From Traditional Neural Networks to Deep Learning and Beyond

Vladik Kreinovich

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@utep.edu
http://www.cs.utep.edu/vladik

(Based on joint work with Chitta Baral, also with Olac Fuentes and Francisco Zapata)

1. Why Traditional Neural Networks: (Sanitized) History

- How do we make computers think?
- To make machines that fly it is reasonable to look at the creatures that know how to fly: the birds.
- To make computers think, it is reasonable to analyze how we humans think.
- On the biological level, our brain processes information via special cells called]it neurons.
- Somewhat surprisingly, in the brain, signals are electric – just as in the computer.
- The main difference is that in a neural network, signals are sequence of identical pulses.

2. Why Traditional NN: (Sanitized) History

- The intensity of a signal is described by the frequency of pulses.
- A neuron has many inputs (up to 10^4).
- All the inputs $x_1, \ldots, x_n$ are combined, with some loss, into a frequency $\sum\limits_{i=1}^{n} w_i \cdot x_i$.
- Low inputs do not active the neuron at all, high inputs lead to largest activation.
- The output signal is a non-linear function

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i - w_0\right).$$

- In biological neurons, $f(x) = 1/(1 + \exp(-x))$.
- Traditional neural networks emulate such biological neurons.

3. Why Traditional Neural Networks: Real History

- At first, researchers ignored non-linearity and only used linear neurons.
- They got good results and made many promises.
- The euphoria ended in the 1960s when MIT's Marvin Minsky and Seymour Papert published a book.
- Their main result was that a composition of linear functions is linear (I am not kidding).
- This ended the hopes of original schemes.
- For some time, neural networks became a bad word.
- Then, smart researchers came us with a genius idea: let's make neurons non-linear.
- This revived the field.

4. Traditional Neural Networks: Main Motivation

- One of the main motivations for neural networks was that computers were slow.
- Although human neurons are much slower than CPU, the human processing was often faster.
- So, the main motivation was to make data processing faster.
- The idea was that:
  - since we are the result of billion years of ever improving evolution,
  - our biological mechanics should be optimal (or close to optimal).

5. How the Need for Fast Computation Leads to Traditional Neural Networks

- To make processing faster, we need to have many fast processing units working in parallel.
- The fewer layers, the smaller overall processing time.
- In nature, there are many fast linear processes – e.g., combining electric signals.
- As a result, linear processing (L) is faster than non-linear one.
- For non-linear processing, the more inputs, the longer it takes.
- So, the fastest non-linear processing (NL) units process just one input.
- It turns out that two layers are not enough to approximate any function.

6. Why One or Two Layers Are Not Enough

- With 1 linear (L) layer, we only get linear functions.
- With one nonlinear (NL) layer, we only get functions of one variable.
- With L→NL layers, we get $g\left(\sum_{i=1}^{n} w_i \cdot x_i - w_0\right)$.
- For these functions, the level sets $f(x_1, \ldots, x_n) = \text{const}$ are planes $\sum_{i=1}^{n} w_i \cdot x_i = c$.
- Thus, they cannot approximate, e.g., $f(x_1, x_2) = x_1 \cdot x_2$ for which the level set is a hyperbola.
- For NL→L layers, we get $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_i)$.
- For all these functions, $d \stackrel{\text{def}}{=} \dfrac{\partial^2 f}{\partial x_1 \partial x_2} = 0$, so we also cannot approximate $f(x_1, x_2) = x_1 \cdot x_2$ with $d = 1 \neq 0$.

