THEORY OF MACHINE LEARNING

LECTURE 20

NEURAL NETWORKS -- OPTIMIZATION

→ Project proposal.

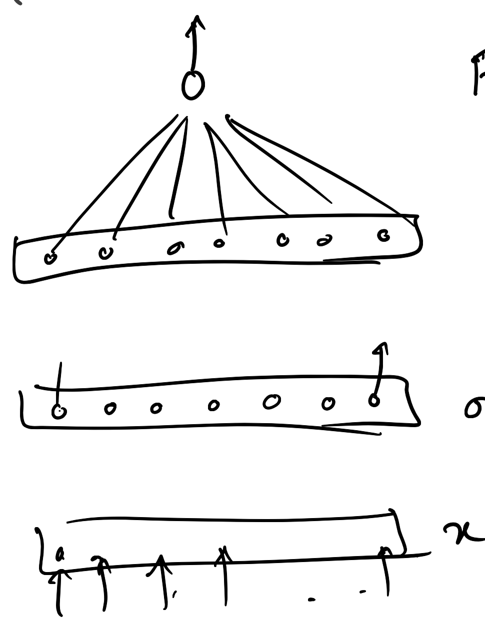
→ HW 3 – posted by tonight.

NEURAL NETWORKS (DNN)

- **Definition.** A layered "circuit" that takes a vector of input features x, produces output y = $F_r \circ F_{r-1} \circ \cdots \circ F_1(x)$, where each F_i is a function of the form $F_i(z) = \sigma(Az + b)$, for some activation function $\sigma()$ (that acts coordinate-wise)

F_r has a single output.

- Common activation functions:
  - Threshold
  - Sigmoid: (continuous approx.) $\frac{1}{1+e^{-x}}$
  - ReLU, Tanh
  - …

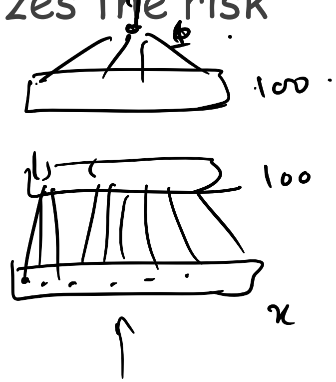


$\sigma(A_1 x + b) \equiv F_1$

LEARNING NEURAL NETWORKS

$\mathbb{R}^n$. label/value.

- **Question (supervised learning):** given data $(x_1, y_1), (x_2, y_2), \ldots$ from some distribution D, find h (with given "architecture") that minimizes the risk



- ERM problem usually called neural network **training**
- Neural networks can represent/approximate *any* function (Barron, Cybenko)
- Depth vs width trade-offs
- Choosing network architecture is key (inductive bias)
  - No general rules (heuristics like CNN, transformers, Hebbian learning, …)

(no ~~genera~~ "good" general technique).

LEARNING NEURAL NETWORKS

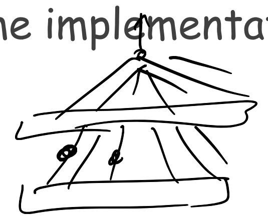
Theorem. (see textbook) Given an architecture, it is NP-hard to learn weights, even if classification error is 0 and we just have 3 internal nodes

(Threshold..)
ReLU, ..

- Worst case result – clearly not reflective of practice
- Can we obtain more "positive" results? (theoretical)

  → Width is large then you can train efficiently; depth 2 + random inputs.

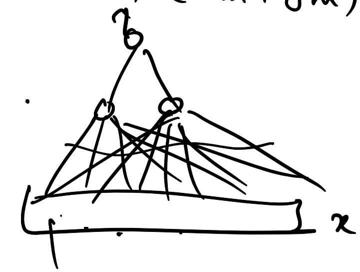
- Common algorithm: gradient descent – not too hard to compute gradients (exercise in chain rule)
  - Linear time implementation via "back propagation" (Rumelhart, Hinton, Williams) '86.



(storing "vertex gradients")

Theorem (NP hardness): Given $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$, with $x_1 \in \mathbb{R}^n$, $y_1 \in \{0,1\}$.

decide if $\exists$ a network of the foll structure: h :

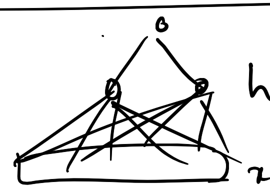


such that $h(x_i) = y_i \quad \forall i$

→ Inputs are " ~~adv~~ worst case" → what if inputs are $N(0,1)^n$?

