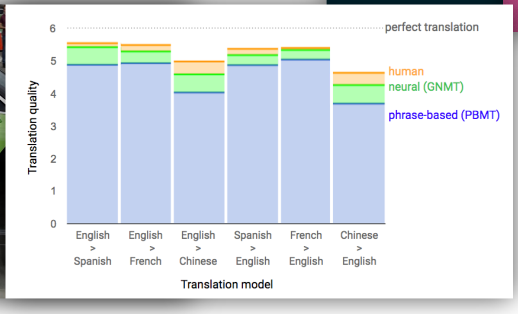
Opening the black box: Toward mathematical understanding of deep learning

Sanjeev Arora
Princeton University

http://www.cs.princeton.edu/~arora/
Group website: unsupervised.princeton.edu
Blog: www.offconvex.org
Twitter: @prfsanjeevarora

(Funding: NSF, Simons Foundation, ONR, Schmidt Foundation, Amazon Research, Mozilla Research, SRC/JUMP)

Deep learning in the news



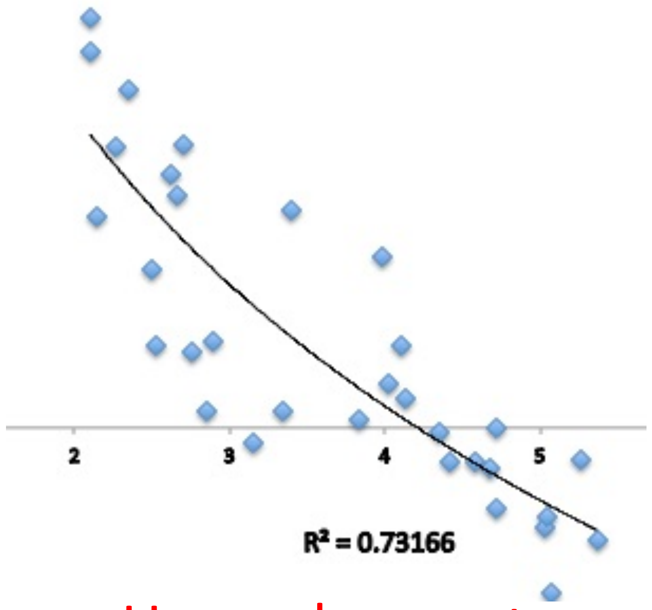
This talk

- Some difficult mathematical questions about deep learning (and why they are difficult)
- Examples of mismatch between traditional frameworks (learning theory, optimization) and deep learning phenomena
- Survey of some understanding (and new puzzles) from recent years.
- Wrap up

Main idea of ML: Curve fitting (Gauss, c. 1800)

Phillips curve (1958):

Inflation



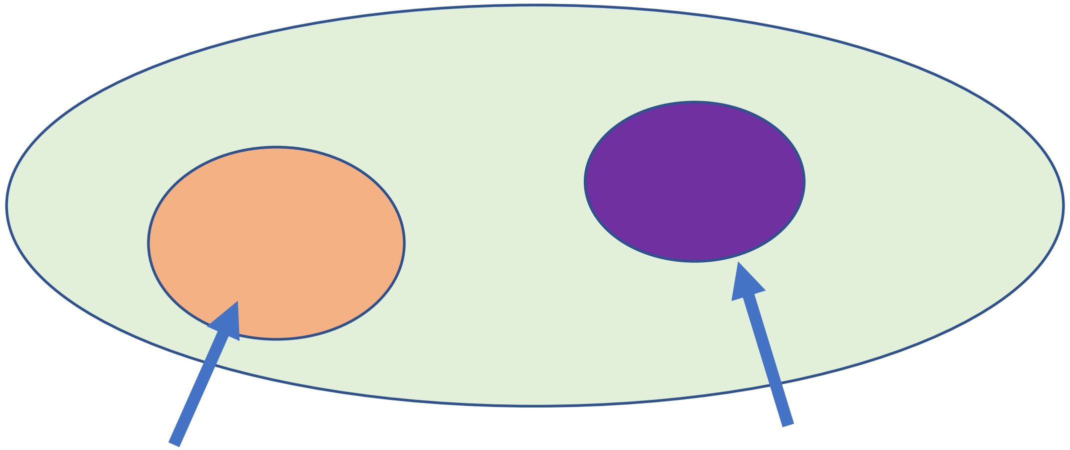
Unemployment

Machine Learning = Surface fitting, with many more variables

Statistical issues (common to all data science)

Datapoint = (input, label)

D: Distribution on datapoints



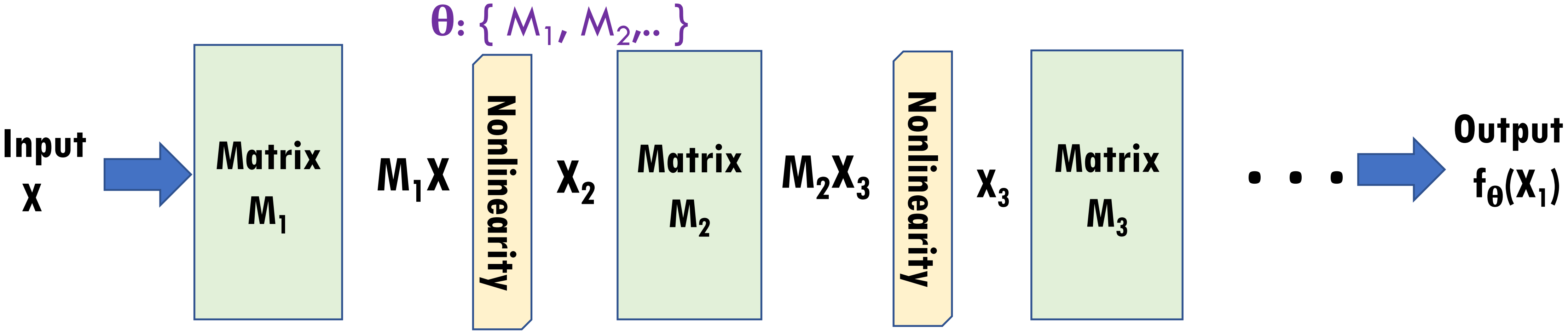
Random sample
used for training
("training set")

Random sample ("Holdout set") to use
as proxy for estimating trained model's error
on full distribution. (AKA Test error)

If test error >> training error,
model has "overfitted"
("failed to generalize")

("Deep" = multilayered)

Deep Nets*



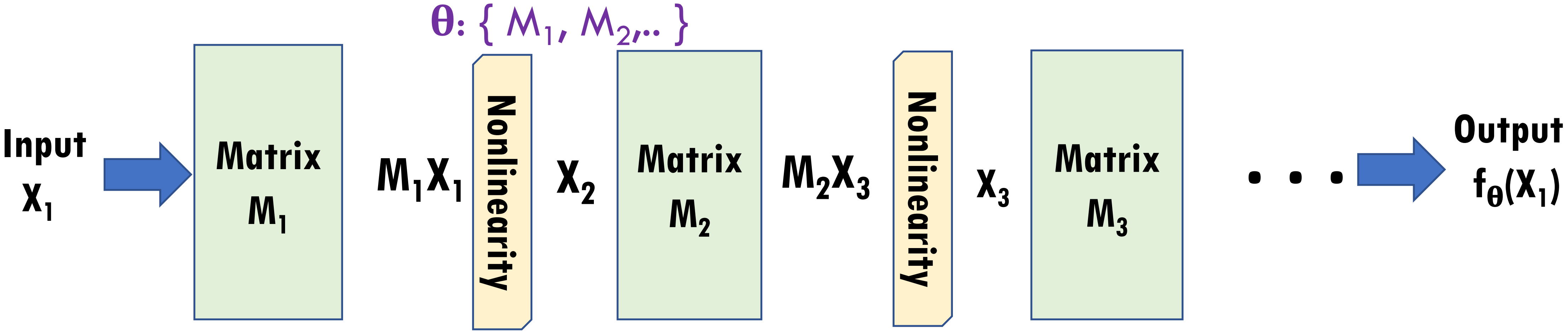
"ReLU Nonlinearity": Given a vector, turn all negative entries to zero.