7. Why Three Layers Are Sufficient: Newton's Prism and Fourier Transform

- In principle, we can have two 3-layer configurations: L→NL→L and NL→L→NL.
- Since L is faster than NL, the fastest is L→NL→L:

$$y = \sum_{k=1}^{K} W_k \cdot f_k\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

- Newton showed that a prism decomposes while light (or any light) into elementary colors.
- In precise terms, elementary colors are sinusoids

$$A \cdot \sin(w \cdot t) + B \cdot \cos(w \cdot t).$$

- Thus, every function can be approximated, with any accuracy, as a linear combination of sinusoids:

$$f(x_1) \approx \sum_{k} (A_k \cdot \sin(w_k \cdot x_1) + B_k \cdot \cos(w_k \cdot x_1)).$$

8. Why Three Layers Are Sufficient (cont-d)

- Newton's prism result:

$$f(x_1) \approx \sum_k (A_k \cdot \sin(w_k \cdot x_1) + B_k \cdot \cos(w_k \cdot x_1)).$$

- This result was theoretically proven later by Fourier.
- For $f(x_1, x_2)$, we get a similar expression for each x_2, with $A_k(x_2)$ and $B_k(x_2)$.
- We can similarly represent $A_k(x_2)$ and $B_k(x_2)$, thus getting products of sines, and it is known that, e.g.:

$$\cos(a) \cdot \cos(b) = \frac{1}{2} \cdot (\cos(a+b) + \cos(a-b)).$$

- Thus, we get an approximation of the desired form with $f_k = \sin$ or $f_k = \cos$:

$$y = \sum_{k=1}^{K} W_k \cdot f_k \left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0} \right).$$

9. Which Activation Functions $f_k(z)$ Should We Choose

- A general 3-layer NN has the form:

$$y = \sum_{k=1}^{K} W_k \cdot f_k\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

- Biological neurons use $f(z) = 1/(1 + \exp(-z))$, but shall we simulate it?
- Simulations are not always efficient.
- E.g., airplanes have wings like birds but they do not flap them.
- Let us analyze this problem theoretically.
- There is always some noise c in the communication channel.
- So, we can consider either the original signals x_i or denoised ones $x_i - c$.

10. Which $f_k(z)$ Should We Choose (cont-d)

- The results should not change if we perform a full or partial denoising $z \to z' = z - c$.
- Denoising means replacing $y = f(z)$ with $y' = f(z-c)$.
- So, $f(z)$ should not change under shift $z \to z - c$.
- Of course, $f(z)$ cannot remain the same: if $f(z) = f(z - c)$ for all c, then $f(z) = \text{const}$.
- The idea is that once we re-scale x, we should get the same formula after we apply a natural y-re-scaling T_c:

$$f(x - c) = T_c(f(x)).$$

- Linear re-scalings are natural: they corresponding to changing units and starting points (like C to F).

11. Which Transformations Are Natural?

- An inverse T_c^{-1} to a natural re-scaling T_c should also be natural.
- A composition $y \to T_c(T_{c'}(y))$ of two natural re-scalings T_c and $T_{c'}$ should also be natural.
- In mathematical terms, natural re-scalings form a *group*.
- For practical purposes, we should only consider re-scaling determined by finitely many parameters.
- So, we look for a finite-parametric group containing all linear transformations.

12. A Somewhat Unexpected Approach

- N. Wiener, in *Cybernetics*, notices that when we approach an object, we have distinct phases:
  - first, we see a blob (the image is invariant under all transformations);
  - then, we start distinguishing angles from smooth but not sizes (projective transformations);
  - after that, we detect parallel lines (affine transformations);
  - then, we detect relative sizes (similarities);
  - finally, we see the exact shapes and sizes.
- Are there other transformation groups?
- Wiener argued: if there are other groups, after billions years of evolutions, we would use them.
- So he conjectured that there are no other groups.

13. Wiener Was Right

- Wiener's conjecture was indeed proven in the 1960s.
- In 1-D case, this means that all our transformations are fractionally linear:
$$f(z-c)=\frac{A(c)\cdot f(z)+B(c)}{C(c)\cdot f(z)+D(c)}.$$
- For $c=0$, we get $A(0)=D(0)=1$, $B(0)=C(0)=0$.
- Differentiating the above equation by c and taking $c=0$, we get a differential equation for $f(z)$:
$$-\frac{df}{dz}=(A'(0)\cdot f(z)+B'(0))-f(z)\cdot(C'(0)\cdot f(z)+D'(0)).$$
- So, $\dfrac{df}{C'(0)\cdot f^2+(A'(0)-C'(0))\cdot f+B'(0)}=-dz.$
- Integrating, we indeed get $f(z)=1/(1+\exp(-z))$ (after an appropriate linear re-scaling of z and $f(z)$).

