Introspection of Neural Networks

Looking inside the box of NNs

8.2.2023, TTT workshop TUM-IAS

Sven Krippendorf
(sven.krippendorf@physik.uni-muenchen.de,
@krippendorfsven)

Neural network as a black box?

while

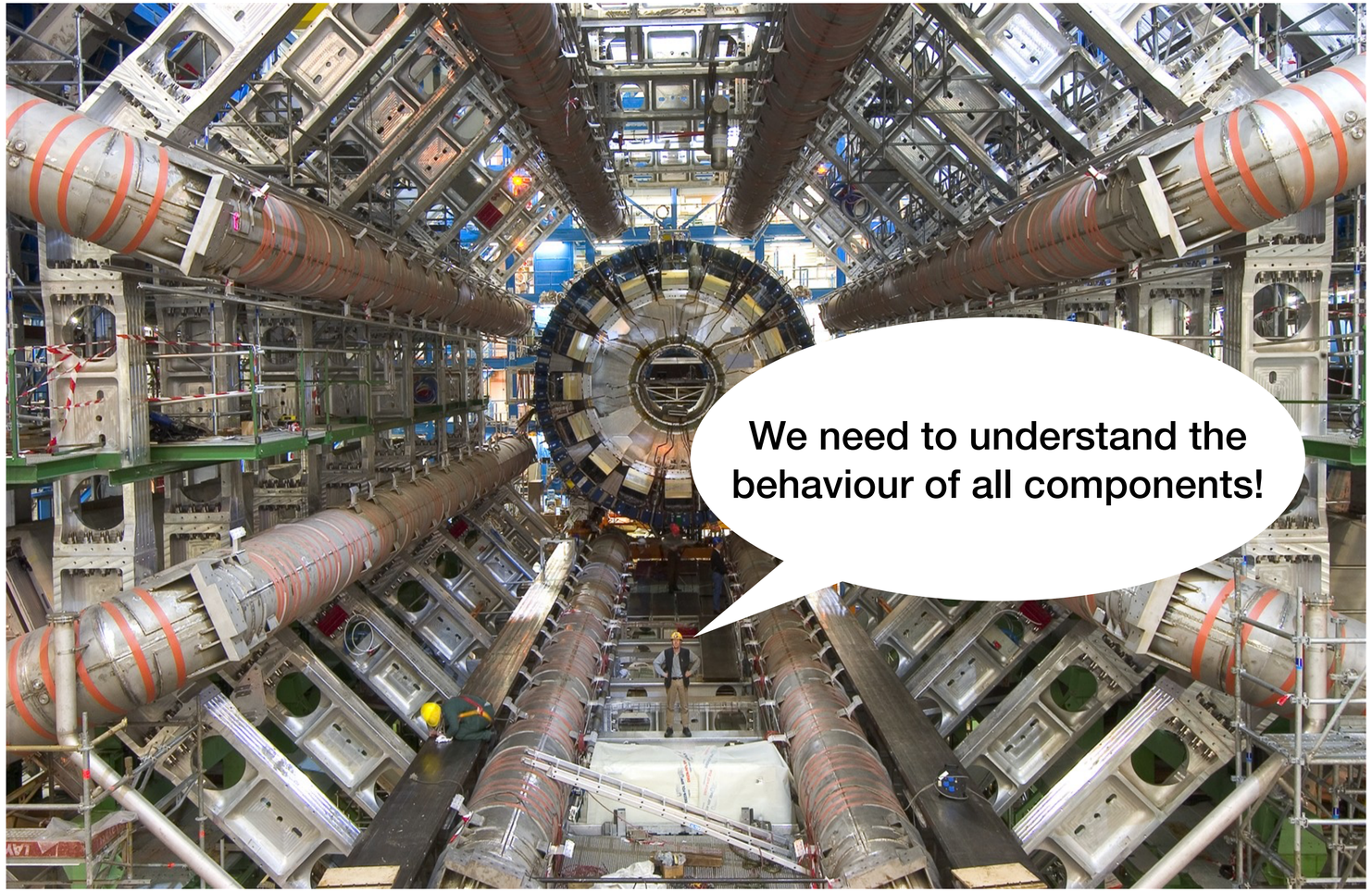
we have an analytic formula describing NNs

$$y = f(x, \theta)$$

from a first glance: lots of components whose interplay is not clear.

Neural network as a black box?

NN

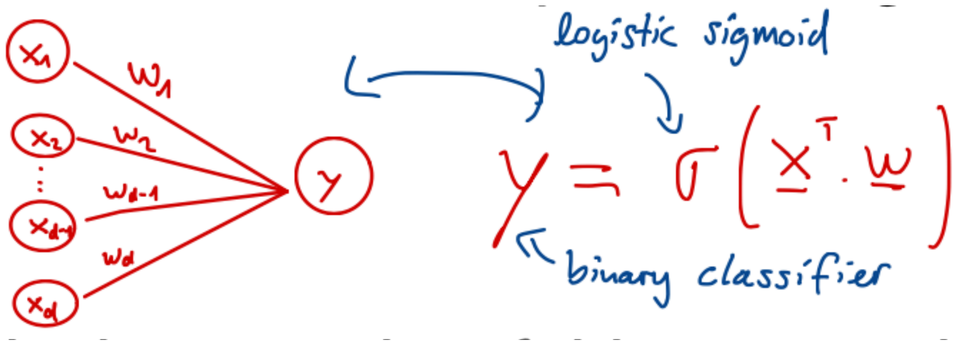


From a first glance: lots of components whose interplay is not clear.

Content

- Capacity of a single neuron: the X-OR crisis
- Decoding what your network has learned in data representations in latent dimensions
- Quantifying and visualising correlations: saliency maps
- Learning dynamics: understanding what your networks learns and what it does not learn?