Training data: $\{(X_1^{(i)}, Y^{(i)}): i=1,..,N\}$ $\ell(\theta) = \frac{1}{M}\sum_{i=1}^{M}(f_\theta(X^{(i)}) - Y^{(i)})^2$

(Loss function)

(*Highly simplistic: could have "convolution", "bias", "skip connections", other loss fns etc.)

Deep Nets (more formal)



"Nonlinearity": Given a vector, output same vector but negative entries turned to zero.

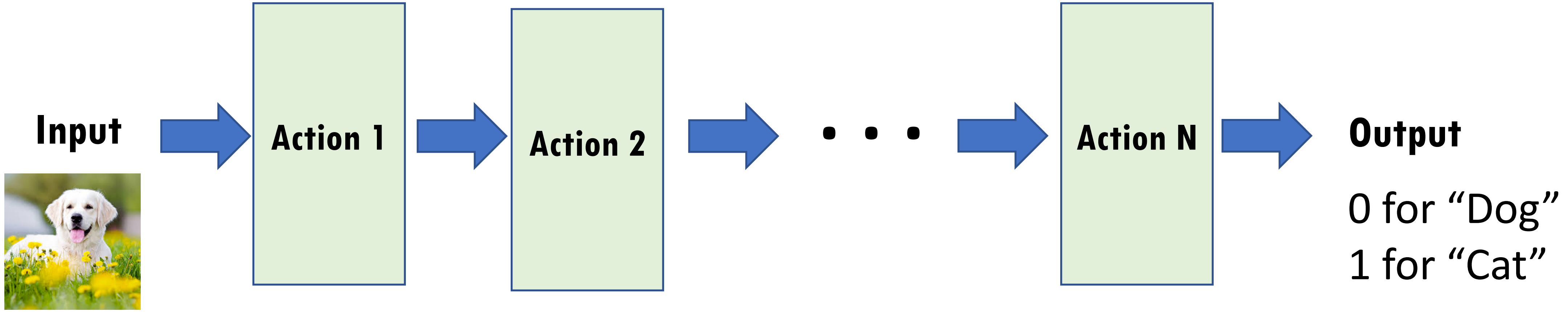
Training data: $\{(X_1^{(i)}, Y^{(i)}): i=1,..,N\}$ $\ell(\theta) = \sum_{i=1}^{N} (f_\theta(X_1^{(i)}) - Y^{(i)})^2$

Algorithm: Gradient Descent (Move θ in direction opposite to gradient of loss $-\nabla_\theta(\ell)$

Backpropagation computes gradient; clever application of chain rule [Werbos'77, Rumelhart et al'84]

Differential computing paradigm

Deep Net (simplistic!)

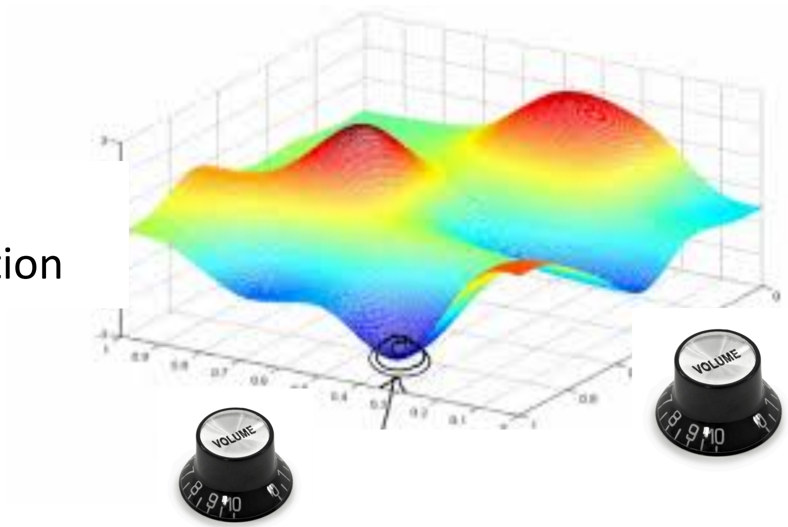


Output depends on large bank of parameters ("tunable knobs")

Gradient descent Training: Using large training set of labeled inputs, adjust tunable knobs to make the Net's output match the labeled outputs as closely as possible.

Mystery 1: Gradient descent (GD) quickly makes training loss zero

Loss Function

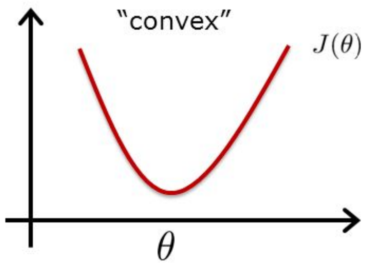


$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_\theta(\ell)$$

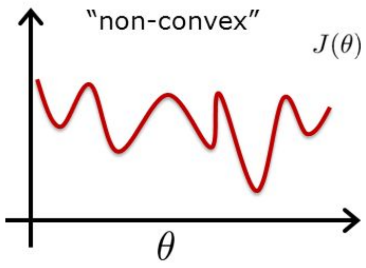
$$(\eta = \text{step size/"learning rate"})$$

Flies against experts' intuitions!

