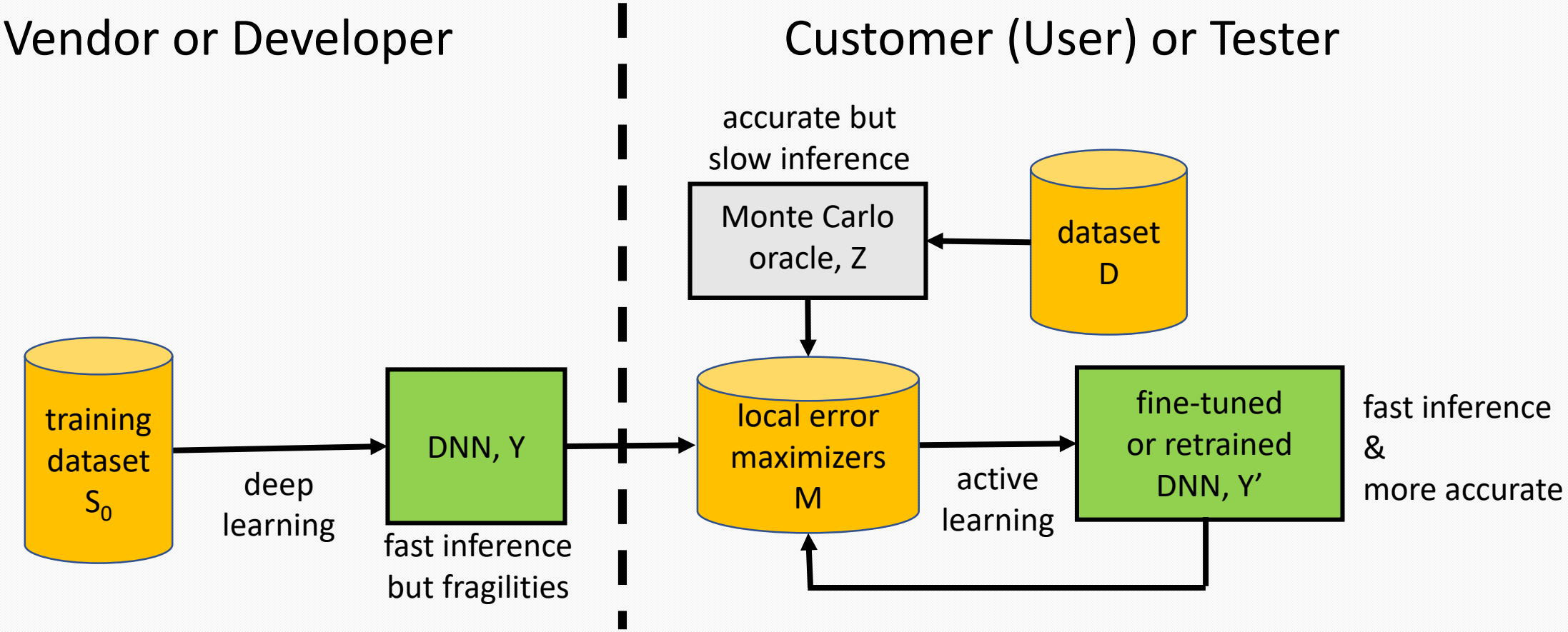
David J. Miller
Zhen Xiang
George Kesidis