A single perceptron

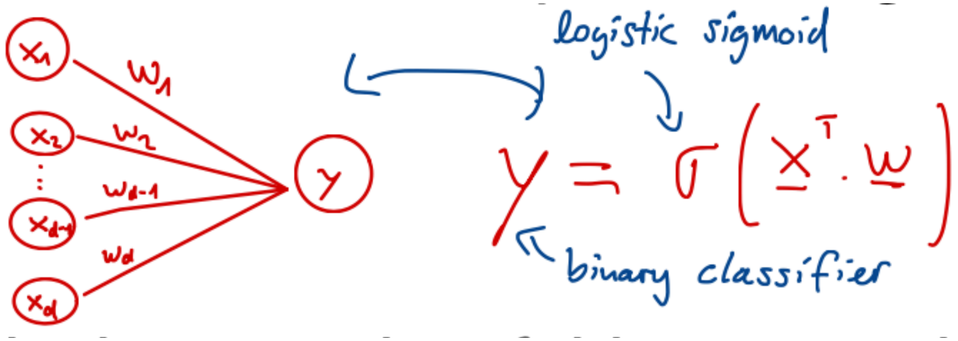


Literature: MacKay; Information Theory, Inference, and Learning Algorithms Chapter 40

Perceptron

Binary classifier = single neuron (perceptron)

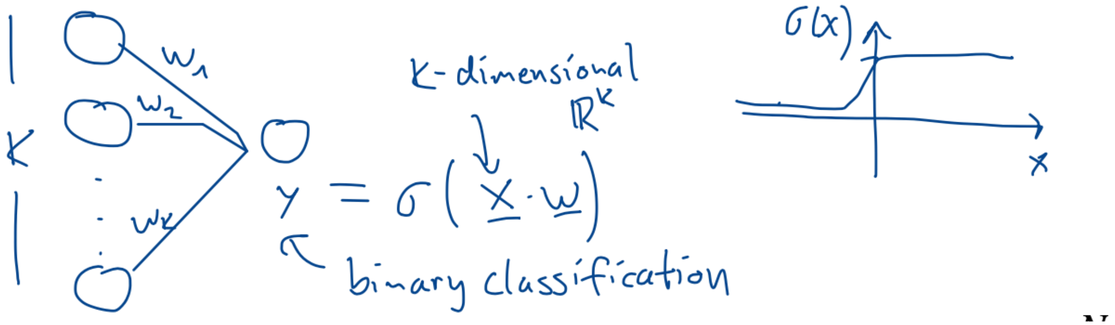
- Our binary classifier is an example of a single neuron (perceptron)
- We study the capacity of this system, i.e. how much information can be stored by training a neural network.
- Enables you to understand classifying techniques and provides you with the tools to design algorithms with enhanced capability.
- Capacity = infinite (as each weight is real number)? No, as receiver is not able to examine the weights directly and not able to probe the inputs with arbitrary weights.

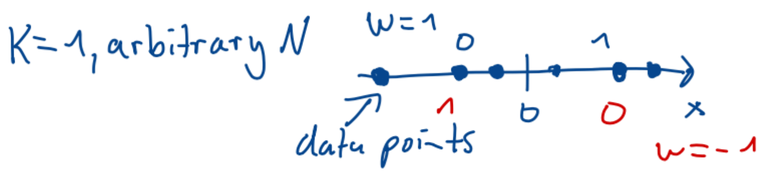


Perceptron

Capacity

- K inputs for perceptron, N data points. Possible number of binary labels 2^N.
- What is the probability that all N bits are correctly reproduced?
- How large can N become, for a given d, while keeping this probability close to 1?
- Assumption: Generic position of data points (i.e. subset of K or less elements are linearly independent)
- Assumption: no bias for our perceptron
- T(N,K) number of distinct threshold functions



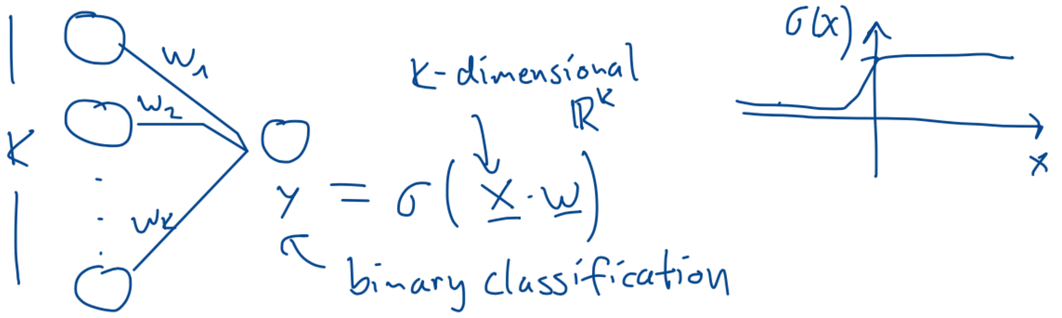


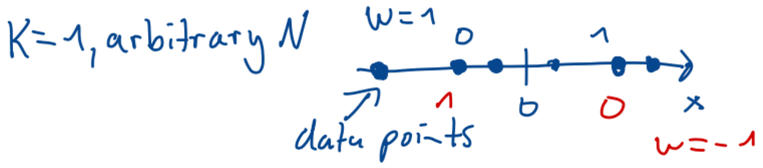
$T(N,1) = 2$ 2 threshold fcts.

Perceptron

Capacity

- K inputs for perceptron, N data points. Possible number of binary labels 2^N.
- What is the probability that all N bits are correctly reproduced?
- How large can N become, for a given d, while keeping this probability close to 1?
- Assumption: Generic position of data points (i.e. subset of K or less elements are linearly independent)
- Assumption: no bias for our perceptron
- T(N,K) number of distinct threshold functions for fixed data points.

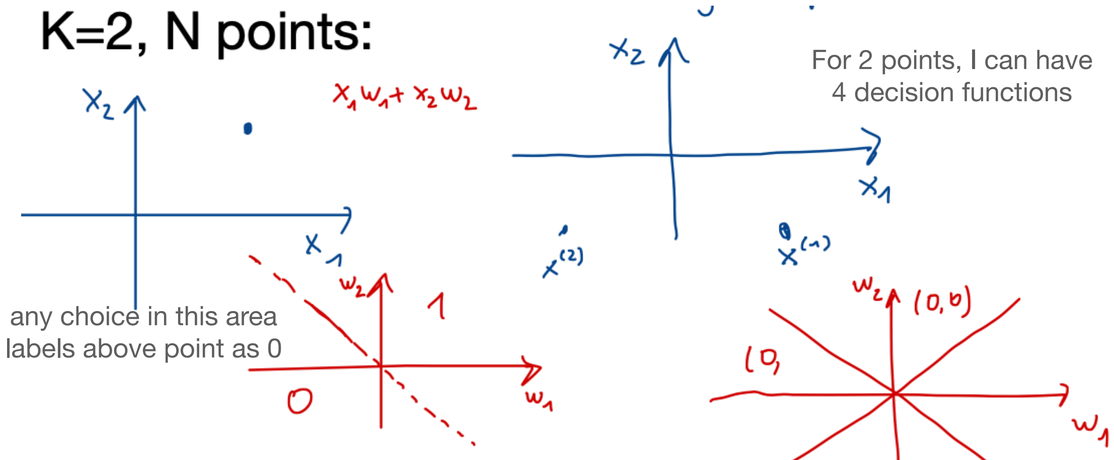




N=1, arbitrary K

arbitrary dim. $w = \pm x$ $\Rightarrow T(1,K) = 2$ only one point (both labellings possible)

