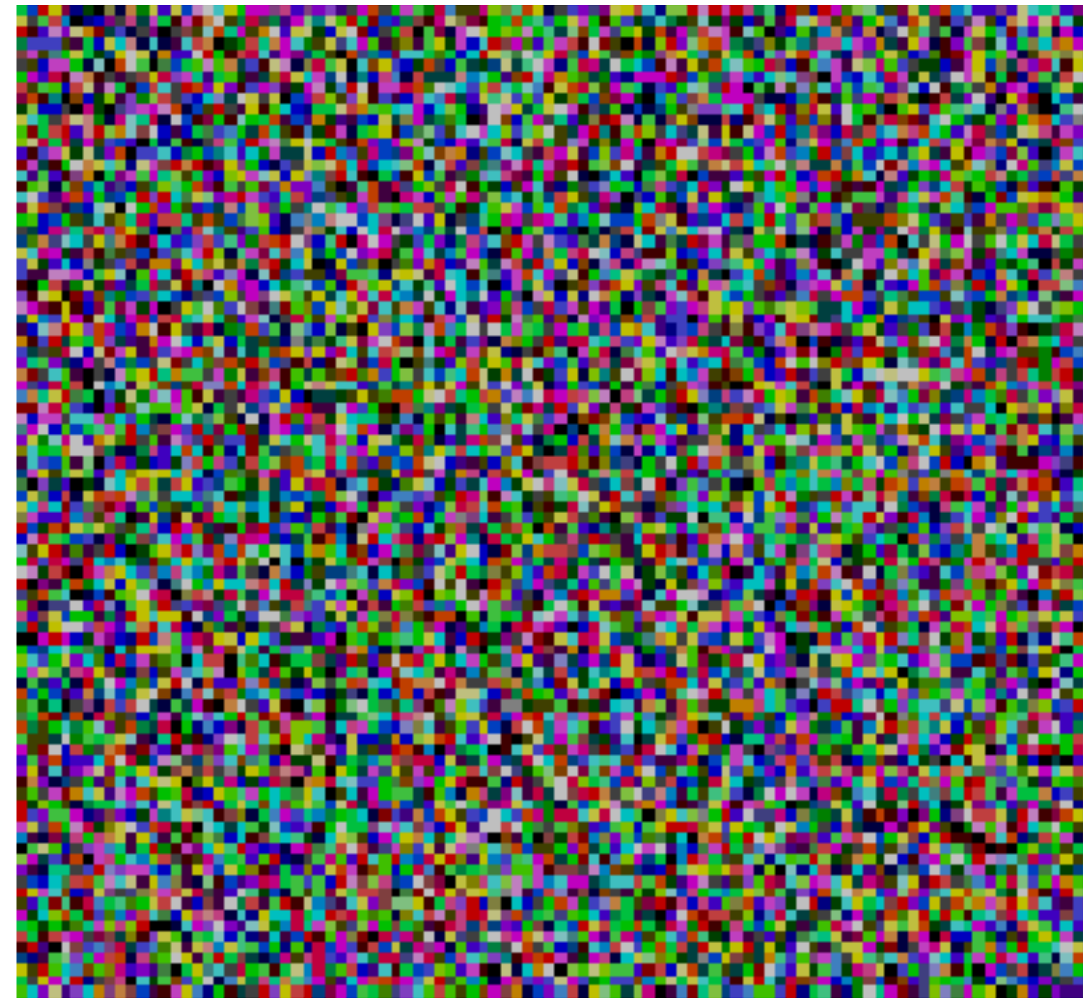




Example: Classify stable bundles



stable



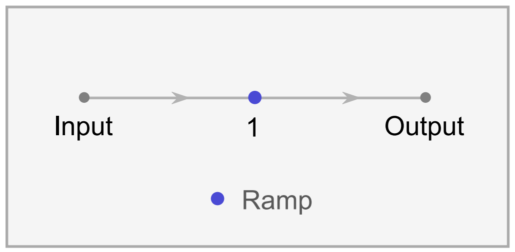
unstable

Example: Compute bundle cohomology

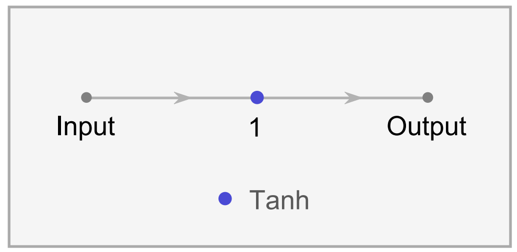
- Species computing $h^1(\mathcal{L})$ for Complete Intersection Calabi-Yau (codim 3) on $\mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^1 \times \mathbb{P}^3$
- 10 Generations
- Fittest 2 survive
- Reproduction via cell division
- Mutation rate 10%
- During mutation, insert/replace genes at any position ("gene splicing")
- Training time 45 seconds on 3000 bundles
- Fitness evaluated on another 7000 bundles

