A Quick Introduction to Machine Learning

18.04.2023 Thomas Fischbacher (tfish@google.com), Google Research

Scope and Disclaimer

- Speaking in personal capacity here:
  views and opinions are my own, not necessarily my employer's.

- All material is assembled from earlier externally-approved external talks and courses.

- This talk is for: people with a technical background who have not touched ML before.

My "ML and Physics" Co-Authors at Google Research

Iulia-Maria Comșa, Moritz Firsching, Thomas Fischbacher, Luciano Sbaiz




Machine Learning - What is it?

- This talk: "pre-2017 ML"
- In December 2017: "Attention Is All You Need"
  (A. Vaswani et al., Google Brain/Google Reseach/U Toronto):
  "Transformer architecture".
- Breakthrough paper that enabled many of the things
  everybody is talking about today.
- Bit hard to explain without first discussing more basic ML.

Machine Learning - What is it?

Philosophical Aspects - "What Are We Doing?"

Machine Learning - What is it?

Distinction: "Analytic(/Algorithmic) approach" vs. "Machine Learning approach"

Analytic Approach (e.g.: Dijkstra, "On the role of scientific thought")

- Decomposition of a complex problem
- Can study various aspects (mostly) in isolation
- "effective ordering of one's thoughts"
- Synthesizing solutions from pieces, solid reasoning over entire construction
- "Unreasonably Effective"

Machine Learning - What is it?

Eddington's comment on "the unreasonable effectiveness of mathematics" (Wigner):

"Some men went fishing in the sea with a net, and upon examining what they caught they concluded that there was a minimum size to the fish in the sea."

- "Maths is a fishing net".
- Will not allow us to catch some fish!
  Example: "Do these 1000x1000 pixels show a shark?" -
  This is hard to write down as a $R^{1000000} \rightarrow R$ function!
- "ML is a different fishing net".

Machine Learning - What is it?

"*ML is what we use when we have to concede defeat on the algorithmic front*"

- Situation: Cannot (yet?) formulate a proper compact algorithm to tackle a problem.

- Stepping back: "Are we asking the right problem?"
  - Is a probably-approximately-correct answer useful?
  - Do we insist on writing down how-to-get-the-answer in human-understandable, verifiable, justifiable form?

Machine Learning - What is it?

("Supervised") Machine Learning approach: "Generalizing from Examples"

Start from a set of "training examples" for which answers are known.

- Set up a parametric function that maps input to output of about the right form.
- Randomly initialize parameters. Initially, outputs will be typically wildly off.
- Use parameter-tuning to improve performance on the training set.
- Evaluation: see if the system performs well on never-seen-before examples drawn from the same distribution as the training set.
- "**probably approximately correct**": The answer for a previously-unseen example is observed to be off by no more than δ with "probability" $\geq p$.

Machine Learning - What is it?

"ML as a kind of telescope"

Being able to quickly classify thousands of examples is often useful - but there is more for us here

- Science uses sense-enhancing technology.
- (Can be as humble as a thermometer or a properly calibrated analytical balance).
- Telescopes allow us to see *faraway* structure.
- ML allows us to see *high-dimensional* structure.
- Useful e.g. for ANOVA-style analysis:
  How well can we predict {target} from {subset of features}?
- Often: Using ML can help us to understand a problem better - up to the point where we can tackle it without a need for ML.

Machine Learning - What is it?

Supervised ML - Generic Aspects

- At the core, there always will be some $R^P \rightarrow R^Q$ function f_θ that depends on (typically: many!) parameters θ.
- The training process fine-tunes parameters θ to improve performance on the training examples.
- Generalization beyond the training set is "wishful thinking" - but usually works surprisingly well.
- Parts of the problem:
  - Representing real-world data (e.g.: images, text, sound, ...) in vector form.
  - Finding some proxy function that quantifies (smoothly) how badly off the output is w.r.t. expected answer.
  - Making an informed choice about the inner structure of f_θ (the "ML model architecture").
  - Iteratively estimating (on the training set) how much performance improves by performing specific parameter tuning - and applying updates.