K=2, N points:



any choice in this area labels above point as 0

For 2 points, I can have 4 decision functions

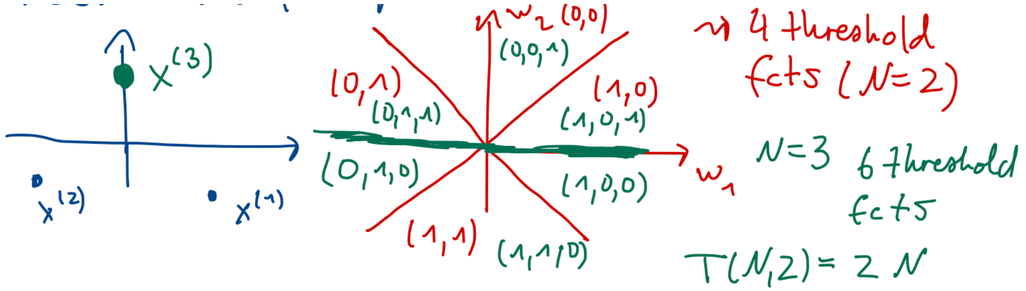
Perceptron

Capacity

- K=2, N points continued
- We see that we cannot reach all possible threshold functions with a single perceptron!
- General case possible with appropriate recursion relations:

$$T(N,K) = 2\sum_{k=0}^{K-1}\binom{N-1}{k} = \begin{cases} 2^N, & K \geq N \\ 2\sum_{k=0}^{K-1}\binom{N-1}{k} & K < N \end{cases}.$$

- For K<N: can memorise N=2K labels but will fail to memorise more

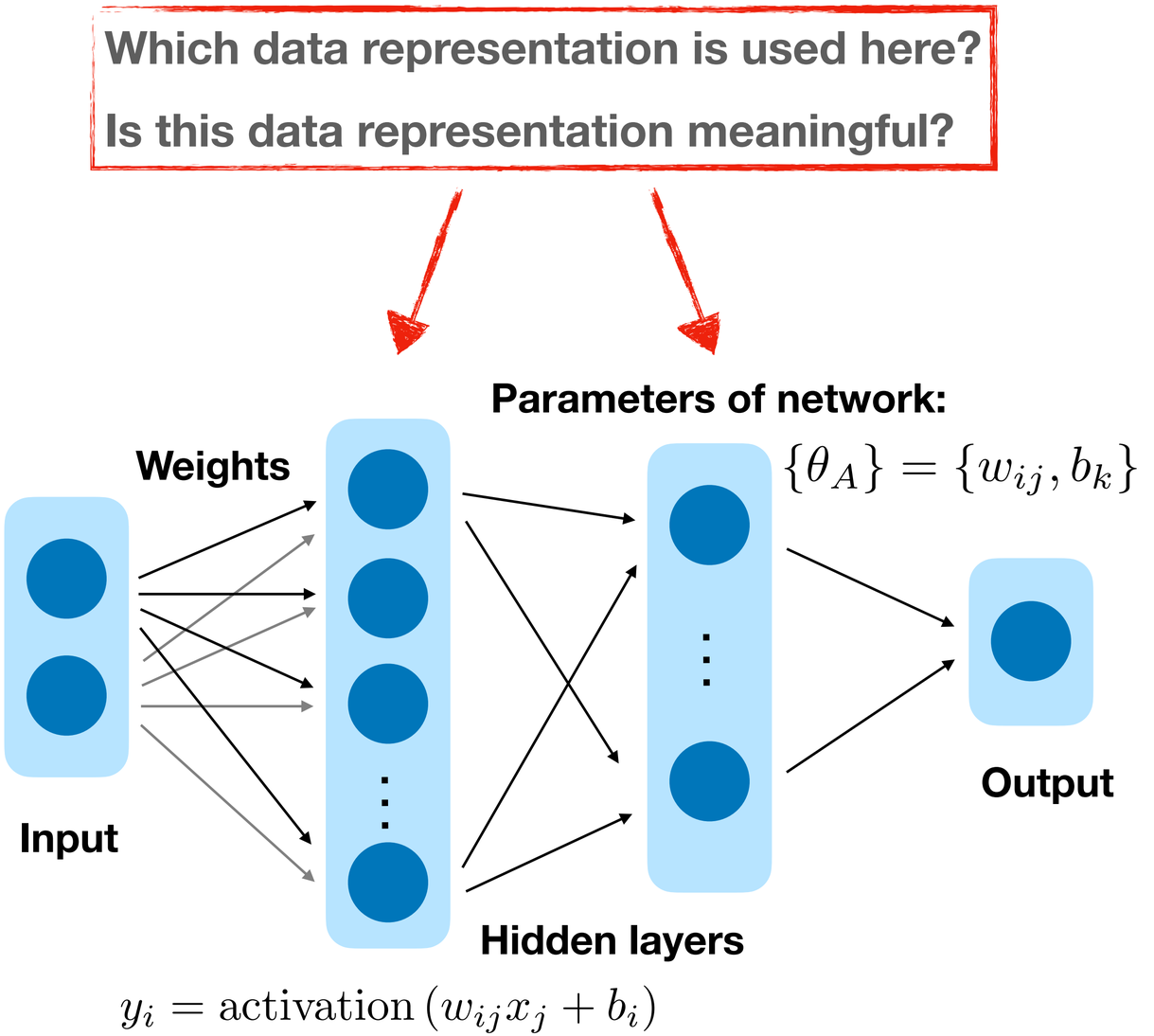


Exercise: special situations appear with non generic points. Can you build the XOR function with a single perceptron?

