Chapter 1: Deep Learning vs. Classical ML & Optimization

I. Going Deeper or not?

- **Universal approximation theorems**
  - [Cybenko, 1989]
  - [Hornik, 1989]
  - [Speicher, 1994]
  - [Kolmogorov, 1956]: exact decomposition of \( f \) by linear combinations of activation functions
  \[ f \in C^0, \exists \theta, N < \infty, g_{\theta,\epsilon,N} = f \]
  
  Being able to approximate \( (\exists \theta, ... ) \neq \) being able to learn & generalize

- **Depth simplifies the approximation estimation task**

  - [Hasselt, 2018]: why does deep & AMP learning work so well?
  
  \( \epsilon \times n \) input variables
  \( x \in \mathbb{R}^n \)
  \( y \in \mathbb{R}^m \)

  - For \( n \) variables:
    
    \[ \sigma(\alpha x) = \sigma(\alpha) + \alpha \sigma'(\alpha) x + \ldots \]
    
    - Given a smooth, possible to approximate multiplication of 2 variables with 4 neurons
    - Solves only the case of binary inputs \( \{0, 1\} \)

  - [Telgarsky, 2016]
    - **Target function**:
      
      \[ \sigma(\alpha x) \approx (\sigma(\alpha))^m \]
      
      \( \Rightarrow \) representable with a thin deep network (depth \( m \), \( n \) neurons)
      
      \[ \sigma(x) \approx (\sigma(x))^m \]
      
      \( \Rightarrow \) with a flat network: \( n+2 \) layers \( \approx \frac{n^2}{2} \) neurons
      
      \( \sum \) layers \( \approx 2^n \) nodes

    - [Pascanu, 2015]
      
      \[ f(x) = \frac{1}{2} x^T x + b \]
      
      \( \Delta \) parameters to tune \( \rightarrow \) \( d \) samples
      
      \[ f(x) = \text{function of } \frac{1}{2} d \text{ of } d \text{ parameters} \]
      
      \( n \) need \( n \) samples

    - If model not known: \( \epsilon = \alpha \) samples per layer
      
      
      Deep is sufficient (without width) [Lecun, 2016]. ResNet with 1 neuron layers - universal approximation property.
Does it work? Why?
- big success: computer vision & NLP
  - reasons: explicit geometry handling
  - compositionality
  - translation invariant
  - hierarchical models

- Random Forest: can't incorporate geometry (gt)
  - few diverging "independent" in the sense no geometry

Success in Reinforcement Learning
- Alpha-Go
- Alpha-0
- 60 million training games >> one life
- 4 hours training on 5000 TPU > 2 years of 1 TPU > 1000 years on single CPU
- Computep intensive training cost
  - NLP: huge networks (BERT, XLnet...) → ≈ 100000 $/

2) Gap between deep learning & classical ML/optimization

- Reminder: ML setup
  - dataset: \((x_i, y_i)\)
  - estimate \(F: x \rightarrow y\)
  - loss function: \(L(F(x), y_i)\)
  - regularizer: \(\sum_{F \in \text{parametrized family}} \text{Regularizer}(F)\)

\[ \inf_{F \in \text{parametrized family}} \sum_{i} \text{Loss}(F(x_i), y_i) + \text{Regularizer}(F) \]

\( \Rightarrow \) make \(F\) smooth \(\Rightarrow\) better generalization

L: Minimum Description Length principle (MDL)
- Occam's razor

- Now with deep learning:
  - millions of parameters
  - good to train networks with 10^6 parameters without overfitting → still, good generalization?
  - end with fewer samples than parameters
  - Do estimator converge?
  - classical optimization: the fewer parameters, the better
    - highly non-convex problem: thought had very hard
  - add more iterative optimization steps to help!
  - training criterion: cross-entropy
    - but evaluate in another way: accuracy ← not differentiable
  - common recommendation: check that the model is the too overfit
  - no regularizer

Closer look at overfitting:
- huge models don't necessarily overfit
2-layer neural network, with ReLU activations and 2nd weights that can associate each sample with a randomly-chosen target.

MNIST handwritten digits

random labels

2nd and 1st layers

0.8 accuracy

for these random labels

1000 samples d & 1000

even very small networks are completely able to overfit

Capacity is not the issue
(Novak, Vapnik, Charnik)

How to detect overfitting? / How to prevent it?
Possible regularizers: weight decay, drop out, data augmentation

nur dre:
not sufficient:
not necessary:

OSSG stopping

SEGD (optimizer) regulates

Regularization:
What kind of functional regularizers?

\[ \int \| \nabla f \|^2 \, dx \]

Black box: First step: check if the norm is 0?

ie: \( F = 0 \) everywhere

ie: all inputs are balanced

\( \theta \): \( (0,0,1,0,1) \)

\( \{0,1\} \rightarrow R \)

\( F = 0 \)

NP-hard

Stochastic approximation of the norm: Feasible

Drop out:

\( F \): o

\( \frac{\partial F}{\partial x_i} \)

\( x_i \)

Not far from

Half of inputs (of that neuron) are missing

Robustness to inputs

Redundancy

Dropout as a Bayesian Approximation

\( F \): o

\( \nabla \theta \)

Each drop-out realization is a sub-network

Family of sub-networks

ensemble technique

Early stopping

dataset \( \rightarrow \text{validation} \). Test only once!
Optimal noise

- around convergence:
  - linear: $\frac{\partial E}{\partial x} = 0$
  - quadratic: $x = \frac{-\nabla^2 E(x)}{\nabla^2 E(x)}$

- if $H > 0$ (positive definite):
  - all eigenvalues $> 0$:
    - SGD noise will not harm

- if not:
  - $L(x) = \sum_{i} \text{errors}$
    - $\Rightarrow$ error = stakes for points in training set
  - $L(x) = \sum_{i} \text{errors}$
    - $\Rightarrow$ SGD overfits after a while

- Be sure that $H > 0$:
  - add a weight regularizer: $\sum \| w \|^2$
  - weight norm is (L2 norm): $\| w \|_2 \Rightarrow \theta \rightarrow 0 \Rightarrow$ still overfit

Optimal landscape

- local minimum:
  - global minimum $\Rightarrow$ specific parameters of the network

- saddle point

Overparameterized

- Scaling descent of gradient $\Rightarrow$ “double gradient descent”

- fixed architecture
  - with variable width
- train without early stopping
- for a few $t$ steps
- without regularizing

- $x$: linear regression
  - $N$ samples of dim $d$,
  - desired outputs $y$; + noise $\epsilon$
  - $N$ steps with non-unique solutions
  - $N + 1$ parameters
  - $E'(\theta) = 0$
  - $\rightarrow$ unique solution
  - non-stable

(Depth double descent)
- Take away message: put as many neurons as you can.

H6: too many neurons lead to overfit? **No!**

**MBL:** high redundancy

- counting #parameters is not the right way to estimate model complexity
- loss compression techniques = 100 rate with similar accuracy
  - video dejitter editing on smartphone [Willing]