Machine Learning - What is it?

Useful Baseline Mental Models

- *For this form of ML* - useful to understand behavior of a blunt "baseline":
  - K-means classifier that can memorize all training examples (each a high-dimensional vector).
  - "Infinite training set size limit" (practically very much unattainable - theoretical tool).
  - Has access to a dense point-cloud in feature-space, averages nearest-neighbor answers.
  - Answers: "what is learnable in principle from this data?"

- Linear/logistic regression: Linear function on feature-space, potentially mapping "evidence" to "probability" via sigmoid function: debuggable/understandable model.

Machine Learning - What is it?

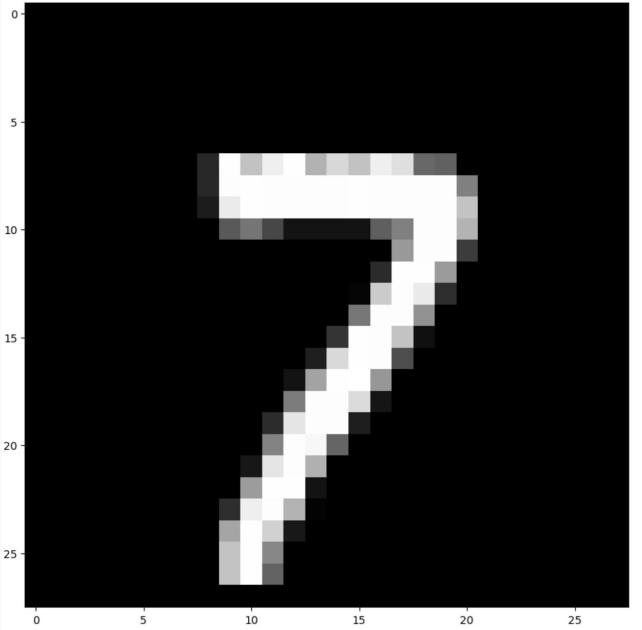
Every discipline has its "fruit fly" / "hydrogen atom" test problem.

For supervised ML, this (arguably? still?) is MNIST: Handwritten digit classification, using the labeled dataset provided by NIST.

Overall, "limited utility" since this ML problem happens to be "too simple" - "everything works on MNIST".

Still useful for:

- Studying exotic architectures
- "Have I wired up everything correctly?" testing



Machine Learning - What is it?

The "Deep Learning Revolution"

- (Difficult to give proper credit to every pioneer in short time - e.g. back story of 2017 "transformers" reaches back to 1992 (Schmidhuber).)
- Pre-2006: Variety of different architectures, emphasis on "feature engineering", i.e. pre-processing input data, specialized "what-to-use-where" knowledge.
  "No one can train NNs for which more than 3 layers bring a benefit".
- Post-2006: "Deep Learning" revolution, started by Hinton's "A Fast Learning Algorithm for Deep Belief Nets" - renaissance of "Deep Neural Networks", shift from other architectures to NNs.
- Other designs (Random Forests, SVMs, etc.) still are relevant for some applications.
- Interest in DNNs uncovered a few earlier mistakes that impeded progress - all rectified by now. In particular: Hinton's original pre-training procedure plays little role these days.

Machine Learning - What is it?

- Very basic DNN ML model architecture:
  - Organized in "layers". First is "input layer", last is "output layer", in-between are "hidden layers".
  - Every "layer" receives data from the previous layer, plus maybe even earlier ("towards the input") layers.
  - Layers apply some simple-to-compute non-linear transform to their input data. Simple yet powerful architectural choice:

$$x^{\{\text{Layer } k+1\}}{}_i = G(W^k{}_{ij} x^{\{\text{Layer } k\}}{}_j + B^k{}_i), \quad R^1 \to R^1 \text{ non-linearity } G \text{ e.g. ReLU: } x \mapsto (x + |x|/2)$$

  - Parameters: Weights $W^k{}_{ij}$, Biases $B^k{}_i$.
  - Most important: "Model's capacity for learning". Details (e.g. choice of non-linearity, layers-as-components) moderately relevant. "Can learn with most nonlinear designs".