Data	XOR
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

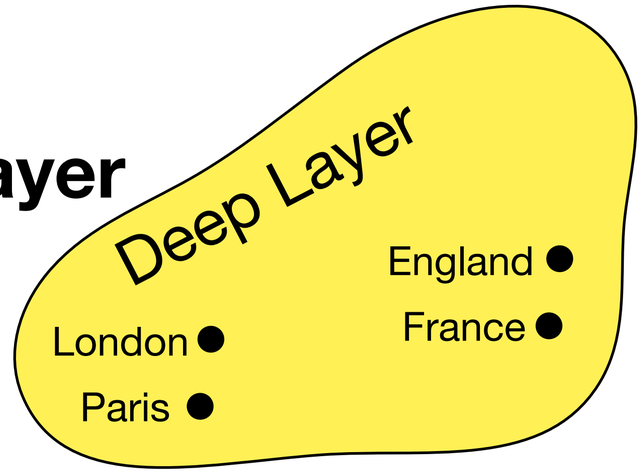
Introducing hidden layers allows to circumvent these limitations (Universal Approximation Theorem)

Neural Network Representations in Hidden Layers

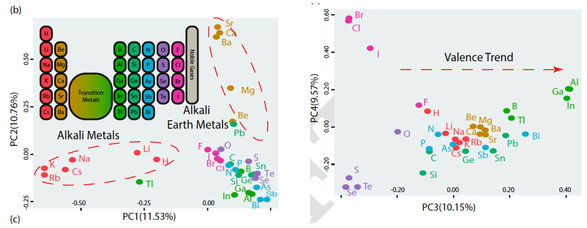


The inspiration

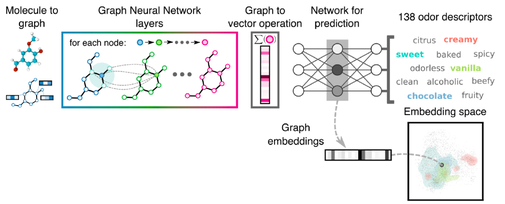
Meaningful embedding in second to last network layer

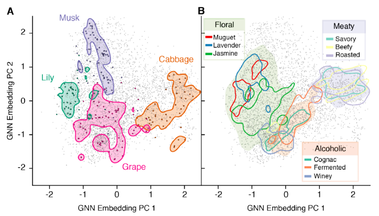


- Multiple layers are necessary to reach meaningful decisions (limitations of single perceptron)
- Embeddings deep in neural networks become meaningful (grouping data sharing the same properties):
- Word2Vec (England - London = Paris - France)
  (1301.3781)
- Applications in Atom2Vec → Periodic Table
- Classifying scents of molecules (GNN)
- **Can we apply this to detect symmetries in Physics?**



1807.05617





1910.10685

Finding symmetries with Neural Networks

Solution

1. **Step 1: Find invariances**
   - Set up classification problem (e.g. value of potential, Hodge numbers)
   - Analyse position in embedding layer (directly or via dimensional reduction)
2. **Step 2: Which symmetry is generating them?**
   - Analyse nearest neighbours in the pointcloud (dim. reduction if necessary)

$$p' = p + \epsilon_a T^a p$$

   - Appropriate regression problem to constrain all components of generators.

Examples Step1: Finding Invariances

Scalar potential and superpotential

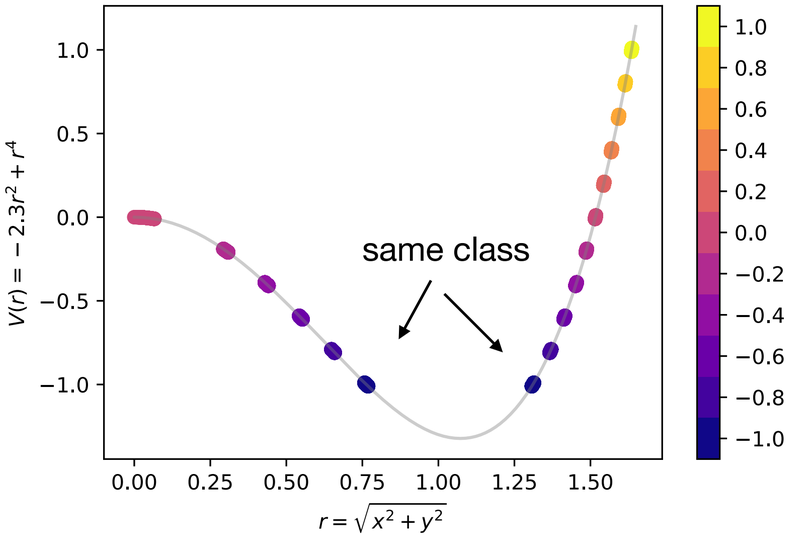
- Standard Higgs potential:

$$V(x,y) = V(r^2 = x^2 + y^2) = r^4 - ar^2$$

- Classification problem:

Input: (x,y) Output: $V_k = \left[\frac{k}{5} - 10^{-3}, \frac{k}{5} + 10^{-3}\right]$