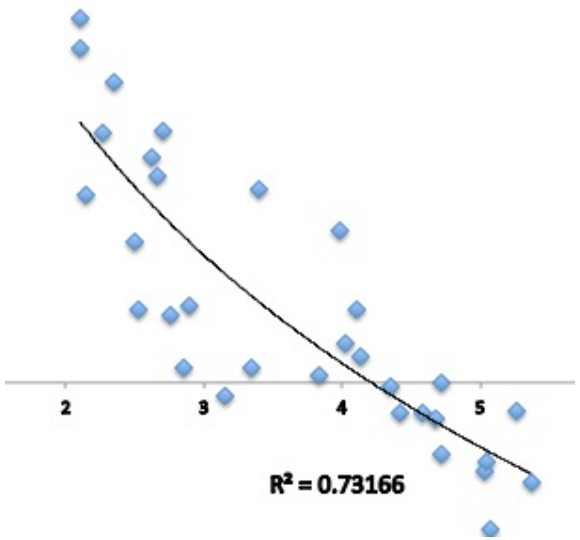
CONVEX

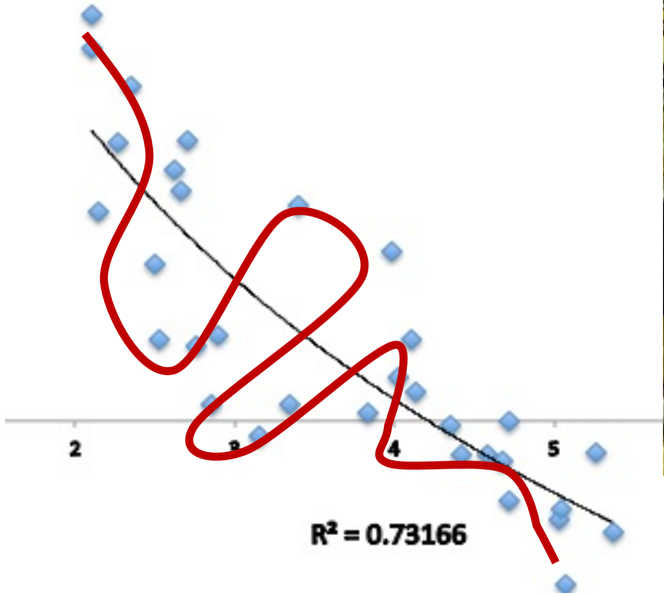


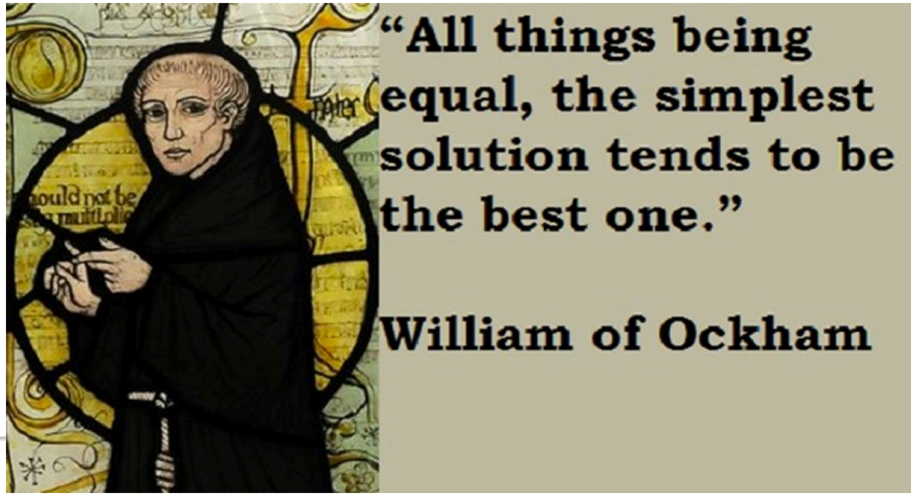
NONCONVEX



Mystery 2: NO Overfitting







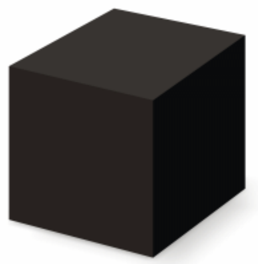
Rule of thumb: Overcomplicated explanations overfit to training data and do not "generalize" to explaining new data.

Overparametrized nets (way more parameters than data points) outperform smaller models

Many other mysteries, some are later in the talk

Hurdles for theory

Loss function is currently a black box to mathematics since it depends on complicated training data ("dog vs cat", "English to German", etc.?)

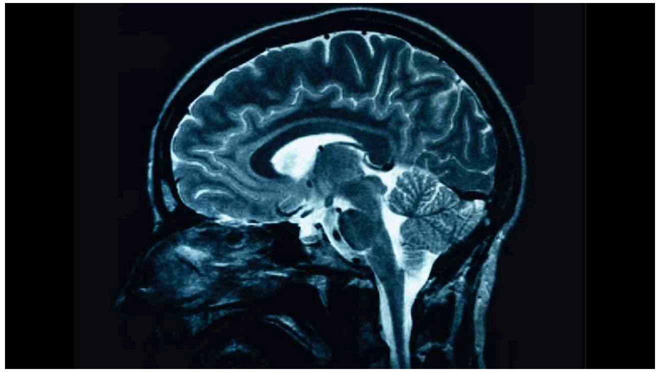


No fruitful theory is possible for blackbox (ie fully general) nonconvex function! (No efficient algorithm to find optima, let alone those that generalize.)

Min $\ell(\theta)$ for some loss function $\ell()$; solve as fast as possible

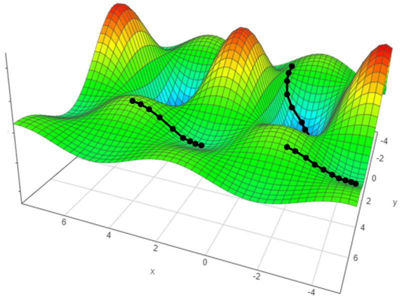
Unclear: Is optimization even the right language for understanding current deep learning?

Old debate: Does the brain ("spiking neurons in a vat of chemicals…") amount to optimization of an objective?



Suggestion today: Deep net training is also imperfectly captured by value of the objective.

Multiple minima! Any tweak to training => different trajectories, with different solution. Trajectory properties determine generalization!



["Implicit bias of gradient descent" [Neyshabur et al'17, Gunasekar et al'17])

Agenda for theory: Open the black box

Gradually analyze more and more complicated deep nets, and see if theory can explain these mysteries…

Example: Understand trajectory of GD for 3-layer nets on a simple dataset. Understand process of reaching zero loss and good generalization

Agenda for theory: Open the black box

Gradually analyze more and more complicated deep nets, and see if theory can explain these mysteries…

CAVEAT: Evolution of systems of tens of millions of parameters notoriously hard to analyze in math (e.g., famous unsolved questions in PDEs, dynamical systems, complex systems...)

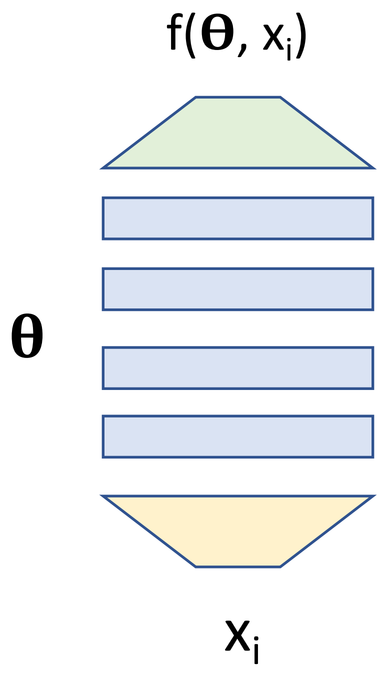
Vignette 1: Training of infinitely wide* Deep Nets

(“GD picks a meaningful solution out of infinitely many possibilities”)

(* Motivations: “Thermodynamic limit” + “Gaussian Process View of DL”)

Paper 1: On Exact Computation with an Infinitely Wide Net (A., Simon Du, Wei Hu, Zhiyuan Li, Russ Salakhutdinov, Ruosang Wang NeurIPS‘19)

Inspired by works on overparametrized nets [Li, Liang’18], [Allen-Zhou, Li’18] and infinite fully connected nets [Jacot et al’18]



Dataset: UCI Primary Tumor
(multiclass; 17-dimensional input, # training samples= 339)

Want to train fully connected 5-layer net on it. Infinitely wide!

Means: Keep input and output layer fixed, but allow width of inner layers $\to \infty$
(initialize with suitably-scaled Gaussians so expected node value is equal at all layers)

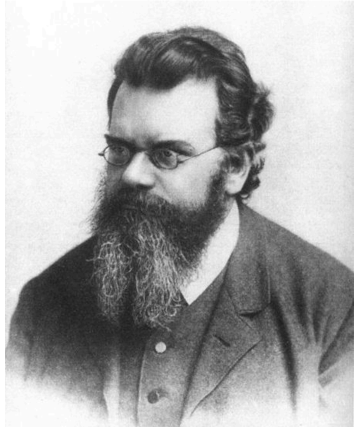
Too expressive! Will overfit to training data.
(Arbitrarily wide 2-layer nets can represent every finite function, so # of zero-loss solutions $\to \infty$)
Plus, infeasible to train!