Example: Compute bundle cohomology

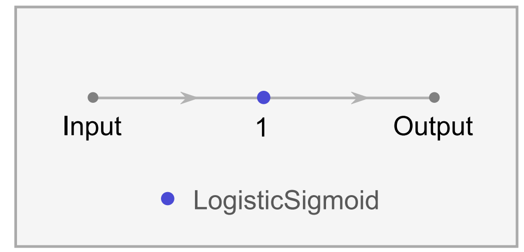
Available gene pool



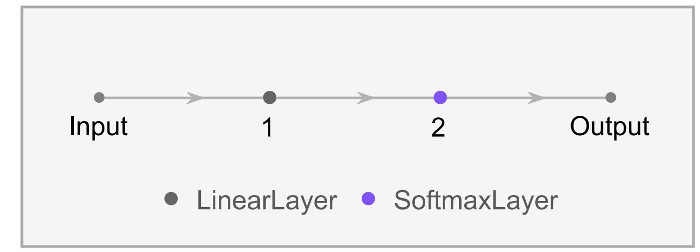
Ramp Layer



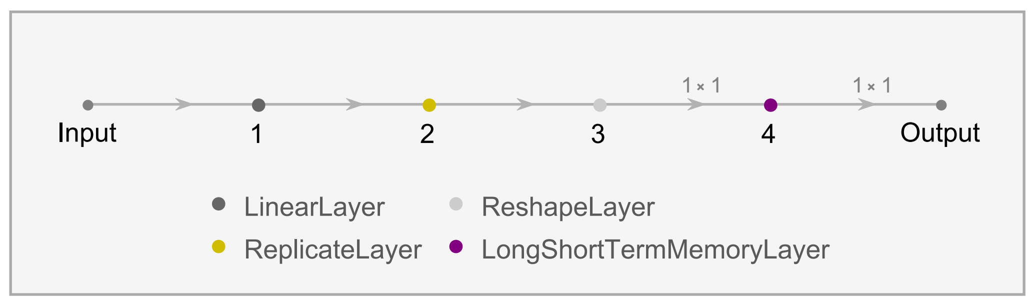
Tanh Layer



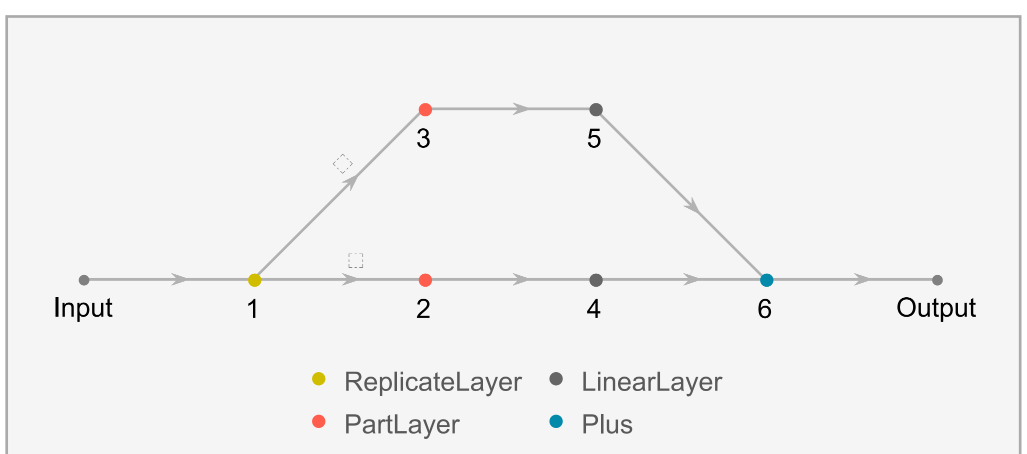
Logistic Sigmoid Layer



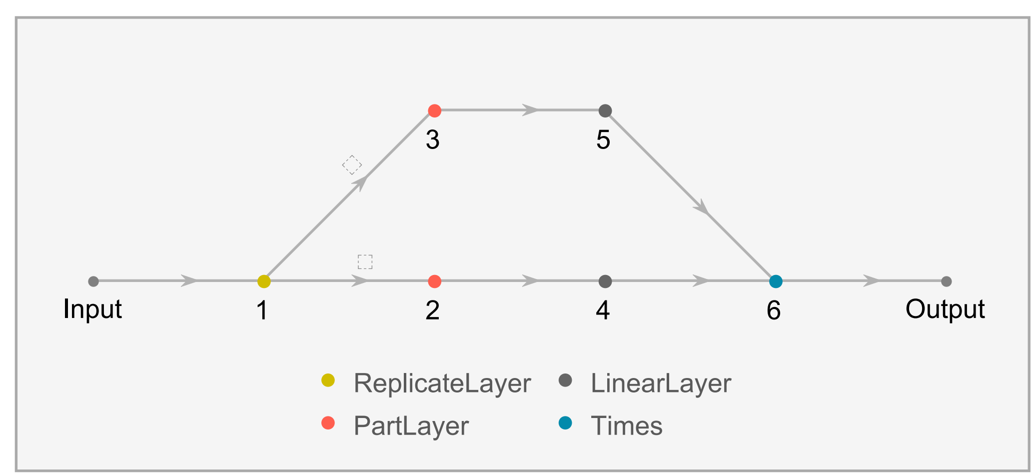
Softmax Layer



LSTM Layer

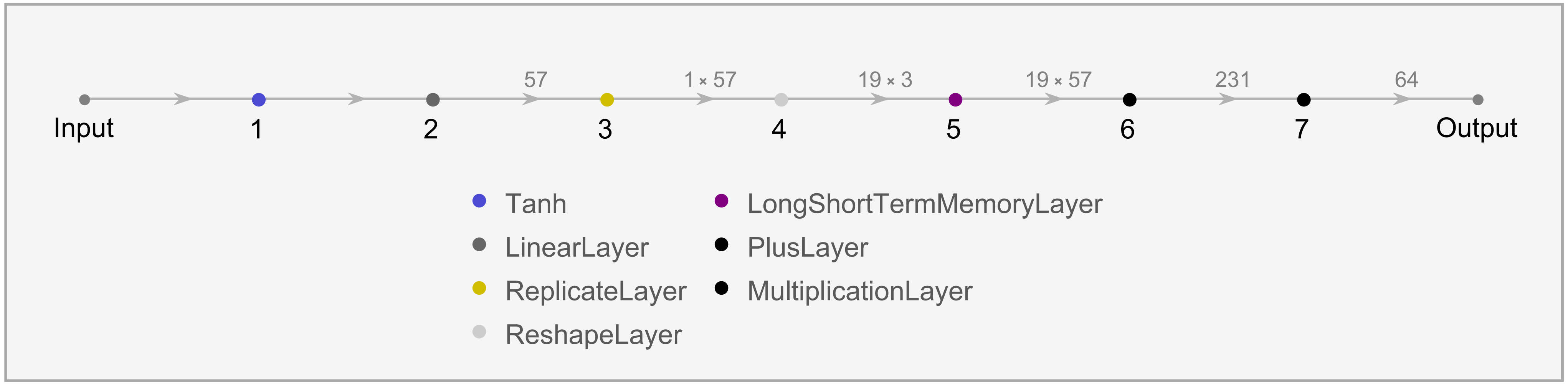


Addition Layer

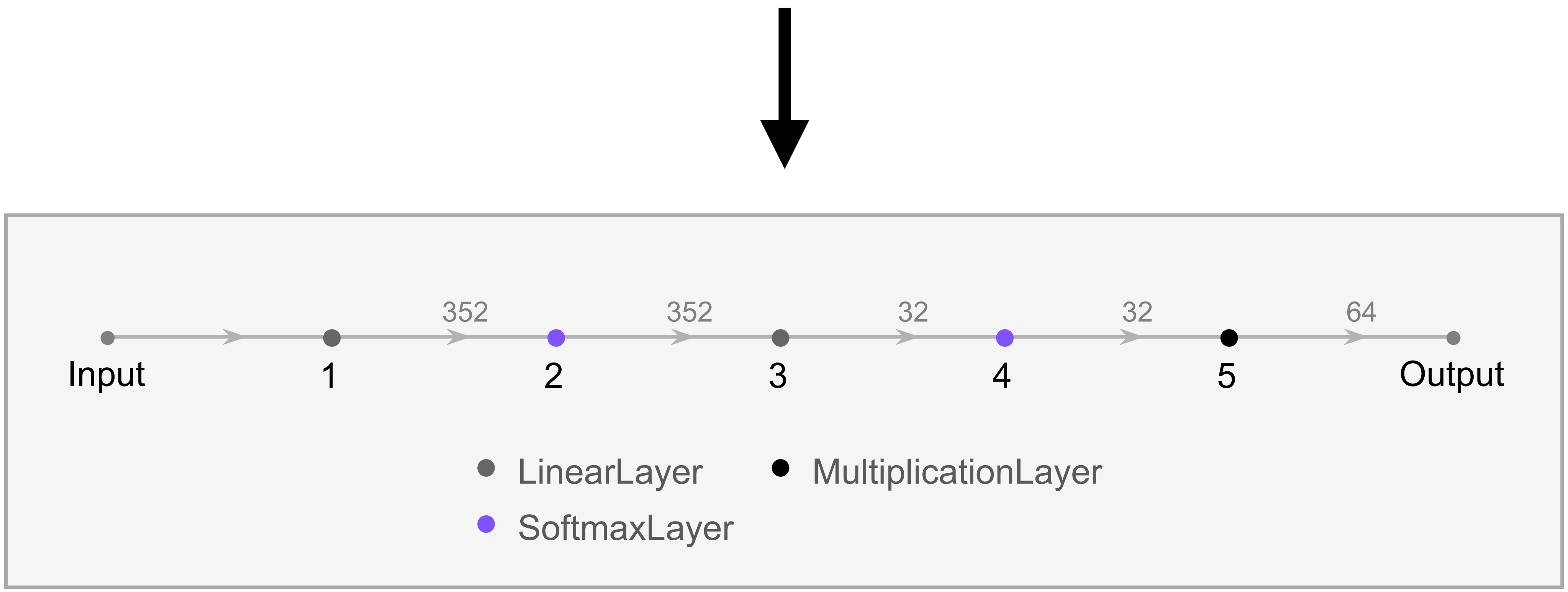