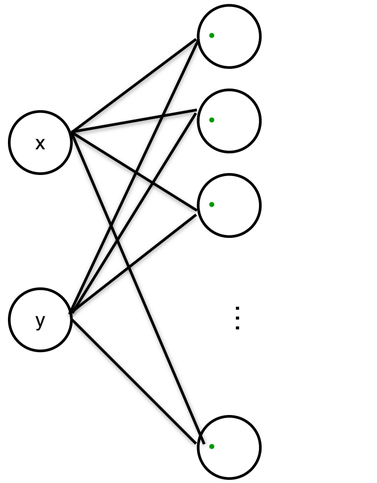
- Sample points for these classes



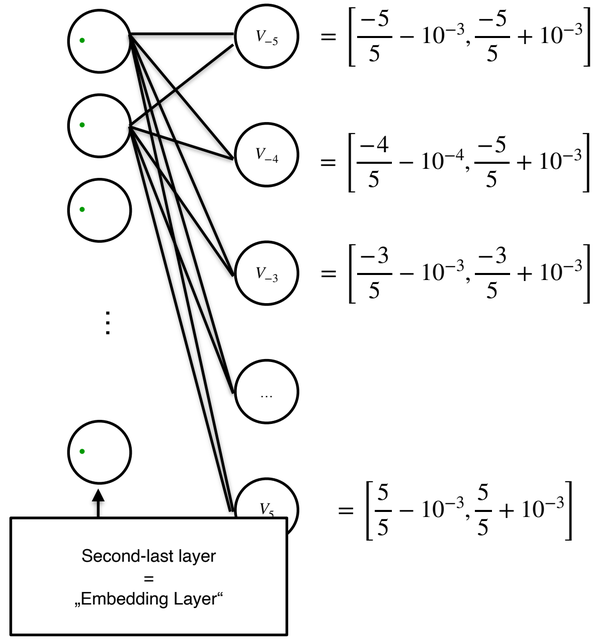
Examples Step1: Finding Invariances

Scalar potential and superpotential

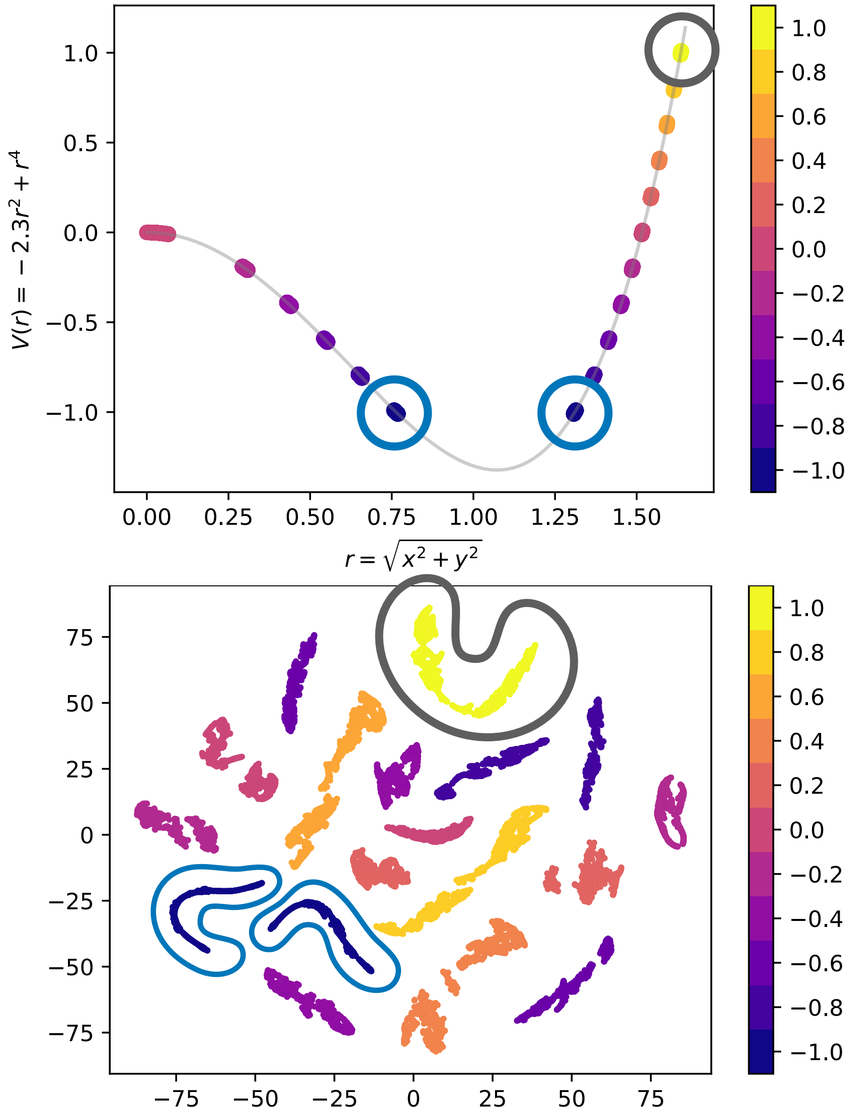
- Neural Network:



$$= \max\left(0, \vec{\mathbf{x}}\mathbf{W} + \mathbf{b}\right)$$



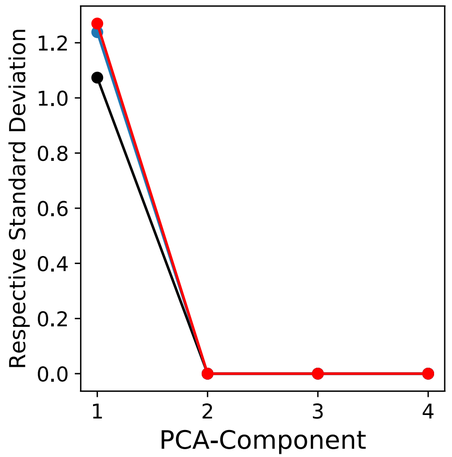
- Trains easily to accuracy > 95%.
- Visualise embedding layer representation (80-dim.) via dimensional reduction (here TSNE)

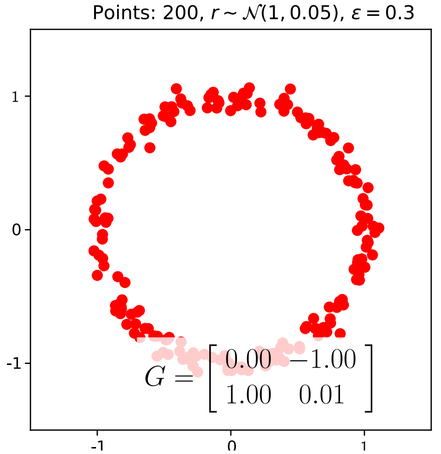


Finding Generators

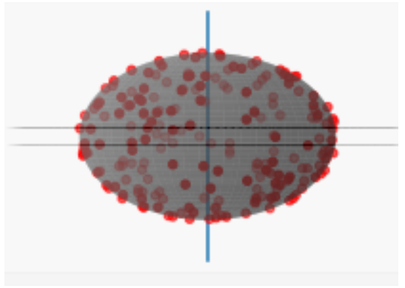
Finding Generators

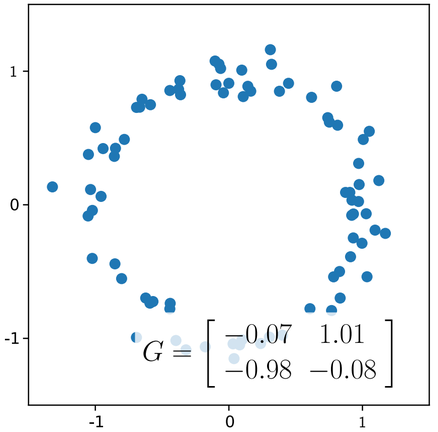
$$p' = p + \epsilon_a T^a p$$

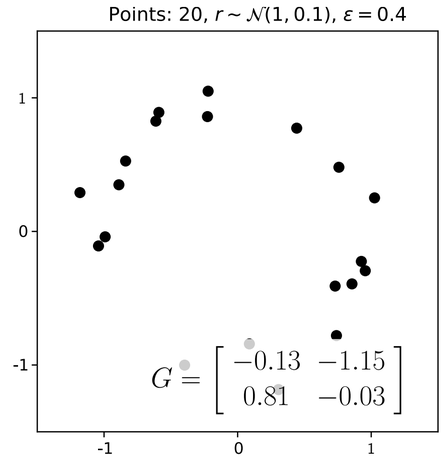




- Finding generators in noisy point cloud P:
  1. Dimensionally reduce and center data
  2. Pick hyperplane spanned by two random points $\in P$. This is the hyperplane the generator is acting on.
  3. Filter points in this hyperplane (ϵ-neighbourhood)
  4. Filter points close to each other
  5. Use these points to constrain the generator T via:
     $$p' - p = \sigma_H(p, p')/|p|(p' - p)Tp$$
     Additional constraints are given by requirement that T acts in hyperplane.
  6. Linear regression
  7. Repeat (2-6) [all directions]. Apply PCA on the generators found.
- Algorithm is able to find generator (error depending on quality of data)
- Clear evidence for only one generator here for SO(2)