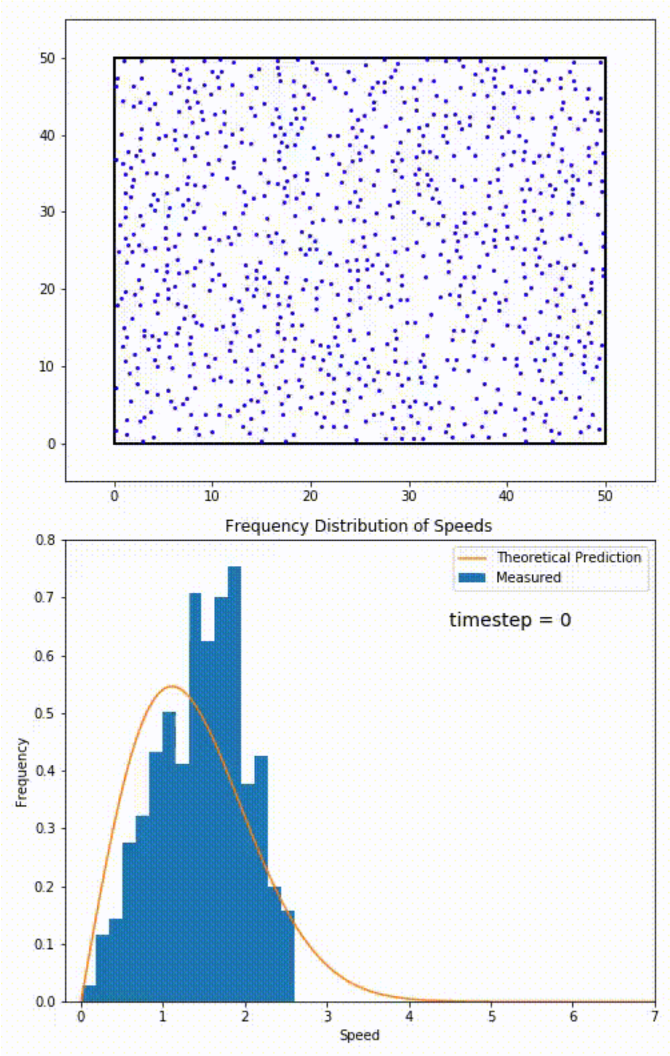
Test accuracy: 51.5

(Random Forest: 48.5, Gaussian Kernel: 48.4)

For CIFAR10, infinite convolutional net has accuracy 77% (vs 83 % for corresp. finite net)

Reminder: Original “thermodynamic limit”





(credit: Wikipedia)

Maxwell-Boltzmann distribution

$$f(v)\, \mathrm{d}^3 v = \left(\frac{m}{2\pi kT}\right)^{3/2} e^{-\frac{mv^2}{2kT}}\, \mathrm{d}^3 v,$$

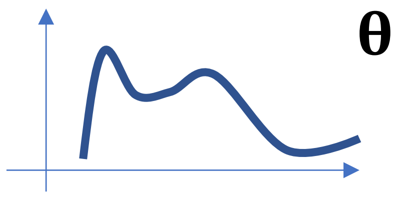
Neural Tangent Kernels NTKs (and Convolutional NTKs)

[Jacot et al'18][Arora et al'19]

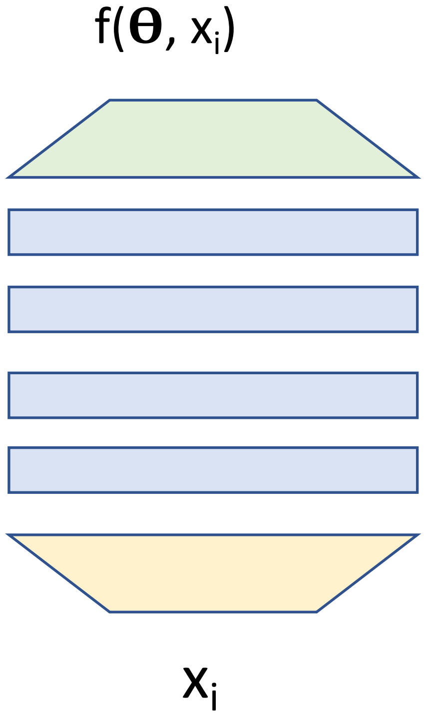
Following are equivalent for any finite dataset: (i) infinite-width fully connected nets (resp., Conv Nets) trained with GD with infinitesimally small learning rate
 (ii) Kernel l_2 regression wrt NTK (resp., CNTK)

Thm[A. et al Neurips'19] Efficient and GPU-friendly algorithm for computing CNTK exactly. (Dynamic Programming!)

⇒ Can compute exact performance of infinite-width net on finite datasets.



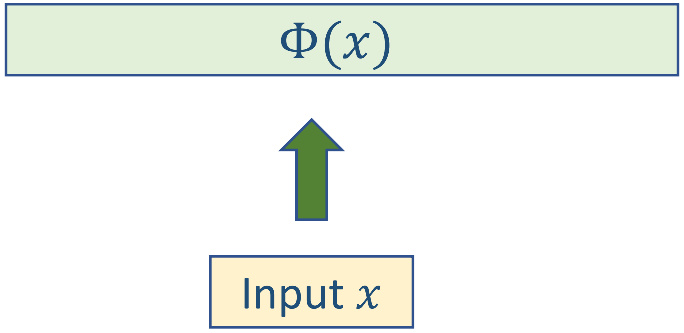
Distribution of values at a layer



Previous slide unpacked (Kernel linear regression/SVM reminder)

Kernel trick: l_2 regression possible if can compute $< \Phi(x_1), \Phi(x_2) >$ for any given input pair x_1, x_2

"Reproducing Kernel Hilbert Space"



(e.g., polynomial kernel, Gaussian kernel,..)

Neural Tangent Kernel H^*:

Each coordinate of $\Phi(x)$ corresponds to parameter w in the net.

Corresponding entry is $\partial(output)/\partial w$ at $t = 0$

To do regression wrt H^* only need algorithm to compute $< \Phi(x_1), \Phi(x_2) >$ for any given input pair x_1, x_2

(our algorithm does that efficiently)

[A., Du, Hu, Li, Salakhutdinov, Wang, Yu ICLR'20]

Classifier	Friedman Rank	Average Accuracy	P90	P95	PMA
NTK	**28.34**	**81.95%±14.10%**	**88.89%**	**72.22%**	**95.72% ±5.17%**
NN (He init)	40.97	80.88%±14.96%	81.11%	65.56%	94.34% ±7.22%
NN (NTK init)	38.06	81.02%±14.47%	85.56%	60.00%	94.55% ±5.89%
RF	33.51	81.56% ±13.90%	85.56%	67.78%	95.25% ±5.30%
Gaussian Kernel	35.76	81.03% ± 15.09%	85.56%	**72.22%**	94.56% ±8.22%
Polynomial Kernel	38.44	78.21% ± 20.30%	80.00%	62.22%	91.29% ±18.05%

On 90 classic UCI datasets (<5000 samples)

[Processed as in Fernando-Delgadez'14]

(Li, Wang, Hu, Yu, Salakhutdinov, Arora, Du, Manuscript 2019]

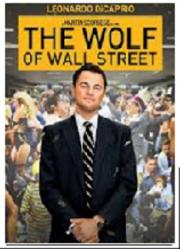
Souped-up CNTK that rivals AlexNet on CIFAR10 (89% accuracy; best kernel that was not trained on data)

Open: Fully understand generalization for kernels !

“Netflix Challenge” (~2007)

MOVIES

USERS

				
Bob	4	?	?	4
Alice	?	5	4	?
Joe	?	5	?	?
Sam	5	?	?	?

Vignette 2: Solving matrix completion via deep linear nets

(“GD is amazing even in simple models, but exactly formalizing its effect can be tricky.”)

“Implicit regularization in deep matrix factorization” [A., Nadav Cohen, Wei Hu, Yuping Luo, NeurIPS 2019]

Matrix Completion

Unknown low rank *$n \times n$ matrix M. Entries revealed in a random subset Ω of locations*

Goal: Recover M.

[Srebro et al'05] Find matrix with best squared error and smallest nuclear norm (convex!)

$$\sum_{ij \in \Omega} (M_{ij} - b_{ij})^2 + \underbrace{\lambda |M|_*}_{\text{regularizer}}$$

$|M|_*$ = sum of singular values of M
Low rank: "# nonzero singular values is small"

[Candes, Recht'10]: This is statistically "optimal" !

[Gunasekar et al'17] Find M as product of 2 matrices (depth 2 linear net); no regularizer!