14. How to Train Traditional Neural Networks: Main Idea

- *Reminder:* a 3-layer neural network has the form:

$$y = \sum_{k=1}^{K} W_k \cdot f\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

- We need to find the weights that best described observations $\left(x_1^{(p)}, \ldots, x_n^{(p)}, y^{(p)}\right)$, $1 \le p \le P$.
- We find the weights that minimize the mean square approximation error $E \stackrel{\text{def}}{=} \sum_{p=1}^{P} \left(y^{(p)} - y_{NN}^{(p)}\right)^2$, where

$$y^{(p)} = \sum_{k=1}^{K} W_k \cdot f\left(\sum_{i=1}^{n} w_{ki} \cdot x_i^{(p)} - w_{k0}\right) - W_0.$$

- The simplest minimization algorithm is gradient descent: $w_i \to w_i - \lambda \cdot \dfrac{\partial E}{\partial w_i}$.

15. Towards Faster Differentiation

- To achieve high accuracy, we need many neurons.
- Thus, we need to find many weights.
- To apply gradient descent, we need to compute all partial derivatives $\dfrac{\partial E}{\partial w_i}$.
- Differentiating a function f is easy:
  - the expression f is a sequence of elementary steps,
  - so we take into account that $(f \pm g)' = f' \pm g'$, $(f \cdot g)' = f' \cdot g + f \cdot g'$, $(f(g))' = f'(g) \cdot g'$, etc.
- For a function that takes T steps to compute, computing f' thus takes $c_0 \cdot T$ steps, with $c_0 \le 3$.
- However, for a function of n variables, we need to compute n derivatives.
- This would take time $n \cdot c_0 \cdot T \gg T$: this is too long.

16. Faster Differentiation: Backpropagation

- Idea:
  - instead of starting from the variables,
  - start from the last step, and compute $\dfrac{\partial E}{\partial v}$ for all intermediate results v.
- For example, if the very last step is $E = a \cdot b$, then $\dfrac{\partial E}{\partial a} = b$ and $\dfrac{\partial E}{\partial b} = a$.
- At each step y, if we know $\dfrac{\partial E}{\partial v}$ and $v = a \cdot b$, then $\dfrac{\partial E}{\partial a} = \dfrac{\partial E}{\partial v} \cdot b$ and $\dfrac{\partial E}{\partial b} = \dfrac{\partial E}{\partial v} \cdot a$.
- At the end, we get all n derivatives $\dfrac{\partial E}{\partial w_i}$ in time

$$c_0 \cdot T \ll c_0 \cdot T \cdot n.$$

- This is known as *backpropagation*.

17. Beyond Traditional NN

- Nowadays, computer speed is no longer a big problem.
- What *is* a problem is *accuracy:* even after thousands of iterations, the NNs do not learn well.
- So, instead of computation speed, we would like to maximize learning accuracy.
- We can still consider L and NL elements.
- For the same number of variables w_i, we want to get more accurate approximations.
- For given number of variables, and given accuracy, we get N possible combinations.
- If all combinations correspond to different functions, we can implement N functions.
- However, if some combinations lead to the same function, we implement fewer different functions.

18. From Traditional NN to Deep Learning

- For a traditional NN with K neurons, each of $K!$ permutations of neurons retains the resulting function.
- Thus, instead of N functions, we only implement

$$\frac{N}{K!} \ll N \text{ functions.}$$

- Thus, to increase accuracy, we need to minimize the number K of neurons in each layer.
- To get a good accuracy, we need many parameters, thus many neurons.
- Since each layer is small, we thus need many layers.
- This is the *main idea* behind *deep learning*.
- *Another idea:* replace not-very-efficient gradient descent with more efficient optimization techniques.