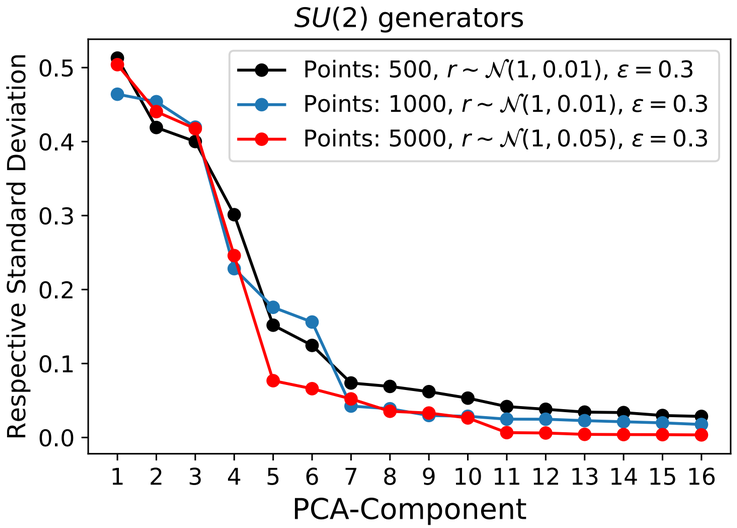


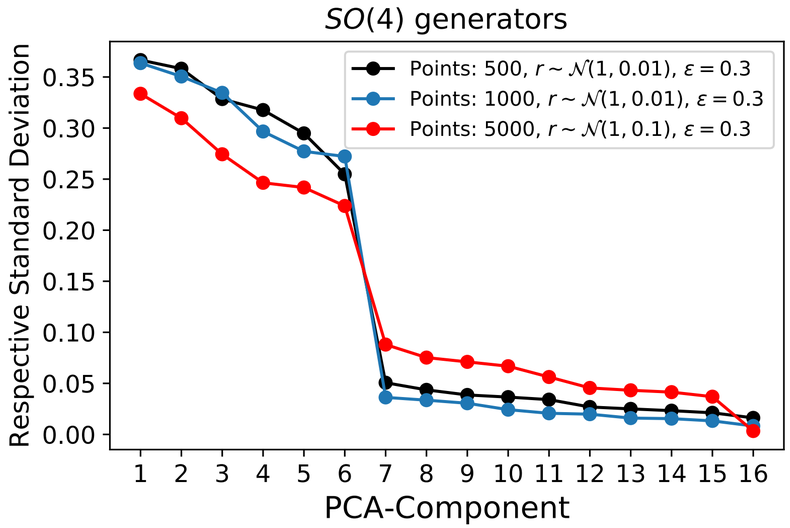


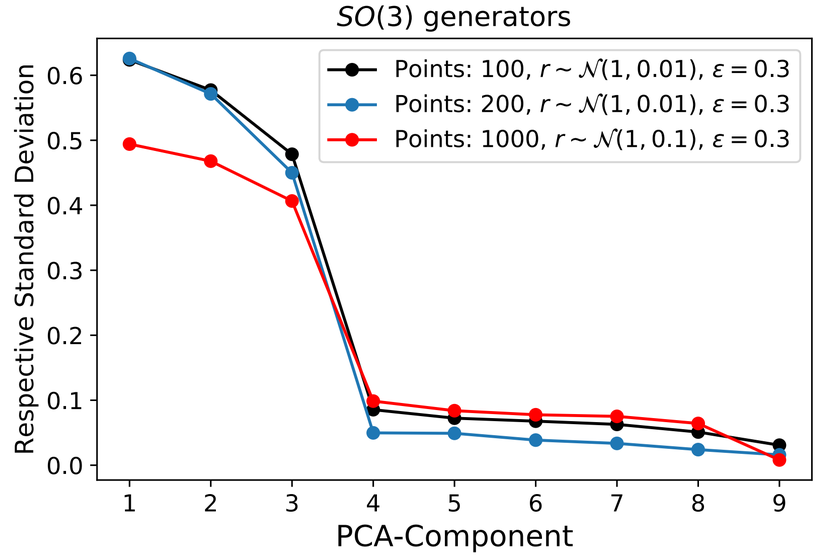
Finding Generators

Further Examples

- Successfully applied for SO(n) groups and subgroups
- Slight modification for subgroups (multiple pointclouds).





