THEORY OF MACHINE LEARNING

LECTURE 19

NEURAL NETWORKS – REPRESENTATION, OPTIMIZATION

groups of 2

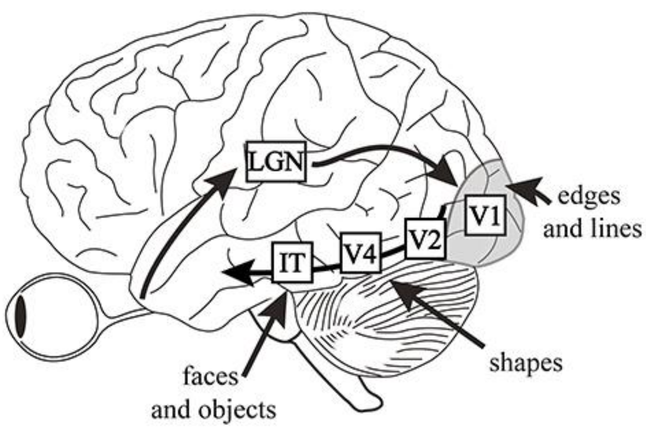
→ HW3 will be out this weekend.

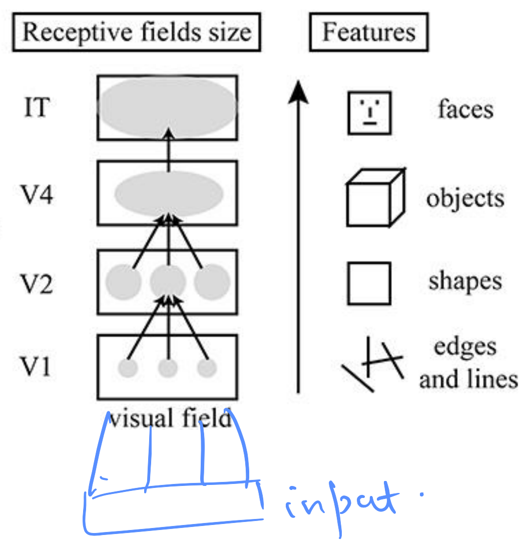
→ Project proposal → find a paper you think is "worth presenting" as project, write one para abt it.

→ No lecture this Thursday..

RECAP

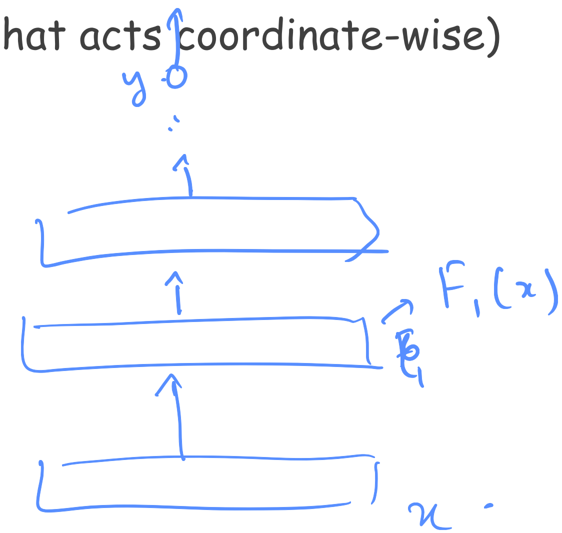
- Idea behind neural nets:
  - Perceptrons detect "basic" or "primitive" features; 'composing' them allows for complex decision-making
  - Supported by human visual system (V1, V2, …)



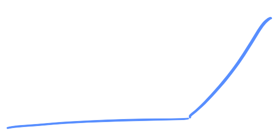


RECAP: ARTIFICIAL/DEEP NEURAL NETWORK (DNN)

- **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)



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



LEARNING NEURAL NETWORKS

- Defines a hypothesis class
- **Question (vanilla supervised learning):** given data $(x_1, y_1), (x_2, y_2), \ldots$ from some distribution D, find h in this class that minimizes the risk
- ERM problem usually called neural network "training" – given data, find best hypothesis ($f(x_i) = y_i$) for all i

THEORY OF DEEP LEARNING – THREE BROAD DIRECTIONS

deep neural net.