Adversarial Learning and Secure AI

CAMBRIDGE
UNIVERSITY PRESS & ASSESSMENT

Chapter 12

Robust Deep Regression and Active Learning

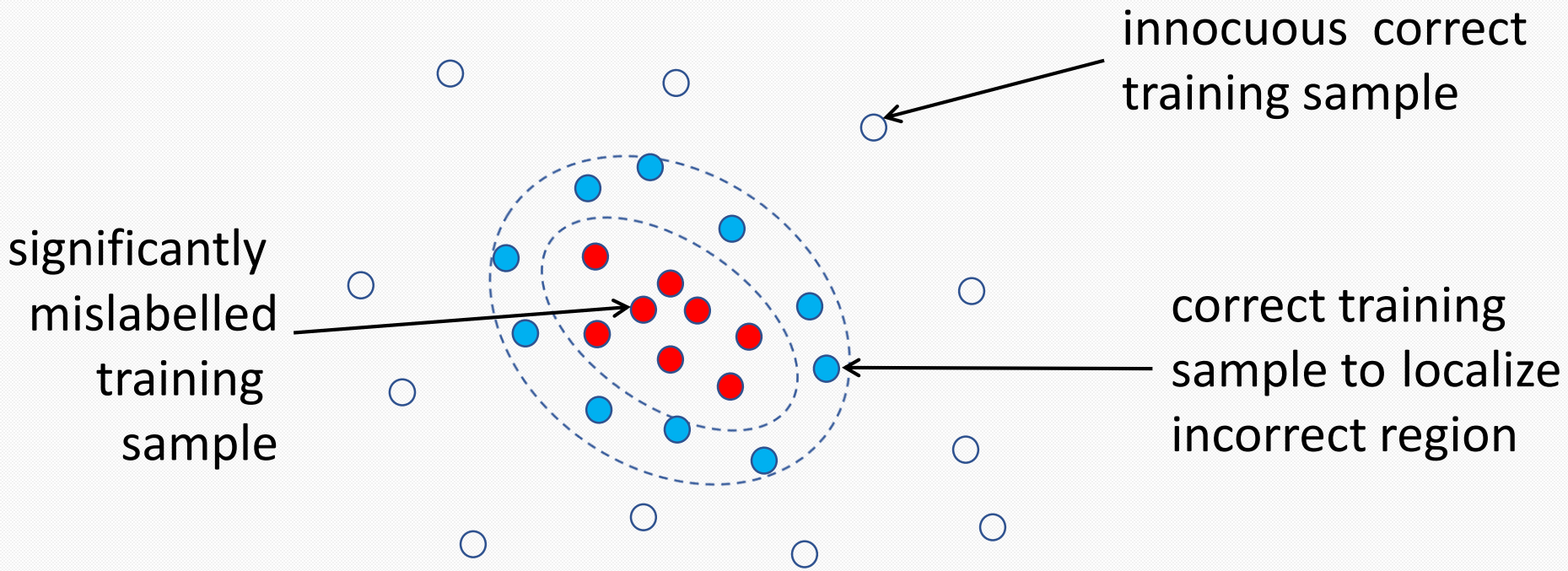


- Dataset S_0 may not be available during the active learning (or just testing) process.
- Objective is to detect (testing) and address (active learning - AL) any "hidden fragilities" in the DNN Y.
- e.g., a possibly highly localized region of input space where Y produces incorrect answers.

Small Error Regions in Deep Regression

- A regression model may generalize poorly in some region of input space, i.e. where the DNN Y incurs significant error.
- This could be an **artificial** worst-case benchmark flaw chosen by the designer to calibrate the testing or active-learning process of the DNN.
- But this could also be caused by inadequate training data, model bias, by "model drift" over time, or
- by deliberate poisoning of S_0 by an adversary (an insider or through outsourcing).
- In the latter case a backdoor is planted in the DNN Y, involving:
  - consistent mislabeling in the backdoor region, e.g., labelled values are all 50% higher
  - at test time, adversary waits until the backdoor region is naturally reached, or
  - could manipulate an input (e.g., volatility) to move an input x into the backdoor region.

A worst-case testing benchmark for AL: localized regional flaw in the initial DNN Y



- These regions may not be consistent with any theoretical model.
- Harder to detect when mislabelled region is very small and the region is “localized."
- Breadth of search may detect such “worst-case” regions.
- Fine-tuning by active learning may “tamp down” errors in such regions.