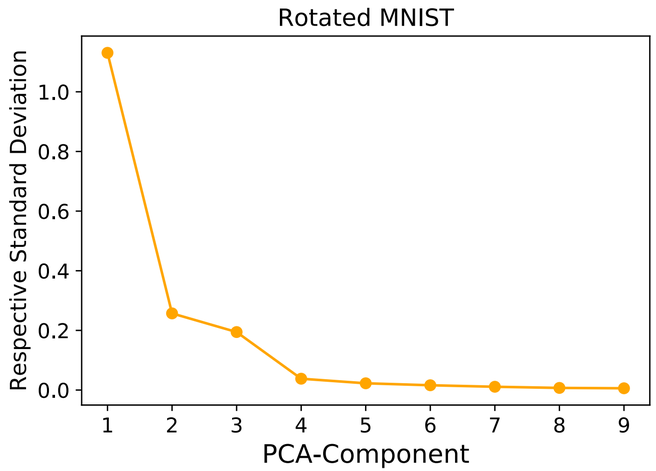


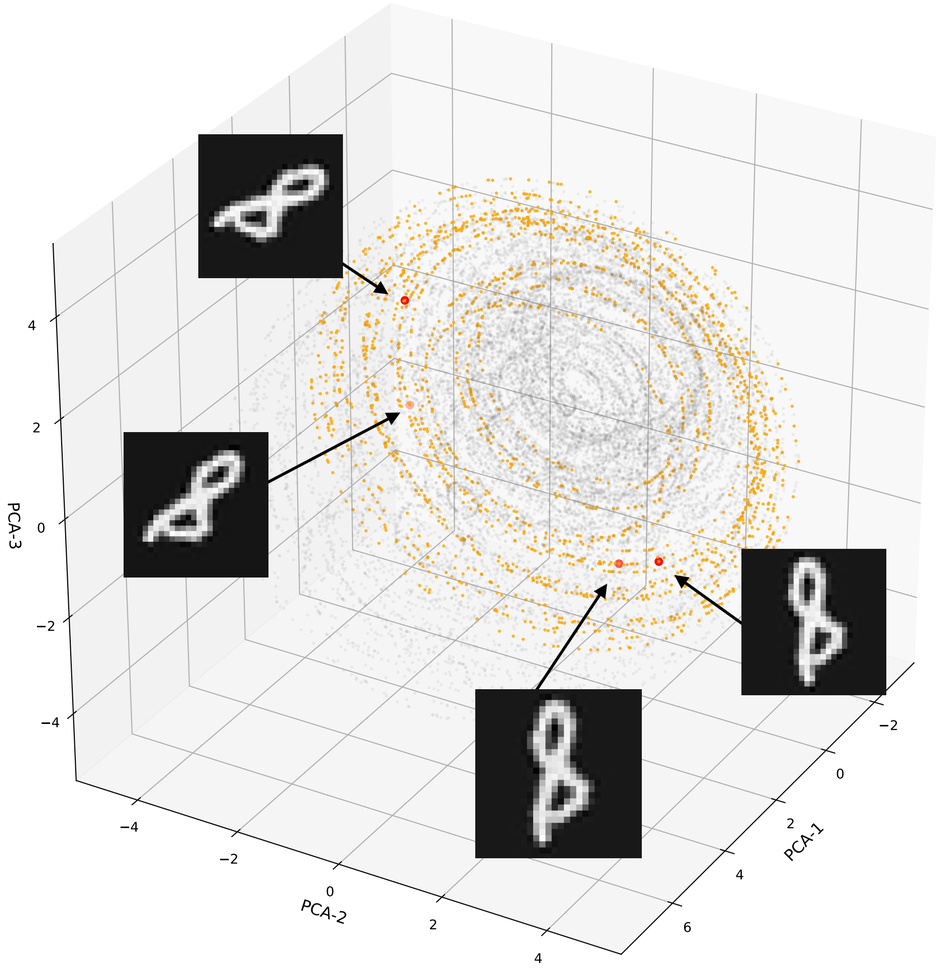
High dimensional input data

Rotated MNIST

- Generalisation to images
- Clear identification of dominant generator

$$G = \begin{pmatrix} -0.06 & -0.00 & -0.07 \\ 0.01 & -0.01 & 1.00 \\ 0.08 & -0.99 & 0.04 \end{pmatrix}$$





Saliency Maps

What does a trained NN do?

- It's a function, usually a chain of functions in feed-forward NNs:

$$f : \mathbb{R}^n \to \mathbb{R}^d,\ x \mapsto y = f(x)$$
$$f(x) = f_N \circ f_{N-1} \circ \cdots \circ f_1(x)$$

- Each function typically is a linear map, followed with a non-linearity.
- It's differentiable!
- **How do our decisions depend on a particular feature in a neural network?**
- This enables access to information on how much changes in one pixel change the output, e.g. gradients w.r.t. a single input pixel:

$$\frac{\partial f_a(\mathbf{x})}{\partial x_i}$$

over to Jupyter Notebook

Neural Network Dynamics

Can I choose hyperparameters without having to train NNs?

Why is understanding NNs important?

- To trust predictions of NNs, we need to understand their performance: systematic and statistical uncertainties. How are neural network prediction biased?
- Efficiency of training process: hyper-parameter tuning of large language models is very costly (e.g. you only want to run them at production)

NN as dynamical systems

- Can we understand supervised learning of neural networks?
  - How do features emerge and provide an appropriate dynamical framework?
  - Can we predict the network performance after training without having to train the network?
  - Can we make supervised training of neural networks more efficient (e.g. use less data, different optimisers, different architectures)

Neural Network

Training process — supervised learning

- NN architecture (e.g. fully connected/dense NNs), data (input/target output), loss function, optimizer fixed
- Randomly initialised NN = NN parameters drawn from appropriate distributions (often normal distributions appropriately rescaled according to the dimension)
- Loss function, e.g. mean-squared-error for regression task
- Update the NN parameters many times, e.g. using standard gradient descent:

$$\theta_i \to \theta_{i+1} = \theta_i - \eta \nabla_i \mathscr{L}(\mathscr{D})$$

- Our NN as a function:

$$f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{d=1} \ , \ (t, x, \theta) \mapsto f(t, x, \theta)$$

Neural Network

Dynamics at large widths

- The dynamics neural networks simplify in the infinite width limit.
- Work in continuous time limit:

$$\theta_{i+1} - \theta_i = -\eta \nabla_\theta \mathscr{L}$$
$$\dot{\theta} = -\eta \nabla_\theta \mathscr{L} = -\eta \nabla_\theta f(y) \nabla_{f(y)} \mathscr{L}$$

- Here: neural tangent kernel (NTK)

$$\dot{f}(x) = \nabla_\theta f(x)\ \dot{\theta} = -\eta \nabla_\theta f(x) \nabla_\theta f(y) \nabla_{f(y)} \mathscr{L} = -\eta \Theta(x,y) \nabla_{f(y)} \mathscr{L}$$

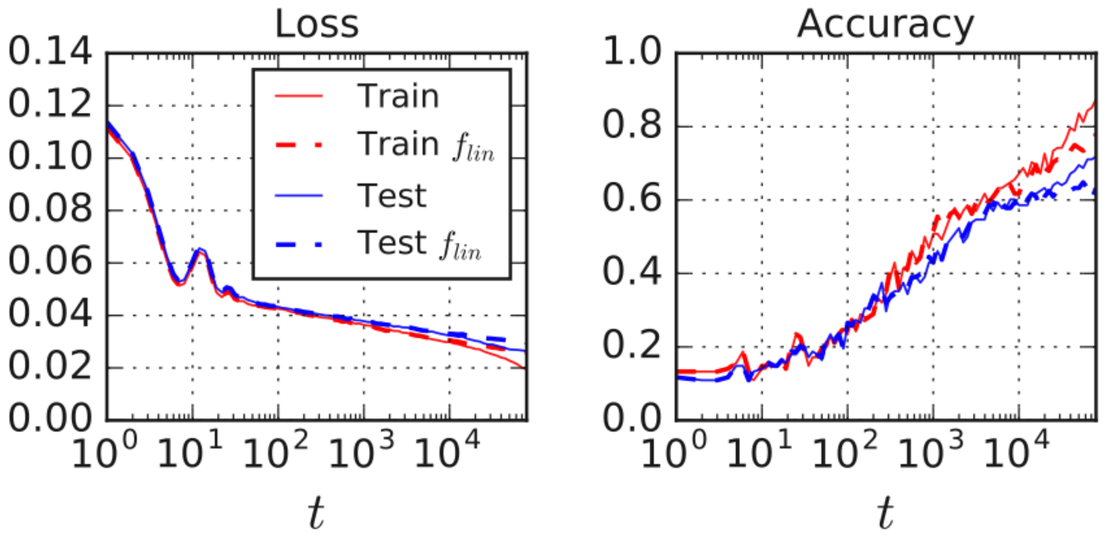
- Assumption: $\Theta(t,x,y) = \Theta(t=0,x,y)$