- $\boxed{\text{Expressibility}}$
  - What kinds of functions can be obtained using a DNN? (complexity of functions → no focus on training).
- Training complexity & training dynamics for GD and variants
  - Can the ERM problem be solved efficiently? What guarantees are possible?
- Generalization (If you get m training examples, can you ensure that test error (assuming same dist) is $< f(m)$ $\frac{1}{\sqrt{m}}$?.
  - What kind of generalization bounds can we prove? (VC dimension?)

  \# params ≈ # data samples..

Key: "easy" answers for all questions, but unsatisfactory for realistic settings

EXPRESSIBILITY BASICS

- **Barron's theorem [93].** *Any* continuous function f that satisfies an appropriate "niceness" condition (parametrized by C) can be approximated to error ϵ *(in L2!)* by a 2-layer NN with $\sim \frac{C^2}{\epsilon}$ internal nodes
- (Nice functions can be approximated by small NNs)
- **Universal approximation** [Cybenko, Hornik '87,'91]. *Any* continuous function (over a compact domain) can be approximated by a 2-layer NN with *any* non-linearity (not a polynomial)

Curse of dimensionality for Cybenko **(not Barron)**

Moral:
depth-2 nets are universal (express ANY function — given large enough width.)

WHY "DEEP" NETWORKS?

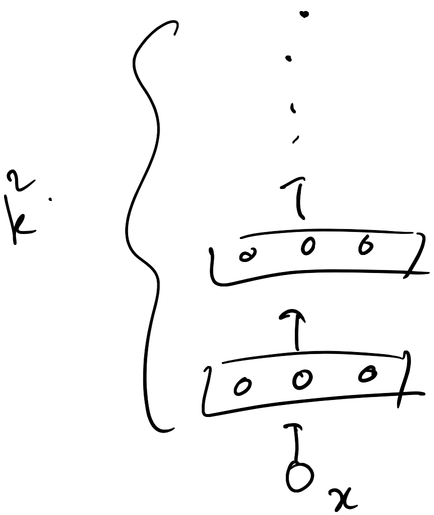
- Practical intuition:

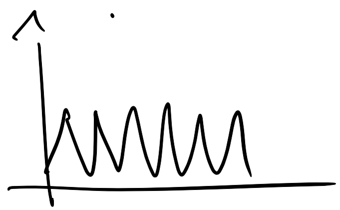
  high-level.
  - Depth allows "meaningful features" while width is for "brute force memorization"
- Universality results degrade rapidly with dimensions
  - Curse of dimensionality
  - Modern nets work with high dimensional data
- Does higher depth lead to higher expressibility (with much fewer neurons)?
- Yes! [Eldan and Shamir, Telgarsky]

POWER OF DEPTH

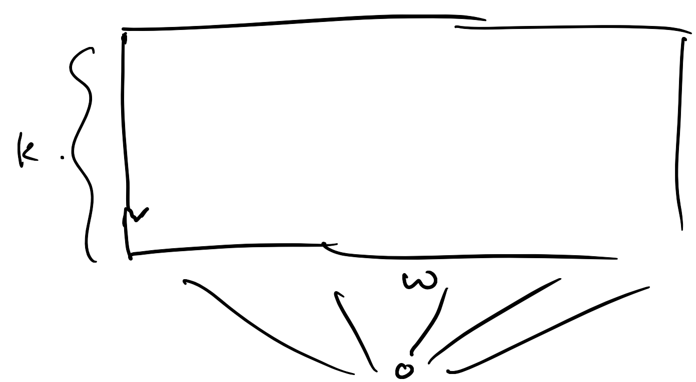
Theorem [Telgarsky 16]. There exists a network of depth k^2 and $O(1)$ width that computes function f, with the property that *any* network of depth k that approximates f requires width > 2^k

(For more general piecewise poly functions, first bound changes to k^3)





saw-tooth fn
with $\sim 2^{k^2}$ oscillations.