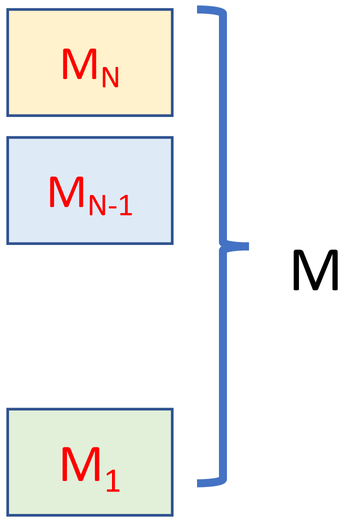
$$\sum_{ij \in \Omega} ((W_2 W_1)_{ij \in \Omega} - b_{ij})^2$$

Conjecture (Gunasekar et al.'18): In depth 2 linear nets GD implicitly minimizes $|M|_*$

Infinitely many solns, but empirically GD finds soln as good as nuc. norm minimization!

Deep matrix factorization (= multilayer linear nets)

[A., Cohen, Hu, Luo, NeurIPS'19]

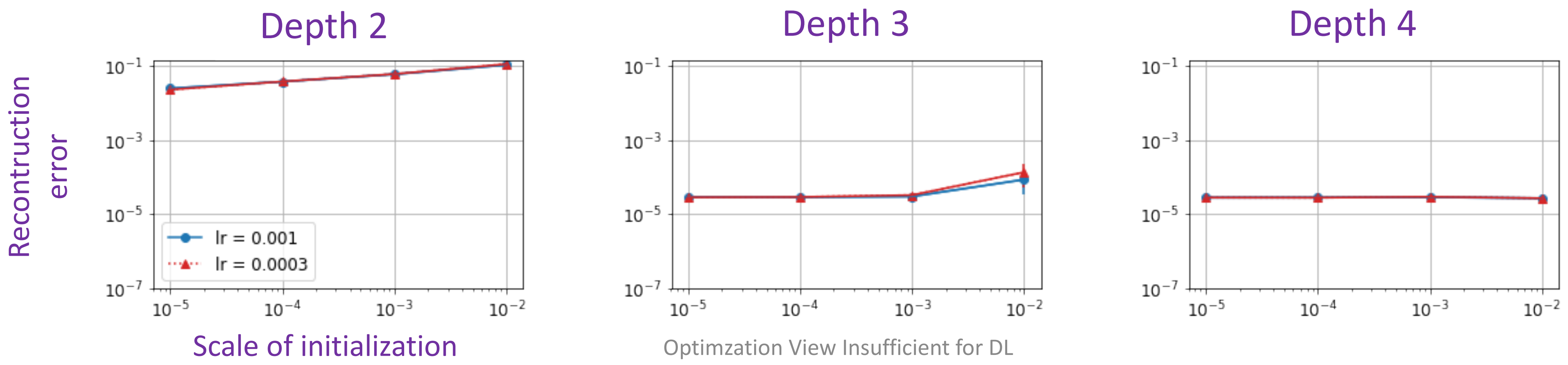


$$\sum_{ij\in\Omega}\left((W_N W_{N-1}\cdots W_2 W_1)_{ij\in\Omega} - b_{ij}\right)^2$$

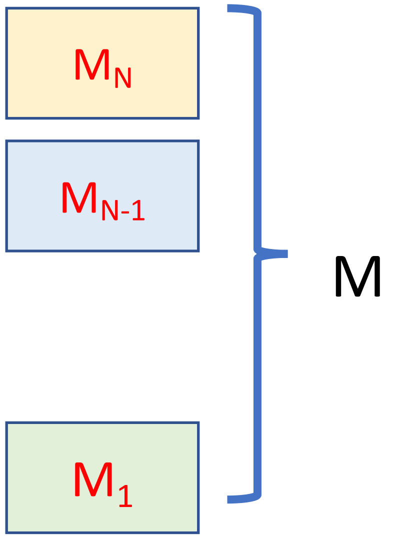
"Ignore domain knowledge trust backprop!"

Good news 1: Empirically, solves matrix completion better (ie with fewer revealed entries) than Nuclear Norm Minimization! (Also mathematical explanation..)

[Initialization: Small random; Learning rate: very small)



Sketch of mathematical analysis



1. Show that singular values and sing. vectors of end-to-end matrix M are analytic functions of time t.

2. **Theorem 3.** *The signed singular values of the product matrix W evolve by:*

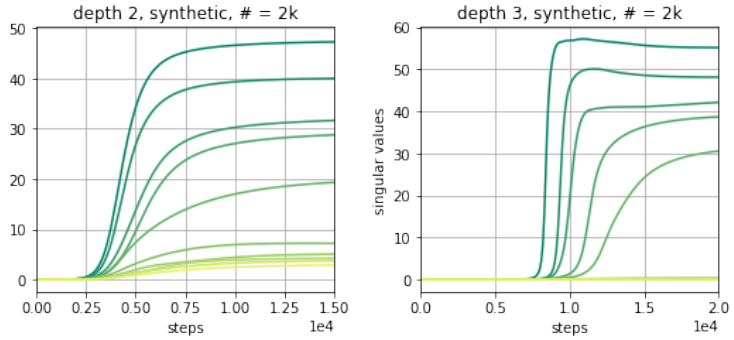
$$\dot{\sigma}_r(t) = -N \cdot \left(\sigma_r^2(t)\right)^{1-1/N} \cdot \left\langle \nabla \ell(W(t)), \mathbf{u}_r(t)\mathbf{v}_r^\top(t) \right\rangle \quad ,$$

"Rich get richer"; promotes low rank

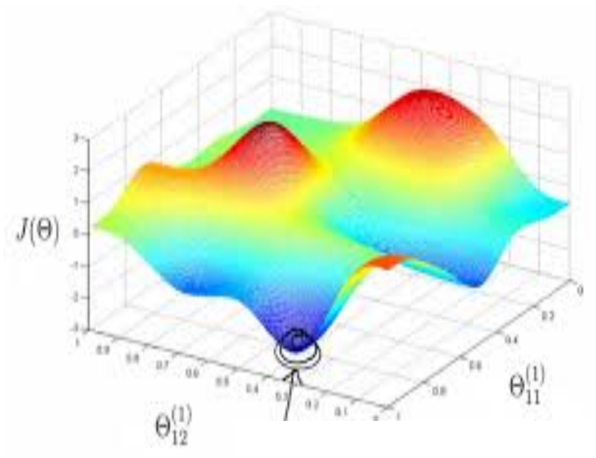
(Interpretation: GD builds up matrix M one singular vector at a time, not all at once.
Building up stops once gradient of loss goes to zero.)

(Paper presents evidence that Gunasekar et al conjecture is false)

Depth 2 Depth 3



Evolution of sing. values w/ time



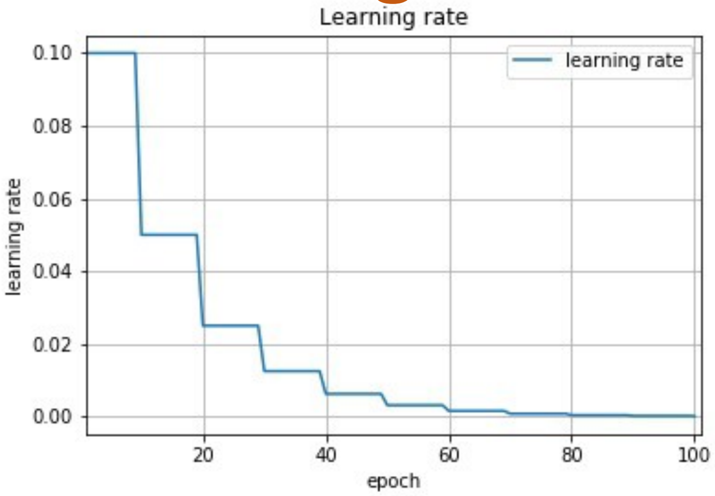
Vignette 3: Exponentially increasing learning rate works for deep learning.

[Zhiyuan Li and A., ICLR'20]

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_\theta(\ell)$$

$$(\eta = \text{step size/"learning rate"})$$

Learning rate in traditional optimization



$$w^{\{t+1\}} \leftarrow w^{\{t\}} - \eta \cdot \nabla L\left(w^{\{t\}}\right)$$

Standard schedule: Start with some l.r.; decay over time.