Jacot, Gabrial, Hongler
Lee, Xiao, Schoenholz, Bahri, Novak, Sohl-Dickstein, Pennington
Novak, Xiao, Hron, Lee, Alemi, Sohl-Dickstein, Schoenholz

Neural networks

A closer look at NTK

- $\Theta(t, x, y) = \Theta(t = 0, x, y)$ can be checked empirically (JAX based neural tangents package)
- A wide range (but clearly not all) of NN dynamics can be described via NTK
- $\dot{f}_{\text{lin}}(x) = -\eta \Theta(t = 0, x, y) \nabla_{f_{\text{lin}}(y)} \mathscr{L}$



Wide reset trained by SGD with momentum on CIFAR-10 (from 1902.06720)

https://github.com/google/neural-tangents

Neural Networks

Gradient descent with momentum

- Modification of optimizer: gradient descent with momentum

$$\theta_{i+1} = \theta_i + v_i \ , \quad v_i = \beta v_{i-1} - \eta \nabla_\theta \mathscr{L}$$

- NN differential equation becomes second order (more familiar from scalar field dynamics)

$$\ddot{f}(x) + \frac{1-\beta}{\sqrt{\eta}} \dot{f}(x) + \Theta(x,y)\mathscr{L}'(f(y)) = 0 \qquad (\Delta t = i\sqrt{\eta})$$

Empirical NTK

- Assumption of no evolving emp. NTK provides us with ODE for NN dynamics:

$$\dot{f}_{\text{lin}}(x) = -\eta\Theta(t=0, x, y)\,\nabla_{f_{\text{lin}}(y)}\mathscr{L}$$

- This ODE can be solved in closed form.
- The evolution depends on the data, the architecture, the loss function, and the optimiser. All components are present and included in this framework.
- In this approximation, we can study their respective influence, e.g.: how does more data or more careful data selection change the training dynamics.
- *My personal interest: Is there a simple phenomenological/physical framework to include non-linear effects (time evolution of the empirical NTK). How do these NN dynamical systems compare with physical systems?*

Effects of data selection on test performance

- To demonstrate such effects, e.g. of different dataset sizes, we can use such analytical frameworks or use ensemble experiments.
- E.g.: comparing two dataset selection methods: Random or Random Network Distillation (strategy to select points which are most distinct for architecture in a given dataset)

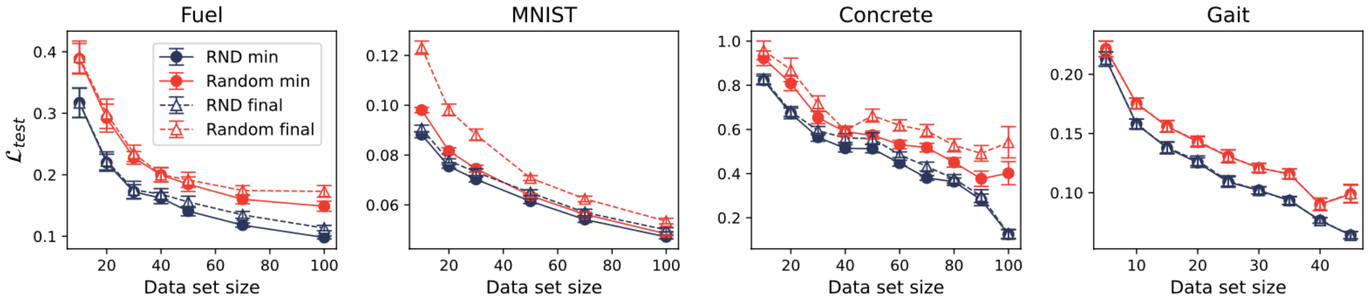


Figure 4. Minimum test loss and final test loss of models trained on data-sets chosen by RND and randomly for several data-set sizes. Size of the error bars corresponds to the ensemble operation over models wherein the experiment was performed 20 times for a single data-set size.

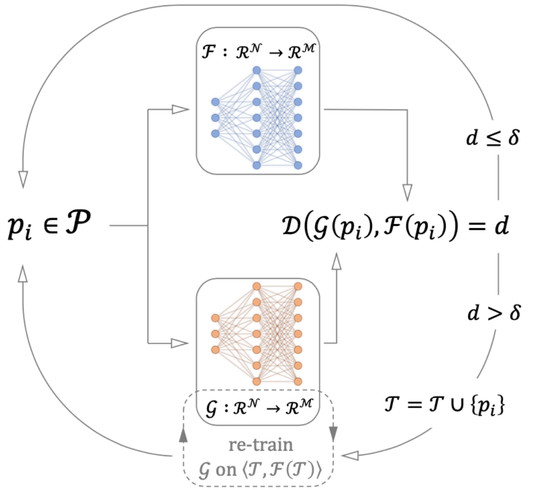


Figure 1. Workflow of RND. A data point, p_i, is passed into the target network, $\mathcal{F}$ and the predictor network $\mathcal{G}$, in order to construct the representations $\mathcal{F}(p_i)$ and $\mathcal{G}(p_i)$. A distance, d is then computed using the metric $\mathcal{D}(\mathcal{F}(p_i), \mathcal{G}(p_i))$. If $d > \delta$, the point, p_i, will be added to the target set $\mathcal{T}$ and the predictor model re-trained on the full set $\mathcal{T}$. If the $d \leq \delta$, it is assumed that a similar point already exists in $\mathcal{T}$ and is therefore discarded. In our notation, $\langle \mathcal{T}, \mathcal{F}(\mathcal{T}) \rangle$ denotes the function set with domain $\mathcal{T}$ and image $\mathcal{F}(\mathcal{T})$.

Conclusions

Looking under the hood of neural networks

- A single perceptron has limited functional capacity (XOR).
- Hidden layers potentially hold meaningful information about physical systems.
- To reveal these methods we can use dimensional reduction and then custom algorithms to reveal the structure (e.g. symmetries).
- Saliency maps to reveal how much features are influenced by other features (e.g. input).
- Neural network dynamics can be analysed empirically and analytically given some assumptions (e.g. empirical NTK being constant during evolution).

Further resources

- physicsmeetsml.org
- Journal: Machine Learning Science and Technology
- MIAPP summer school on machine learning in particle theory
- AIM@LMU: AI as a minor in physics and other subjects at LMU