→ Fixing network size to be "too small". → what if slightly bigger arch is ok?

"Generative Model" :



h $\quad x \sim N(0,1)^n \rightarrow h(x) \quad (x, h(x))$

Unknown/hidden network. input/output behavior to obtain network.

"correlations" between y and x_i

$(x_1, y_1), \ldots (x_N, y_N)$.

$L(x;w) = \underbrace{\sum_{i=1}^{N} \ell(x_i;w)}_{\sim \#\text{ wts.}}$

# of wts.

IS GRADIENT DESCENT (GD) GOOD?

- **Running time?** Naive GD takes time $\sim N \times |W|$ per iteration

  in practice, we divide N into "batches" & compute grads only using a batch.

- **Question:** *given data* $(x_1, y_1), (x_2, y_2), \ldots$, does running GD for N iterations result in training error <= OPT + f(N) [for some decreasing function?] $\frac{1}{\sqrt{N}}$
- Assuming the network architecture allows for zero error, does GD converge to zero error?

  (No, because of NP-hardness)

  [Anandkumar et al.].

- Alternatives to GD – method of moments (shallow nets), ...
  - [Chen, Klivans, Meka 2020]: in time exp(# internal nodes, depth, other params), can learn what GD can't ☺ FPT.

N inputs

OVERPARAMETRIZATION

(~2016 - -)

- Question out of desperation: can we show that GD is good in *any* reasonable generality? (width ≥ #inputs)

  "all local opt are close to global opt"

Theorem. [Jacot, Gabriel, Hongler 18] [Arora, et al. 2019] A width ~ n^3 network with *any* number of layers trained via GD from random initialization achieves zero training error. (whp.)

(key idea: parameters don't change much during training if width is so large...)

(point is that GD does it).

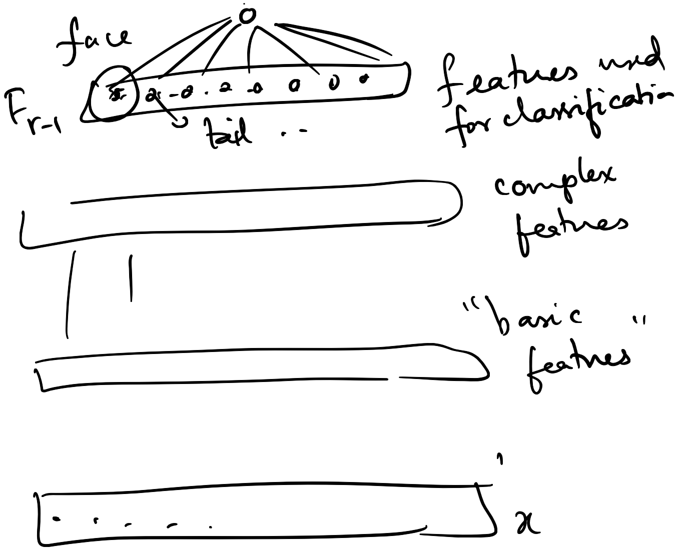
– Neural Tangent Kernel.

GD-based training in NNs works ~ kernel method with appropriately defined kernel. (at least with infinite width).

→ Width vs memorization

FEATURE LEARNING

NN is essentially performing a linear classification in "feature space"



- Problems with the infinite width regime
- What about learning "features" from data?
  - What does this even mean?

NN is "embedding" $x \to$ feature space.

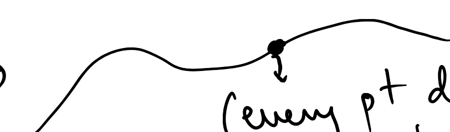
- Traditional approaches: sparse coding ⇝ autoencoders.

Given some inputs, can we "extract useful features"?

→ Contrastive learning.

→ discriminative in terms of predicting class + "orthogonal" to one another.

→ Manifold learning →



(every pt defined by ~$k \ll N$ parameters)

→ Feature embedding of data points.

$$x \longrightarrow f(x)$$

raw data feature embedding.

(i) $f(x)$ should be useful to discern between classes.

(ii) coordinates of $f(x)$ should be "independent" of one another.

(iii) $f(x)$ should be "as informative" about x as possible.

Knowing $f(x)$, ~~there~~ we should be able to "recover" x to a certain extent.