Multiplication Layer

Example: Compute bundle cohomology

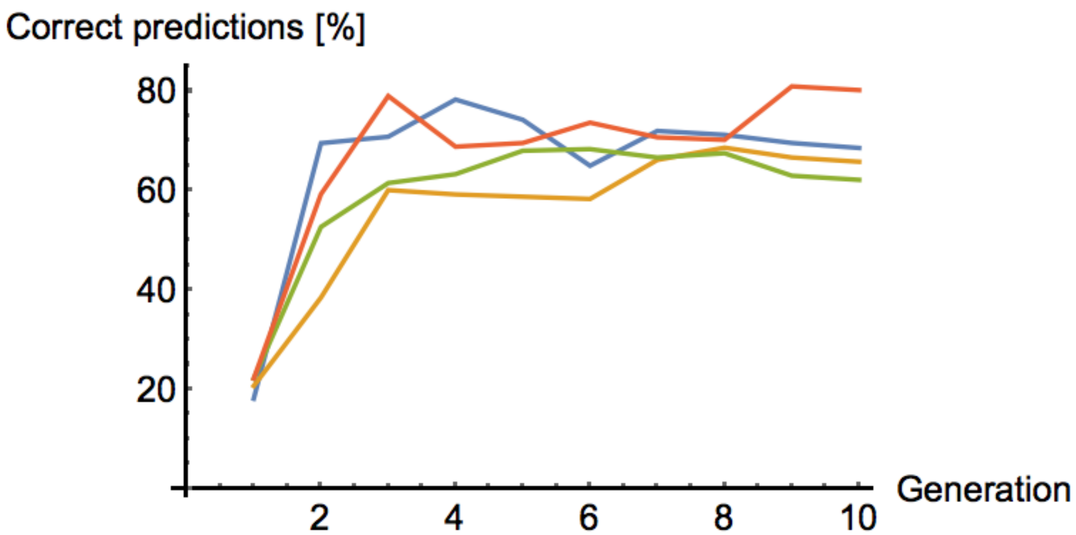


Species: $h^1(\mathcal{L})$, Generation: 1, Fitness 0.59

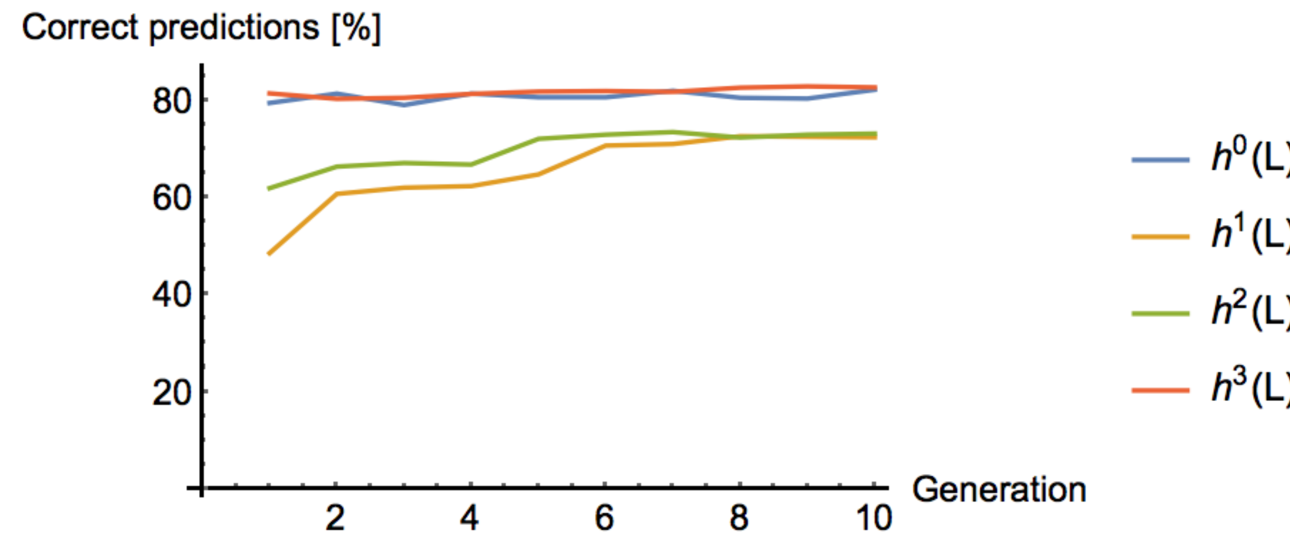


Species: $h^1(\mathcal{L})$, Generation: 10, Fitness 0.71

Example: Compute bundle cohomology



Average Fitness



Maximal Fitness

- $h^0(\mathcal{L})$ and $h^3(\mathcal{L})$ max out at 83%, $h^1(\mathcal{L})$ and $h^2(\mathcal{L})$ max out at 72%
- $h^1(\mathcal{L})$ and $h^2(\mathcal{L})$ more complex, evolve LSTM Layer
- Longer training of winner does not improve results
- Computation of 10 000 cohomologies takes
    - 5 hours using Koszul / Leray spectral sequences
    - 30 seconds with trained network
- Same network works on other CICYs (with same ambient space dimension) if trained with their data

Conclusion

- We have large sets of data in string theory with (potentially) interesting structure
  - Geometry (Calabi-Yaus)
  - String models
- Machine learning / NN can be applied to

  (A) Find & bypass implementations of algorithms

  (B) Approximate functions (predictor)

  (C) Classify data
- Tasks are versatile $\Longrightarrow$ dynamically evolve NN that is best equipped to handle individual situations
  - Feasible to evolve NN to compute bundle cohomologies
  - These NNs can be applied to different manifolds (if trained on them)

Thank you for your attention!