19. Need to Go Beyond Deep Learning

- All this emulates only learning from examples.
- We humans also learn by explicitly learning formulas and rules.
- How can we incorporate known formulas and rules into deep learning techniques?
- First case: we have exact constraints

$$g_\ell(y_1, \ldots, y_m) = 0, \quad 1 \le \ell \le L.$$

- In this case, we can first train the NN off-line to learn the constraints.
- Then, we minimize the least squares $E \stackrel{\text{def}}{=} \sum_{j=1}^{m} \sum_{p=1}^{P} \left(y_j^{(p)} - y_{j,NN}^{(p)}\right)^2$ under constraints

$$g_\ell(y_1, \ldots, y_m) = 0.$$

20. Beyond Deep Learning: Case of Exact Constraints

- We minimize $E = \sum_{j=1}^{m} \sum_{p=1}^{P} \left(y_j^{(p)} - y_{j,NN}^{(p)}\right)^2$ under constraints $g_\ell(y_1, \ldots, y_m) = 0$.
- Lagrange multiplier method reduces this to unconstrained minimization of

$$E' \stackrel{\text{def}}{=} E + \sum_{\ell=1}^{L} g_\ell(y_1, \ldots, y_m).$$

- This can be done by a similar (e.g., backpropagation) technique.
- The Lagrange multiplier λ_ℓ can be then adjusted so as to satisfy the constraints.
- This was the gist of our 2016 paper.

21. Constraints Are Usually Approximate

- In practice, constraints are usually only approximate.
- How can we take this into account?
- We know that $y \approx f(x, a)$ for some parameters

$$a = (a_1, a_2, \ldots).$$

- For example, we may know that the dependence of y on x is approximately linear: $y \approx a_1 + a_2 \cdot x$.
- In this case, when we have the observations $\left(x^{(p)}, y^{(p)}\right)$, practitioners usually use two options.
- The first option is to simply find the values a for which $y^{(p)}$ is the closest to the model: $y^{(p)} \approx f\left(x^{(p)}, a\right)$.
- In other words, we find the values a for which $\sum\limits_{p} \left(y^{(p)} - f\left(x^{(p)}, a\right)\right)^2$ is the smallest possible.

22. How Approximate Constraints Are Handled

- The second option is to use a NN.
- In this case, we find the weights w for which the output $f_{NN}\left(x^{(p)}, w\right)$ of the NN is closest to $y^{(p)}$.
- In other words, we find $\sum\limits_p \left(y^{(p)} - f_{NN}\left(x^{(p)}, w\right)\right)^2$ is the smallest possible.
- The problem with the first approach is that the model $f(x, a)$ is crude and approximate.
- We want more accurate predictions.
- The problem with the second approach is that:
  - since we do not use the known model,
  - learning is slower than it should be.

23. Seemingly Natural Idea and Its Limitations

- It is therefore reasonable to simultaneously look for a and for w for which

$$y^{(p)} \approx f\left(x^{(p)}, a\right) \text{ and } y^{(p)} \approx f\left(x^{(p)}, a\right).$$

- A seemingly natural idea is to apply least squares and minimize the sum

$$\sum_p \left(y^{(p)} - f\left(x^{(p)}, a\right)\right)^2 + \sum_p \left(y^{(p)} - f_{NN}\left(x^{(p)}, w\right)\right)^2.$$

- *Problem:* we get two two independent optimization problems: of finding a and of finding w.
- So, we do not gain anything.

24. Carnegie-Mellon Idea

- In 2016, two papers by Carnegie Mellon researchers proposed a solution:
  - since $y^{(p)} \approx f\left(x^{(p)}, a\right)$ and $y^{(p)} \approx f_{NN}\left(x^{(p)}, w\right)$,
  - we can conclude that $f\left(x^{(p)}, a\right) \approx f_{NN}\left(x^{(p)}, w\right)$.
- Thus, it is reasonable to find a and w for which

$$y^{(p)} \approx f\left(x^{(p)}, a\right), \quad y^{(p)} \approx f_{NN}\left(x^{(p)}, w\right), \quad \text{and}$$

$$f\left(x^{(p)}, a\right) \approx f_{NN}\left(x^{(p)}, w\right).$$

- In other words, we minimize the triple sum

$$\sum_p \left(y^{(p)} - f\left(x^{(p)}, a\right)\right)^2 + \sum_p \left(y^{(p)} - f_{NN}\left(x^{(p)}, w\right)\right)^2 +$$

$$\sum_p \left(f\left(x^{(p)}, a\right) - f_{NN}\left(x^{(p)}, w\right)\right)^2.$$