Exemplar AL method: Gradient ascent on squared error

- For some selected samples x_0:
- Gradient ascent on squared error $W(x)=|Y(x)-Z(x)|^2$ with step size h>0:

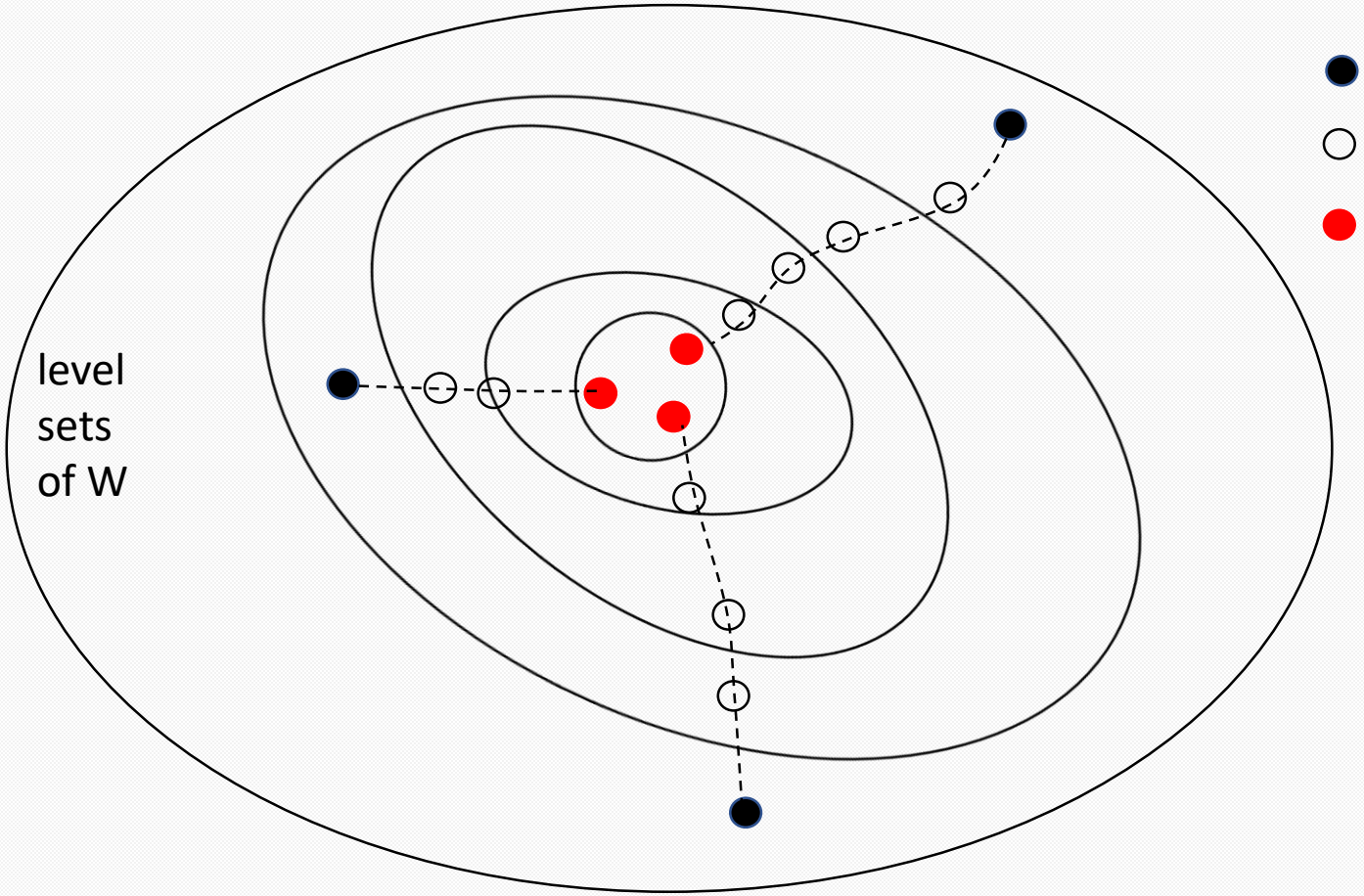
$$x_n = x_{n-1} + h\,(Y(x_{n-1})-Z(x_{n-1}))(\nabla Y(x_{n-1}) - \nabla Z(x_{n-1}))\ ,\ \text{where}$$