(extensive literature in optimization justifying this)

Result 1 (empirical): Possible to train today's deep architectures, while growing l.r. exponentially (i.e., at each iteration multiply by $(1 + c)$ for some $c > 0$)

Result 2 (theory): Mathematical proof that nets produced by existing training schedules can also be in obtained (in function space*) via exponential l.r. training schedules.

(* In all nets that use batch norm [Ioffe-Szegedy'13] or any other layer normalization scheme.)

Learning Rate

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{v}_t$$

$$\boldsymbol{v}_t = \gamma \boldsymbol{v}_{t-1} + \nabla_{\boldsymbol{\theta}} \left(L_t(\boldsymbol{\theta}_{t-1}) + \frac{\lambda_{t-1}}{2} \|\boldsymbol{\theta}_{t-1}\|^2 \right)$$

Momentum

ℓ_2 regularizer (aka Weight decay)

General training algorithm for deep learning today

Thm (informal): For nets w/ batch norm or layer norm, following is equivalent to above : weight decay 0, momentum γ, and LR schedule $\eta_t = \eta_0 \alpha^{-2t-1}$ where α is nonzero root of

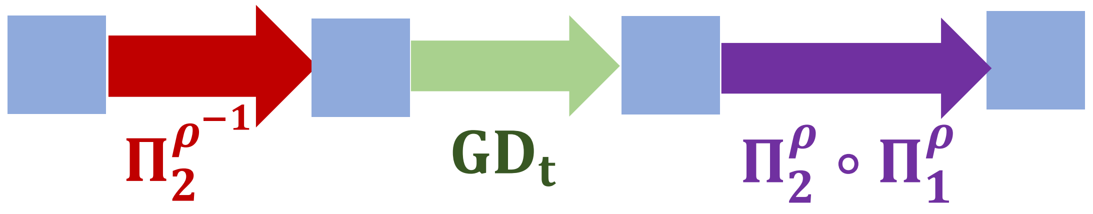
$$x^2 - (1 + \gamma - \lambda\eta)x + \gamma = 0,$$

(proof uses: a trajectory-based analysis + scale-invariance created due to batch-norm)

$$f_{\boldsymbol{\theta}} = f_{c\boldsymbol{\theta}}, \ \forall c > 0 \quad \Rightarrow \quad \nabla_{\boldsymbol{\theta}} L\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} = c \nabla_{\boldsymbol{\theta}} L\big|_{\boldsymbol{\theta}=c\boldsymbol{\theta}_0}, \textit{ for any } c > 0$$

Run GD with WD for a step: $\mathrm{GD}_t^{\rho}(\boldsymbol{\theta}, \eta) = (\rho\boldsymbol{\theta} - \eta\nabla L_t(\boldsymbol{\theta}), \eta)$; $(\boldsymbol{\rho} = \mathbf{1} - \boldsymbol{\lambda\eta})$

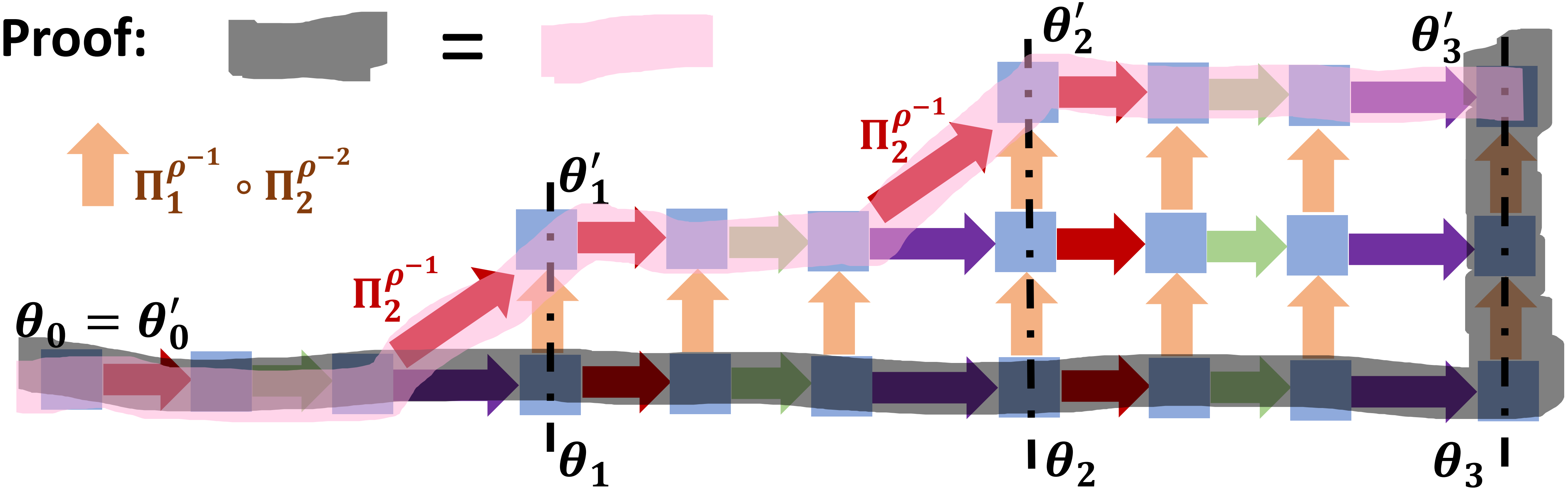
$$GD_t^{\rho} = \Pi_2^{\rho} \circ \Pi_1^{\rho} \circ GD_t \circ \Pi_2^{\rho^{-1}} =$$

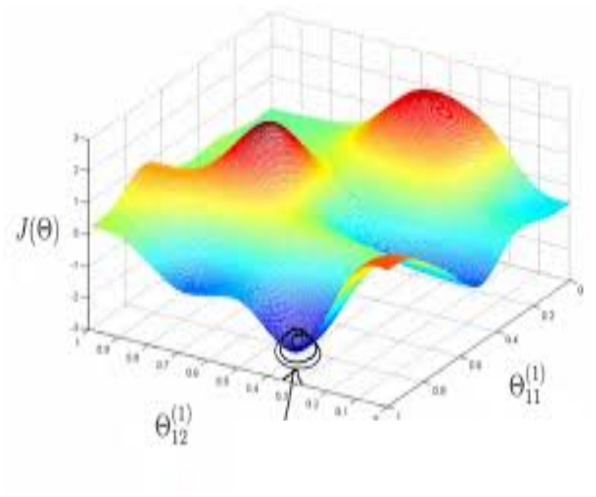


Theorem: GD + WD + constant LR= GD + Exp LR.

$$\Pi_1^{\rho^{-t}} \circ \Pi_2^{\rho^{-2t}} \circ GD_{t-1}^{\rho} \circ \cdots \circ GD_0^{\rho} = \Pi_2^{\rho^{-1}} \circ \mathrm{GD_{t-1}} \circ \Pi_2^{\rho^{-2}} \circ \cdots \circ \mathrm{GD_1} \circ \Pi_2^{\rho^{-2}} \mathrm{GD_0} \circ \Pi_2^{\rho^{-1}}$$

Proof:





Vignette 4: How to allow deep learning on your data without revealing your data.