25. Carnegie-Mellon Idea (cont-d)

- We can solve this optimization problem if we:
  - first fix w and minimize by a,
  - then fix a and minimize by w, etc.
- When we look for a, we no longer look for values for which $f\left(x^{(p)}, a\right) \approx y^{(p)}$.
- Instead, we look for values a for which

$$f\left(x^{(p)}, a\right) \approx \frac{y^{(p)} + f_{NN}\left(x^{(p)}, w\right)}{2}.$$

- Here, w is what we have so far.
- Similarly, when we look for values w for which

$$f_{NN}\left(x^{(p)}, w\right) \approx \frac{y^{(p)} + f\left(x^{(p)}, a\right)}{2}.$$

- Here, a is what we have so far in parameter estimation.

26. Carnegie-Mellon Idea: Can We Do Better?

- The Carnegie-Mellon idea enables us:
  - to guide NN in the direction of the model,
  - and at the same time avoid exact fit with the model.
- Can we do better?
- The above description assumed that NN and model have equal accuracy.
- In reality, NN usually has higher accuracy.
- Then, instead of equal weights, we will have:
  - smaller weight for the model and
  - higher weight for the data $y^{(p)}$.

27. New Idea: Details

- If we have $y^{(p)} \approx f\left(x^{(p)}, a\right)$ with accuracy $\sigma_{\rm model}$ and $y^{(p)} \approx f_{NN}\left(x^{(p)}, w\right)$ with accuracy $\sigma_{\rm meas}$, then

$$f\left(x^{(p)}, a\right) \approx f_{NN}\left(x^{(p)}, w\right) \text{ with accuracy } \sqrt{\sigma^2_{\rm model} + \sigma^2_{\rm meas}}.$$

- Thus, we minimize the sum

$$\sum_p \frac{\left(y^{(p)} - f\left(x^{(p)}, a\right)\right)^2}{\sigma^2_{\rm model}} + \sum_p \frac{\left(y^{(p)} - f_{NN}\left(x^{(p)}, w\right)\right)^2}{\sigma^2_{\rm meas}} +$$

$$\sum_p \frac{\left(f\left(x^{(p)}, a\right) - f_{NN}\left(x^{(p)}, w\right)\right)^2}{\sigma^2_{\rm model} + \sigma^2_{\rm meas}}.$$

28. New Idea: Details (cont-d)

- *Reminder:* we minimize the sum

$$\sum_p \frac{\left(y^{(p)} - f\left(x^{(p)}, a\right)\right)^2}{\sigma^2_{\rm model}} + \sum_p \frac{\left(y^{(p)} - f_{NN}\left(x^{(p)}, w\right)\right)^2}{\sigma^2_{\rm meas}} +$$

$$\sum_p \frac{\left(f\left(x^{(p)}, a\right) - f_{NN}\left(x^{(p)}, w\right)\right)^2}{\sigma^2_{\rm model} + \sigma^2_{\rm meas}}.$$

- Now, we look for a for which:

$$f\left(x^{(p)}, a\right) \approx \frac{(1+z)\cdot y^{(p)} + f_{NN}\left(x^{(p)}, w\right)}{2+z}, \text{ where } z \stackrel{\rm def}{=} \frac{\sigma^2_{\rm meas}}{\sigma^2_{\rm model}}.$$

- Similarly, we look for for w for which

$$f_{NN}\left(x^{(p)}, w\right) \approx \frac{(1+z)\cdot y^{(p)} + z\cdot f\left(x^{(p)}, a\right)}{1+2z}.$$