Machine Learning - Why should we care?

Even if we keep focused on the effort to understand the world via analytic reasoning, ML is still interesting in multiple ways:

- Can eliminate tedious work (such as: manual image-tagging, data pre-screening).
- Can be used to generate ideas ("observation: we can make a pretty good guess about Z from X even without knowing Y which we thought to be essential").
- The technology that has been built to power the DL Revolution can be used straight away for other challenging Non-DL tasks!

Machine Learning - Why should we care?

Performance-improvement via parameter-tuning = some numerical minimization problem ("minimization of loss"), but a somewhat quirky one:

- Very many optimization parameters (~10^4-10^{12})!
- (Behavior of numerical optimization problems "changes flavor" at around 5000 parameters.)
- *We are not actually interested in finding the minimum!*
- Instead: What matters is the trained model's ability to generalize beyond the training set - evaluating this does not involve the "badness" (/ "loss") function we use in training.
- Gradient-based minimization ("1st order method"), but numerically estimated gradients will very often do: Pick random sample of ~30 examples, compute average badness as a function of model-parameters, take gradient, change parameters by taking a small step "downward" along opposite-of-gradient direction.
- (Personal educational problem: talking people out of using ML optimization approaches when using ML tech for conventional numerical optimization with <5000 parameters.)

Machine Learning - Why should we care?

ML Tech for Numerics

- Even "conventional" DL ML often uses some mildly unusual mathematical constructions.
- Good support for numerical multilinear algebra & basic nonlinear operations.
- Excellent support for computing fast gradients.
- Allows "wiring up" a calculation in a high level programming language (e.g. Python), but actually running the computation as fast (possibly distributed) compiled code on: CPU, GPU, TPU.
- "**Every physicist nowadays should know how to solve a non-malicious 1000-parameter optimization problem numerically**" - ML infrastructure makes this *incredibly* easy.

Machine Learning - Why should we care?

Fast Gradients - How do they Work? ("Sensititity Backpropagation" / RM-AD)

If we are given some function f: $R^P \rightarrow R$ and another function g: $R \rightarrow R^Q$, both in the form of computer code, then:

- There is a program transformation "Code[f] → Code[∇f]" that translates the code that evaluates f into code that also computes f's gradient at the evaluation-point, which needs no more than 8× the computational effort, *independent of P*! ("RM-AD").
- There is a program transformation "Code[g] → Code[∇g]" that translates the code that evaluates g into code that also computes g's gradient at the evaluation-point, which needs no more than 8× the computational effort, *independent of Q*! ("FM-AD").

Machine Learning - Why should we care?

FM-AD vs. RM-AD ("Forward/Reverse-Mode Automatic Differentiation")

- FM-AD is equivalent to "Gaussian Error Propagation": Can go through the calculation in a way where we replace real-valued intermediate quantities with "dual numbers", schematically: $Z+c_1 dx+c_2 dy$ with "$dx^2=0$", "$dy^2=0$", i.e. perturbatively retaining corrections to inputs "to 1st order".
- RM-AD is slightly more involved:
  - Need to perform the "forward-"computation of the end result, *but in a way that remembers each intermediate quantity computed, and each branching-decision taken in the code*.
  - Once result is known, can proceed to the last-computed intermediate quantity and ask: "by how much would the end result change (relative to ε) if I interrupted the forward-calculation right after obtaining this quantity and tweaked it by ε?
  - ...

Machine Learning - Why should we care?