  - ∇Y is computed by back-propagation, and
  - ∇Z is approximated by a finite difference-quotient, e.g.,

$$\nabla Z_k\,(x) \approx (Z(x+\delta e_k)-Z(x))/\delta \quad \text{with } e_k = k^{th} \text{ std unit vector (2 oracle evaluations here)}$$

- Stopping condition: $|W(x_n) - W(x_{n-1})| < \sigma W(x_n)$
- To reduce oracle labellings:
  - In each AL stage, line search for (fixed) h to reduce number of grad ascent steps.
  - Also given some proximal labelled examples, can interpolate Z by differentiable Z^* and then use closed-form ∇Z^* in subsequent steps of gradient ascent in the same vicinity.

Identify unique local error maximizers (LEMs)



- seed point
- grad ascent step
- LEM

Assumption:
Distance between such "same" local maxima on the order of the sizes of final gradient ascent steps

Exemplar method to find unique LEMs

- Take two local maximizers x, ξ discovered by gradient ascent on W to be the same if

$$\|x - \xi\| < a \cdot \max\{ \|\Delta x\| , \|\Delta \xi\| \}$$

where Δx is the last step used to find x by gradient ascent.

- Obviously, larger a means more aggressive clustering of LEMs.

- This clustering is important to understand how many significant LEMs are known at any given AL stage.

Iterative LEM refinement (Cuckoo Search) over multiple AL stages

- Search refinement in subsequent AL stages to climb higher on the "hills" of squared error, W.
- Beginning with unique LEMs of W, reduce difference quotient parameter δ, run gradient ascents, and identify new unique LEMs.
- Augment existing training set with the identified LEMs (among other samples) and retrain or fine-tune the DNN.
- Repeat

Simple AL rule combining breadth and depth of search: parameters β,D

1. Suppose L'≥0 unique LEMs and D' ≥0 randomly chosen samples are carried forward from the previous AL stage to the current one.
2. Choose βD samples as seeds to find new LEMs in current stage - these may include some or all of the L' & D' (with worst error), and newly selected random samples as needed.
3. Choose (1- β)D samples independently & uniformly at random from feasible input space - these may include any of the D' samples not used as seeds for new LEMs in the current stage.
4. (1- β)D random samples + L unique new LEMs used to retrain or fine tune the DNN.
5. Next stage's D' = (1- β)D and L'=L.

- So, β is like the inverse-temperature in annealing based search.
- In the iterative retraining process, the parameter β may be periodically adjusted.
- For example, if after 5 re-trainings, no new significant LEM regions are found, then decrease β (more search depth).

A single-barrier financial option experiment

- $Z(x) \geq 0$ is true expected present value of a "down-and-out put" option, where
- x = (barrier/spot(0), strike/spot(0), time to maturity, interest rate, volatility)
  $= (B,K,T,R,V) \in (\mathbb{R}^+)^5$ with B<K and 0= inception time
- $Z(x) = (1+R)^{-T} E[(\text{strike} - \text{spot}(T))^+ \mathbf{1}\{\text{spot}(t) > \text{barrier } \forall t < T\}]$
- R,V may be produced by other (dynamically adapted) neural networks based on multiple market signals.
- At least one Monte Carlo based "oracle" that computes Z(x).
- Train a regressor neural network Y to approximate Z but with much faster inference.

Training the AI: Single-barrier option example

- Consider a ``large'' DNN, Y, consisting of 5 layers, respectively, with 128, 256, 512, 256, 128 ReLU neurons.
- Input is normalized to $[0,1]^5$, but output not normalized with dimension [dollars].
- Other training details: no dropout, ADAM (no SGD), learning rate .001 (÷10 every 100 epochs), deep learning halted when the normalized change in training MSE loss <0.001 over 10 epochs or when 300 epochs reached.
- The training dataset S_0 has 200k oracle labelled samples, possibly with an additional cluster of mislabelled and localizing correctly labelled samples.