29. Bibliography: Which Activation Function to Choose?

- V. Kreinovich and C. Quintana. "Neural networks: what non- linearity to choose?," *Proceedings of the 4th University of New Brunswick Artificial Intelligence Workshop*, Fredericton, New Brunswick Canada, 1991, pp. 627–637.
- O. Sirisaengtaksin, V. Kreinovich, and H. T. Nguyen, "Sigmoid neurons are the safest against additive errors", *Proceedings of the First International Conference on Neural, Parallel, and Scientific Computations*, Atlanta, GA, May 28–31, 1995, Vol. 1, pp. 419–423.
- H. T. Nguyen and V. Kreinovich, *Applications of Continuous Mathematics to Computer Science*, Kluwer, Dordrecht, Netherlands, 1997.

30. Bibliography: Why Deep Learning?

- P. C. Kainen, V. Kurkova, V. Kreinovich, and O. Sirisaengtaksin. "Uniqueness of network parameterization and faster learning", *Neural, Parallel, and Scientific Computations*, 1994, Vol. 2, pp. 459–466.
- C. Baral, O. Fuentes, and V. Kreinovich, "Why deep neural networks: a possible theoretical explanation", In: M. Ceberio et al. (eds.), *Constraint Programming and Decision Making: Theory and Applications*, Springer Verlag, 2018, pp. 1–6. http://www.cs.utep.edu/vladik/2015/tr15-55.pdf

31. Bibliography: Beyond Deep Learning

- C. Baral, M. Ceberio, and V. Kreinovich, "How neural networks (NN) can (hopefully) learn faster by taking into account known constraints", *Proc. 9th Int'l Workshop on Constraints Programming and Decision Making CoProd'2016*, Uppsala, Sweden, Sept. 25, 2016. http://www.cs.utep.edu/vladik/2016/tr16-46.pdf
- Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. P. Xinh, "Harnessing deep neural networks with logic rules", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berkin, Germany, August 7–12, 2016, pp. 2410–2420.
- Z. Hu, Z. Yang, R. Salahutdinov, and E. P. Xing, "Deep neural networks with massive learned knowledge", *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing EMNLP'16*, Austin, Texas, November 2–4, 2016.

32. Appendix: Why Fractional Linear

- Every transformation is a composition of infinitesimal ones $x \to x + \varepsilon \cdot f(x)$, for infinitely small ε.
- So, it's enough to consider infinitesimal transformations.
- The class of the corresponding functions $f(x)$ is known as a *Lie algebra* A of the corresponding transformation group.
- Infinitesimal linear transformations correspond to $f(x) = a + b \cdot x$, so all linear functions are in A.
- In particular, $1 \in A$ and $x \in A$.
- For any λ, the product $\varepsilon \cdot \lambda$ is also infinitesimal, so we get $x \to x + (\varepsilon \cdot \lambda) \cdot f(x) = x \to x + \varepsilon \cdot (\lambda \cdot f(x))$.
- So, if $f(x) \in A$, then $\lambda \cdot f(x) \in A$.

33. Why Fractional Linear (cont-d)

- If we first apply $f(x)$, then $g(x)$, we get

  $x \to (x+\varepsilon\cdot f(x))+\varepsilon\cdot g(x+\varepsilon\cdot f(x)) = x+\varepsilon\cdot(f(x)+g(x))+o(\varepsilon).$

- Thus, if $f(x) \in A$ and $g(x) \in A$, then $f(x)+g(x) \in A$.
- So, A is a linear space.
- In general, for the composition, we get

$$x \to (x + \varepsilon_1 \cdot f(x)) + \varepsilon_2 \cdot g(x_1 + \varepsilon_1 \cdot f(x)) =$$

  $x+\varepsilon_1\cdot f(x)+\varepsilon_2\cdot g(x)+\varepsilon_1\cdot\varepsilon_2\cdot g'(x)\cdot f(x)+$ quadratic terms.