Instance-hiding schemes for private distributed deep learning [Huang, Song, Li and A., ICML'20]

Preamble: Mixup data augmentation [Zhang et al 18]

Idea : teach deep models to behave linearly on training data

- Images are vectors in $[-1,1]^d$, labels are 1-hot vectors in $\{0,1\}^c$, where c = # of classes

- $\lambda \in_{R} (0,1)$, mixed image: $\lambda x_1 + (1-\lambda)x_2$, mixed label: $\lambda y_1 + (1-\lambda)y_2$

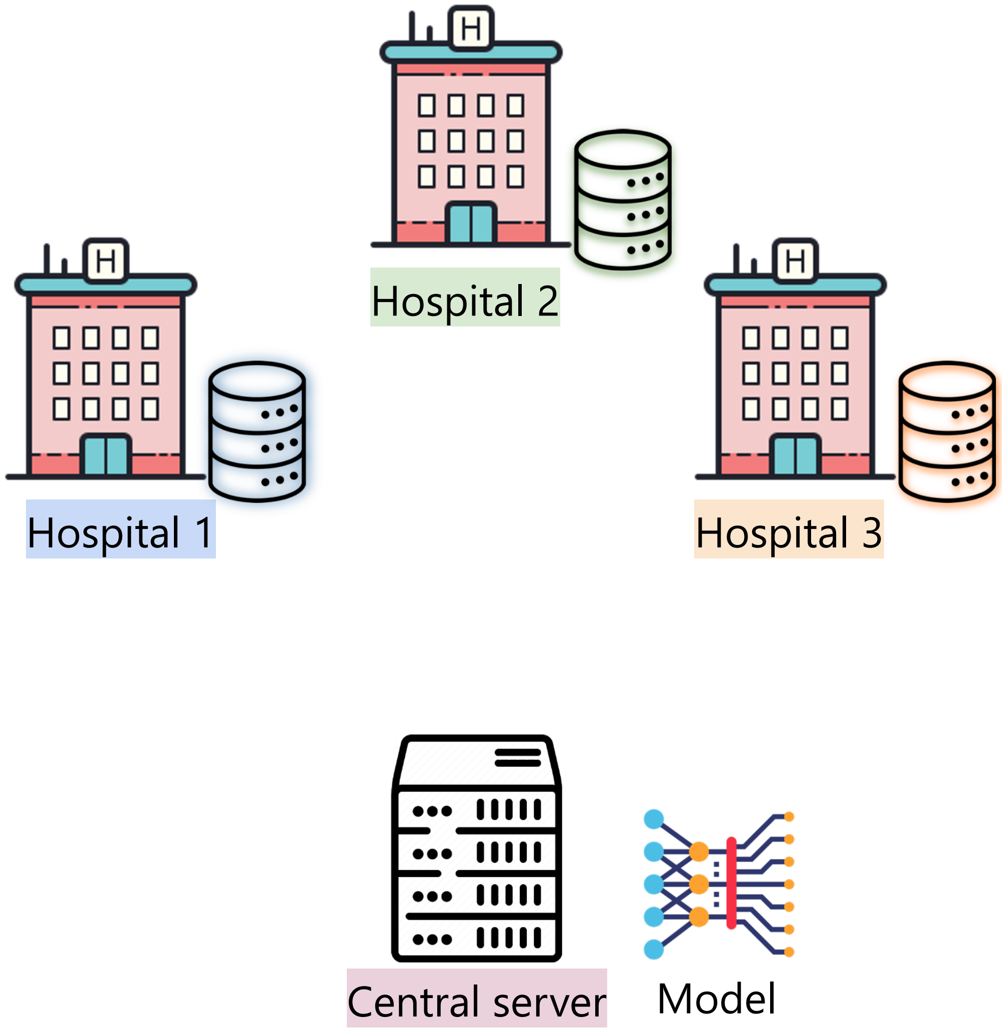
0.6 x  + 0.4 x  = 

(0, 1, 0, 0) Cat **(0, 0, 0, 1)** Car **(0, 0.6, 0, 0.4)** Cat Car

Takeaway: Training data is malleable!

Training with only these mixed data points gives better final accuracy on normal images.

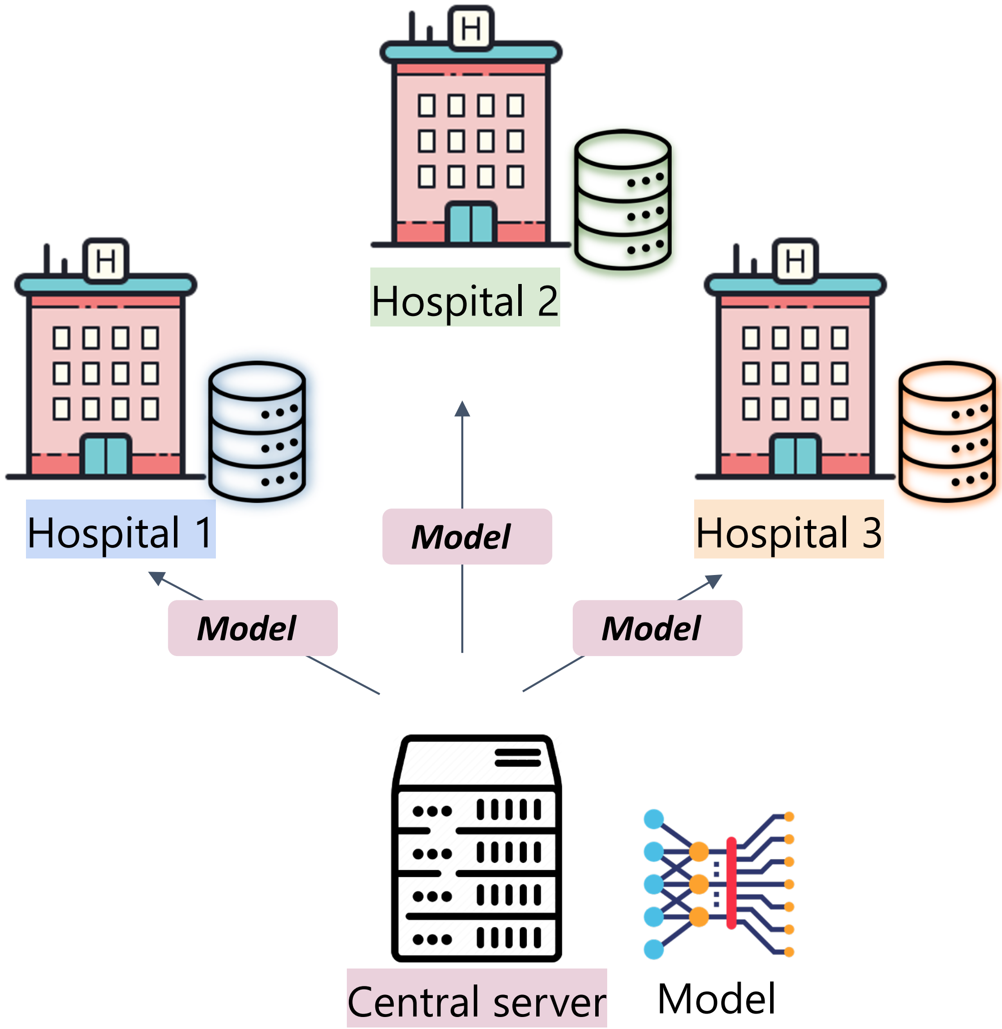
Federated learning with private data



Multiple parties with private data (e.g. hospitals) want to collaboratively train a deep model.

Federated learning [McMahan et al 16]: Server shares current model with the parties. They share model updates (gradients) using their data.