Backdoor/error region assessment after initial training on S_0:
200k total (non-attack) training samples with normalized features;
mislabelled 1.5Z for $0.9<B<K<1$, $T,R \in [0.45,0.55]$, $V \in [0.15,0.25]$;
localizing clean for $0.9<B<K<1$, $T,R \in [.40,.45] \cup [.55,.60]$, $V \in [.10,.15] \cup [.25,.30]$;
performance on 10k test samples (last column uses 10k samples in attack region)

number mislabelled	number localizing	training MSE	training MAE [dollars]	test MSE	test MAE [dollars]	1.4<y/z<1.6 (attack region)
0	0	0.0234	0.1061	0.0307	0.1103	0.0042
2000	0	0.0286	0.1133	0.0399	0.1186	0.9702
4000	0	0.0334	0.1189	0.0396	0.1221	0.9822
2000	2000	0.0344	0.1193	0.0369	0.1244	0.876
2000	8000	0.0241	0.1047	0.0315	0.1134	0.9822
4000	2000	0.0263	0.1094	0.0395	0.1131	0.976
4000	4000	0.0239	0.1044	0.0371	0.1131	0.983
4000	8000	0.0236	0.1017	0.0366	0.1087	0.9723

Detecting localized mislabelled regions by independent, uniform sampling

- If a backdoor region by volume is a fraction ε of the feasible input space, the probability of selecting a seed in this region after N' random samples is obviously

$$1-(1-\varepsilon)^{N'} = \Phi$$

- That is, $N' = \log(1-\Phi) / \log(1-\varepsilon)$
- For previous example, $\varepsilon = 10^{-5}$

DNN trained with backdoor region via 2k mislabelled & 8k localizing clean training samples;
D=1000 randomly selected samples s.t. none in backdoor region;
randomly chose fraction β as seeds for LEMs; 10 trials for each β.
before/after fine tuning (last 2 columns are 10k samples all in backdoor region):

β	N = mean total labellings	Φ = prob detect	N'	test MSE after	test MAE after	LEM MAE before	LEM MAE after	backdoor region MSE after	backdoor region MAE after
0	1000	.01	1000	0.0331	0.1130	-	-	19.69	3.973
0.25	2681	0.2	22314	0.0333	0.1134	0.1313	0.1234	19.31	3.926
0.5	4303	0.6	91628	0.0311	0.1131	0.1226	0.1173	18.84	3.869
0.75	6322	0.5	69314	0.0313	0.1137	0.1241	0.1192	19.01	3.903
1	8245	0.1	10535	0.0311	0.1141	0.1294	0.1211	19.65	3.962
before				0.0315	0.1134			20.12	4.012

- Note that D is just 5% of size of initial training set S_0
- Detection for β>0 trial if any LEM lands in misclassified backdoor region
- N = (1- β)D + labellings for LEMs from βD seeds including for initial step size chosen by line search
- sample std dev of total labellings is <5% of N
- sample std dev of test SE after is <3% of MSE

Discussion: Backdoors in control policies

- DNNs can also be used to implement control policies, e.g., offline deep RL by Q-learning on time-series datasets.
- The DNN maps an input state to a control action (policy) which results in a state-transition and a (possibly noisy) reward increment (as in MDPs).
- One can search the (state, action) space of the Deep Q Network (DQN) for “error” regions (i.e., those giving highly suboptimal rewards or transitioning to undesirable states).
- Such error regions could be deliberately caused by an adversary poisoning the traces used by (offline) RL (e.g., for Deep Q Learning).

Discussion: Online Vigilance and Robust Adaptation by Out of Distribution Detection (OODD)

- During test/operation time between retraining epochs:
- OODD as part of more frequent, active-learning based model refinement,
  - e.g., to address biased training set or model drift.
  - Avoid refinement based on all observed test samples as this may bias the model.
- OODD to detect adversarial inputs (TTEs, Chapter 4) or RE probes (Chapter 14), optionally in a class conditional fashion.
- OODD on (sample, inference) pairs for regression or (state, action, next-state) triplets for policy optimization.
- OODD can be based on deep generative models (reconstruction loss) or traditional null models (p-values) of embedded features.
- Detection of backdoor triggers using (offline) backdoor detection or mitigation, see Chapter 10.