FM-AD vs. RM-AD ("Forward/Reverse-Mode Automatic Differentiation")

- RM-AD is slightly more involved:
  - Need to perform the "forward-"computation of the end result, *but in a way that remembers each intermediate quantity computed, and each branching-decision taken in the code*.
  - Once result is known, can proceed to the last-computed intermediate quantity and ask: "by how much would the end result change (relative to ε) if I interrupted the forward-calculation right after obtaining this quantity and tweaked it by ε?
  - Once we know this "sensitivity of the end result on the last intermediate quantity", we can proceed to the next-last intermediate quantity...
  - ...all the way back to the input parameters. Sensitivity of end result on input parameters = gradient.
  - Procedure is simplified by using "sensitivity-accumulators".
  - (Still useful to master performing such program transforms also by-hand).

Sensitivity Backpropagation - Example

```python
def surface4(a, b, c, d):
  """Hypersurface of a 4d box with sides a, b, c, d."""
  ab = a * b
  cd = c * d
  abc = ab * c
  abd = ab * d
  acd = a * cd
  bcd = b * cd
  half_area = abc + abd + acd + bcd
  ret = 2 * half_area
  return ret
```

```python
def grad_surface4(a, b, c, d):
  """Gradient of surface4()."""
  sa = sb = sc = sd = 0  # Sensitivity of result on a,b,c,d.
  ab = a * b; sab = 0
  cd = c * d; scd = 0
  abc = ab * c; sabc = 0
  abd = ab * d; sabd = 0
  acd = a * cd; sacd = 0
  bcd = b * cd; sbcd = 0
  half_area = abc + abd + acd + bcd
  ret = 2 * half_area
  # return ret
  s_ret = 1  # Sensitivity of the result on the result is 1.
  # Forward: ret = 2 * half_area
  s_half_area = 2 * s_ret
  # Forward: half_area = abc + abd + acd + bcd
  sabc += s_half_area
  sabd += s_half_area
  sacd += s_half_area
  sbcd += s_half_area
  # Forward: bcd = b * cd
  sb += cd * sbcd; scd += b * sbcd
  # Forward: acd = a * cd
  sa += cd * sacd; scd += a * sacd
  # Forward: abd = ab * d; abc = ab * c
  # ...
  # Forward: cd = c * d
  sc += d * scd; sd += c * scd
  # Forward: ab = a * b
  sa += b * sab; sb += a * sab
  return (sa, sb, sc, sd)  # Gradient.
```

Sensitivity Backpropagation in Context

- Sensitivity-Backpropagation belongs to a cluster of ideas that branch out from "Subproblem Optimality".
- If we did RM-AD on a naive ODE-integrator, we would readily discover the "adjoint method" for pulling gradients through ODE-evolution.
- Other closely related ideas include:
  - Dynamic Programming
  - Control Hamiltonians
  - Hamilton-Jacobi Mechanics
  - Classical Optics
  - Wave Optics
  - Path Integrals
  - Schrödinger's Equation
  - Schwinger's Proper Time Formalism
  - "Worldline Numerics"
  - (stretch) Bosonic String Theory

Where to learn more?

For math/physics professionals: can recommend some resources I am myself deeply familiar with:

- Author's 2022 IMPRS course "Machine Learning in Theoretical Physics"
  - Covers: Python basics, basic ML models, ML overview, using ML tech for doing physics
  - 4 days of recorded lectures + colab notebooks;
    includes "raytracing a black hole with ML tech".
- Google's (TensorFlow-based) web Machine Learning Crash Course
  - About 2 days of work.
  - Used at Google to ensure employees "speak the same language"
    and all know basic concepts.
- Kevin Murphy's "Probabilistic Machine Learning" 1 & 2
  - Solid theory.
  - Many code examples.

Conclusion

Both Machine Learning methods and also the (hardware + software) technology that was built to support it allow us to break new ground in Theoretical Physics.

There are lots of exciting things to do.

We at Google Research are happy to discuss ML with academics.