$w > 2^k$.

(1) A function with $\sim 2^{k^2}$ oscillations can be output by a depth k^2 net with 3 neurons/layer.

(2) Any function that is the output of a ~~with~~ width w, ~~for~~ depth k ReLU network is composed of $\leq (2w)^k$ piecewise linear "pieces".

(3) ~~If~~ A pwl function ~~that is~~ with fewer than $2^{k^2/2}$ pieces cannot even approximate fn from (1) to an error $\sim \frac{1}{4}$.

$$(2w)^k > 2^{k^2/2} \implies w \gtrsim \frac{2^{k/2}}{10}.$$

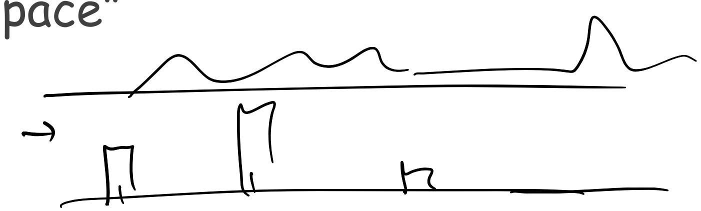
[Similar results expected for other fn classes].

MORALS

C : $\int |\omega| \, |\hat{f}(\omega)| \, d\omega$.

(Barron's thm)

- Depth allows capturing "complex patterns" (oscillations).
- Width allows capturing different behaviors in "different regions of space"



- What is the right network for an application?

  – Mode collapse.

  → Circuit complexity.

  - Very hard question (Neural Architecture Search)
  - Example of Vision + NLP problems
  - 2013 Hebbian principle → 1950s. → "Neurons that fire together wire together"
  - Needs exploiting domain knowledge (physics informed ML)

[Arora, Ma, Risteski]: deeper networks have a higher "Barron ~~coefficient~~ Constant".

(grows exp(d))

NEURAL NETWORK TRAINING

input → label. (x_N, y_N).

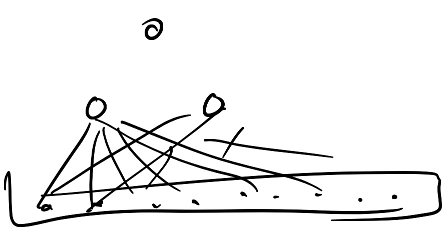
- **Supervised learning of NN:** given data $(x_1, y_1), (x_2, y_2), \ldots$ from some distribution D, find h that minimizes the ~~empirical~~ risk (empirical is what we care about.).
  - Standard metrics: squared loss, cross entropy ($h_w(x)$ is the output of NN).

find w, so as to minimize $\sum_{i=1}^{N} \left(h_w(x_i) - y_i\right)^2$ $w \to$ weights.

$l(h_w(x_i), y_i)$

- ERM problem for neural nets
- NP-hard to learn weights, even if classification error is 0 and we just have 3 internal nodes

[For the squared loss, ERM problem is NP hard, because h_w is non linear.].



input

COMMON ALGORITHM – GRADIENT DESCENT

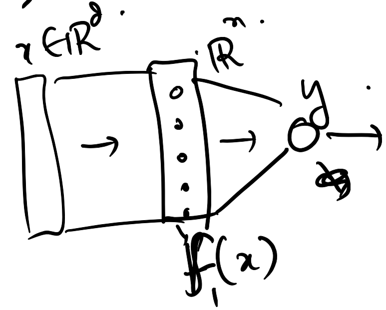
→ Recall how we solved ERM problems:

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

$h(w; x)$: hypothesis ~~where~~ defined by "weight vector" w on ~~the~~ input x.

(linear hypotheses, $h(w; x) = \langle x, w \rangle + c$)

NN?



weight "vector" "w" consists of

$$w = (A, v, b, b')$$

$$F_1(x) = \sigma(Ax + b)$$

$$y = \sigma(v^T F_1(x) + b')$$

In general, weights $\equiv$ (tuple of matrices, biases, ...) (model parameters).