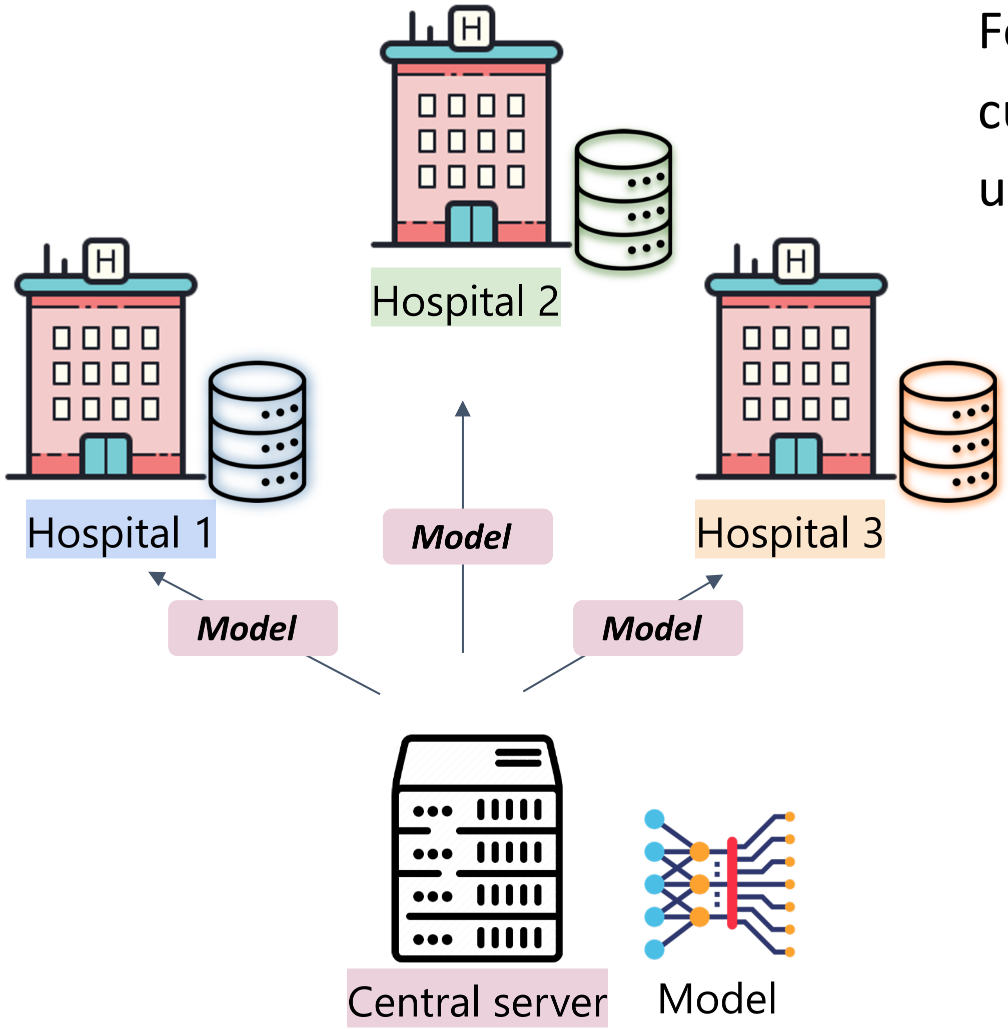
Federated learning with private data



Multiple parties with private data (e.g. hospitals) want to collaboratively train a deep model.

Federated learning [McMahan et al 16]: Server shares current model with the parties. They share model updates (gradients) using their data.

Private Distributed Learning



Federated learning [McMahan et al 16]: Server shares current model with the parties. They share model updates (gradients) using their data.

Approach 1: **Differential privacy** (each party shares model gradients computed using their data, but after adding noise ("DP").
Pros: Provable Privacy guarantees
Cons: Large accuracy drop due to added noise.

Approach 2: **Secure Multiparty Computation using cryptography. (Yao'82, BGW'87)**
Pros: Strongest privacy guarantees.
Cons: High computational overhead; infeasible for modern deep learning.

Needed: An encryption method for data that does not interfere with deep learning

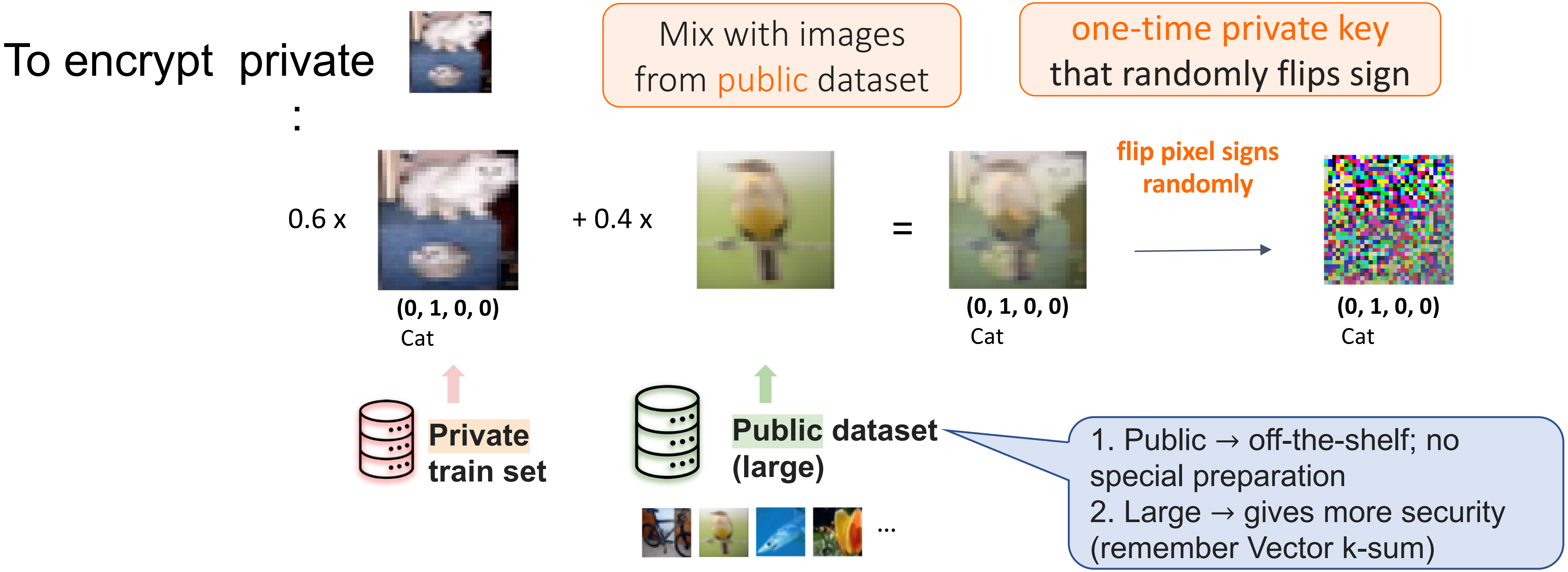
(Usual crypto lifts arithmetic operations to finite fields or lattices)

Take inspiration from Mixup??

Inspiration : Simple addition on datapoints can help to obscure them.

- k-VECTOR SUBSET SUM [Bhattacharyya et al 11]:
    - A set of public N vectors $v_1, \cdots, v_N \in \mathbb{R}^d$
    - Picks k secret indices $i_1, \cdots, i_k \in \{1,..N\}$ and releases $\sum_j v_{ij}$
    - Exponential Time Hypothesis → finding $i_1, \cdots, i_k$ requires $\geq N^{k/2}$ time [Abboud and Lewi 13]
        - k = 4 is already pretty hard

InstaHide: Idea

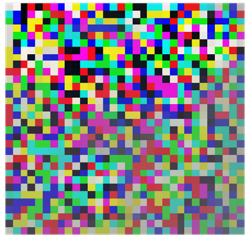


InstaHide: Full description (think of k as 4)

Mix k/2 training images with k/2 public images, followed by pixelwise random sign flip

$\lambda_1 \times$  $+ \lambda_2 \times$  $+ \lambda_3 \times$  $+ \lambda_4 \times$ 

flip pixel signs randomly →



(0, 1, 0, 0)
Cat

(0, 0, 0, 1)
Car

$(0, \lambda_1, 0, \lambda_2)$
Cat Car

Conjecture: Extracting any information about training images requires > min $\{N^{\frac{k}{2}}, 2^d\}$ time
(N = size of public dataset, d= # pixels)

Note: Secret key for encryption = (Choice of images used for mixing, random sign mask)
Never reused during training

Blog: www.offconvex.org, Twitter: @prfsanjeevarora

Concluding thoughts

- Understanding why and how deep learning works is a new frontier for mathematics.

- Attempts to "open the black box" leads to new insights and new methods. (e.g., exponentially increasing learning rates, InstaHide)

- It will be a fun ride!

THANK YOU!!

In der Mathematik gibt es kein ignorabimus
D. Hilbert