- If we then apply the inverses to $x \to x + \varepsilon_1 \cdot f(x)$ and $x \to x + \varepsilon_2 \cdot g(x)$, the linear terms disappear, we get:

$$x \to x+\varepsilon_1\cdot\varepsilon_2\cdot\{f,g\}(x),\ \text{where } \{f,g\} \stackrel{\text{def}}{=} f'(x)\cdot g(x)-f(x)\cdot g'(x).$$

- Thus, if $f(x) \in A$ and $g(x) \in A$, then $\{f,g\}(x) \in A$.
- The expression $\{f,g\}$ is known as the *Poisson bracket*.

34. Why Fractional Linear (cont-d)

- Let's expand any function $f(x)$ in Taylor series:

$$f(x) = a_0 + a_1 \cdot x + \ldots$$

- If k is the first non-zero term in this expansion, we get

$$f(x) = a_k \cdot x^k + a_{k+1} \cdot x^{k+1} + a_{k+2} \cdot x^{k+2} + \ldots$$

- For every λ, the algebra A also contains

$$\lambda^{-k} \cdot f(\lambda \cdot x) = a_k \cdot x^k + \lambda \cdot a_{k+1} \cot x^{k+1} + \lambda^2 \cdot a_{k+2} \cdot x^{k+2} + \ldots$$

- In the limit $\lambda \to 0$, we get $a_k \cdot x^k \in A$, hence $x^k \in A$.
- Thus, $f(x) - a_k \cdot x^k = a_{k+1} \cdot x^{k+1} + \ldots \in A$.
- We can similarly conclude that A contains all the terms x^n for which $a_n \neq 0$ in the original Taylor expansion.

35. Why Fractional Linear (cont-d)

- Since $g(x) = 1 \in A$, for each $f \in A$, we have
$$\{f, 1\} = f'(x) \cdot 1 + f(x) \cdot g' = f'(x) \in A.$$
- Thus, for each k, if $x^k \in A$, we have $(x^k)' = k \cdot x^{k-1} \in A$ hence $x^{k-1} \in A$, etc.
- Thus, if $x^k \in A$, all smaller power are in A too.
- In particular, this means that if $x^k \in A$ for some $k \geq 3$, then we have $x^3 \in A$ and $x^2 \in A$; thus:
$$\{x^3, x^2\} = (x^3)' \cdot x^2 - x^3 \cdot (x^2)' = 3 \cdot x^2 \cdot x^2 - x^3 \cdot 2 \cdot x = x^4 \in A.$$
- In general, once $x^k \in A$ for $k \geq 3$, we get
$$\{x^k, x^2\} = (x^k)' \cdot x^2 - x^k \cdot (x^2)' = k \cdot x^{k-1} \cdot x^2 - x^k \cdot 2 \cdot x =$$
$$(k-2) \cdot x^{k+1} \in A \text{ hence } x^{k+1} \in A.$$
- So, by induction, $x^k \in A$ for all k.

36. Why Fractional Linear (cont-d)

- If $x^k \in A$ for some $k \geq 3$, then $x^k \in A$ for all k.
- Thus, A is infinite-dimensional – which contradicts to our assumption that A is finite-dimensional.
- So, we cannot have Taylor terms of power $k \geq 3$; therefore we have:

$$x \to x + \varepsilon \cdot (a_0 + a_1 \cdot x + a_2 \cdot x^2).$$

- This corresponds to an infinitesimal fractional-linear transformation

$$x \to \frac{\varepsilon \cdot A + (1 + \varepsilon \cdot B) \cdot x}{1 + \varepsilon \cdot D \cdot x} =$$

$$(\varepsilon \cdot A + (1 + \varepsilon \cdot B) \cdot x) \cdot (1 - \varepsilon \cdot D \cdot x) + o(\varepsilon) =$$

$$x + \varepsilon \cdot (A + (B - D) \cdot x - D \cdot x^2).$$

- So, to match, we need

$$A = a_0, \quad D = -a_2, \text{ and } B = a_1 - a_2.$$

37. Why Fractional Linear: Final Part

- We concluded that every infinitesimal transformation is fractionally linear.
- Every transformation is a composition of infinitesimal ones.
- Composition of fractional-linear transformations is fractional linear.
- Thus, all transformations are fractional linear.