<u>ERM:</u> find w so that $\sum_{i=1}^{N} \ell\big(h(w; x_i), y_i\big)$ is min

$\equiv$

given training data, this is $G(w)$;

→ If G were convex, gradient descent (GD) provably works...

→ <u>NNs:</u> G is usually (for most choices of σ) not convex.

→ GD still finds "local opt".

<u>Hope:</u> with sufficient data, finds a "good" local opt

★ "Quality" of "local opt" → dif. between local & global.. } "Landscape analysis"

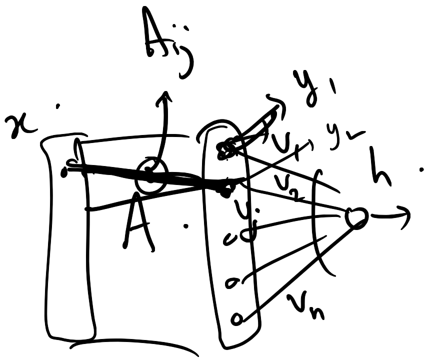
★ # of local opt ⟶

★ <u>Lots of global opt</u> (or "nearly" global opt), but some generalize way better than others

How to do gradient:

$w^{(0)}$

$$w^{(t+1)} = w^{(t)} - \eta \nabla G(w^{(t)})$$



$$G = \sum_{i=1}^{N} \left(h(w; x_i) - y_i \right)^2$$

For simplicity, assume $G = \left(h(w;x) - y \right)^2$

$$\nabla G = 2\left(h(w;x) - y \right) \nabla h(w;x)$$

how does h change if you vary w?

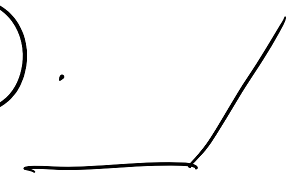
$$h(w;x) = \sigma\left(v^T (\sigma(Ax+b)) + b' \right).$$

$$= \sigma\left(v_1 y_1 + v_2 y_2 + \dots + v_n y_n + b' \right)$$

∇h has components along each entry of A, b, b', v.

$\frac{\partial h}{\partial v_i}$ is "essentially" y_i (if σ were ReLU).

$\partial v_i \to$ one parameter.

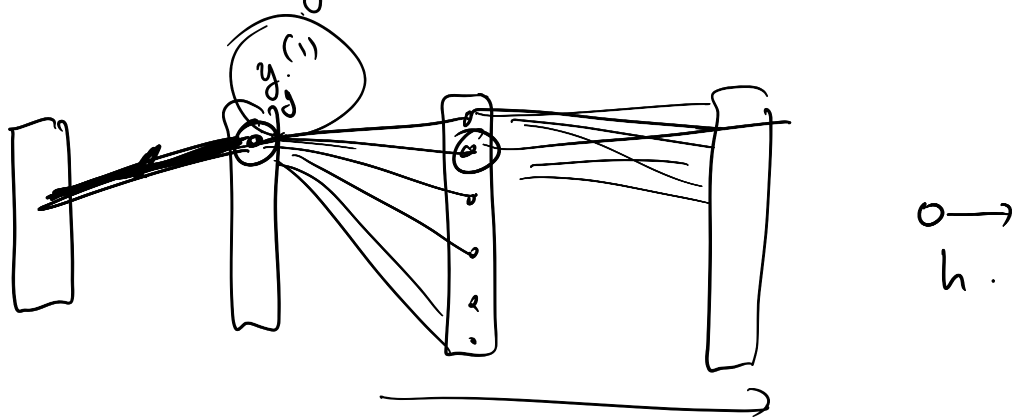


$$\sigma'\left(v^T(\dots) \right) \cdot y_i$$

How do you diff. wrt. A_{ij}?

$$\frac{\partial h}{\partial A_{ij}} = \sigma'\left(v^T(\dots)\right) \cdot \frac{\partial y_j}{\partial A_{ij}} v_j$$

* If you have > 2 layers



[Hinton et al. '86]

Backpropagation algorithm → allows you to compute gradients in ~ linear time.

[Note on Back-propagation].