COMS E6998 010

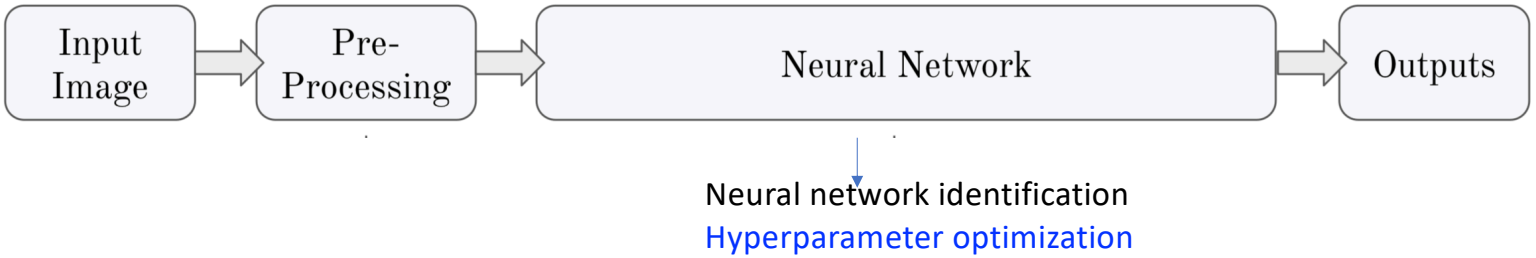
Practical Deep Learning Systems Performance

Lecture 13 12/10/20

Recall from last lecture

- Automated Machine Learning
- Neural Architecture Search (NAS) strategies based on evolutionary algorithms and reinforcement learning, differentiable architecture search
- Train-less accuracy predictor for architecture search (TAPAS)
- NeuNetS, a neural network synthesis system from IBM

Automated Machine Learning



- Data Preprocessing: normalization, data-augmentation
- Neural architecture search (NAS)
  - Standard, off the shelf
  - Synthesize a new
    - Types of layers (conv, maxpool), number of different layers, how to stack the layers
    - Convolution layer parameters (filter dim, stride dim)
- Hyperparameter optimization
  - Batch size, learning rate, momentum
- NAS and hyperparameter optimization can be done jointly or sequentially

Automated Machine Learning

- IBM Auto AI
- Run a sample AutoAI experiment in IBM WML
- H20 AutoML

Hyperparameter Optimization

Hyperparameter optimization

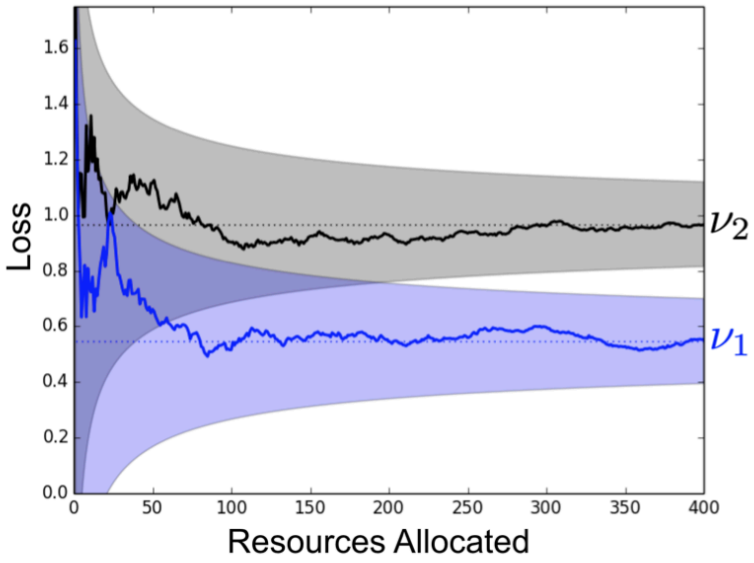
- Problem of identifying good hyperparameter configuration(s) from the set of possible configurations
- Two sub-problems
  - Configuration selection
    - Efficient selection of good configuration
  - Configuration evaluation
    - Adaptive computation, allocating more resources to promising hyperparameter configurations while eliminating poor ones.
- Several techniques:
  - Random search
  - Bayesian optimization methods
    - Focus on configuration selection
  - Bayesian optimization with adaptive resource allocation
  - Hyperband : random configuration search with adaptive resource allocation

Training resources

- Size of training set
- Number of features
- Number of iterations for iterative algorithms
- Hours of training time

Validation loss vs Resource allocated

Validation loss as a function of total resources allocated for two configurations with terminal validation losses ν_1 and ν_2



The shaded areas bound the maximum distance of the intermediate losses from the terminal validation loss and monotonically decrease with the resource.

Possible to distinguish between the two configurations when the envelopes no longer overlap

More resources are needed to differentiate between the two configurations when either
(1) the envelope functions are wider or
(2) the terminal losses are closer together.

Successive Halving

- Underlying principle: Even if performance after a small number of iterations is very unrepresentative of the *absolute* performance of the , its *relative* performance compared with many alternatives trained with the same number of iterations is roughly maintained.
  - Relative ordering among configurations converges much faster than their true values
- Uniformly allocate a budget to a set of hyperparameter configurations, evaluate the performance of all configurations, throw out the worst half, and repeat until one configuration remains
- Allocates exponentially more resources to more promising configurations
- Given some finite budget B and n configurations, it is not clear a priori whether we should
  - Consider many configurations (large n) with a small average training resources; or
  - Consider a small number of configurations (small n) with longer average training resources.
- Successive Halving suffers from the "*n vs B/n*" trade-off

Li et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2018

n Vs. B/n

- Consider a simple strategy
  - If hyper-parameter configurations can be discriminated quickly
    - *n* should be chosen large
  - If hyper-parameter configurations are slow to differentiate
    - *B/n* should be large
- Drawbacks of the simple strategy
  - If *n* is large, then some good configurations which can be slow to converge at the beginning will be killed off early.
  - If *B/n* is large, then bad configurations will be given a lot of resources, even though they could have been stopped before.

Hyperband

- Formulating hyperparameter optimization as a pure-exploration adaptive resource allocation problem addressing *how to allocate resources among randomly sampled hyperparameter configurations*.
- *Considers several possible values of n for a fixed B, in essence performing a grid search over feasible value of n*
- Hedges and loops over varying degrees of the aggressiveness balancing breadth versus depth based search.
- Resource constrained hyperparameter optimization

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  `// begin SUCCESSIVEHALVING with` (n, r) `inner loop`
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Hyperband in action

Each bracket uses B total resources and corresponds to a different tradeoff of n and B/n

```
max_iter = 81
eta = 3
B = 5*max_iter
```

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

random search

A larger value of n corresponds to a smaller r and hence more aggressive early-stopping

A single execution of Hyperband takes a finite budget of $(s_{max}+1)B$

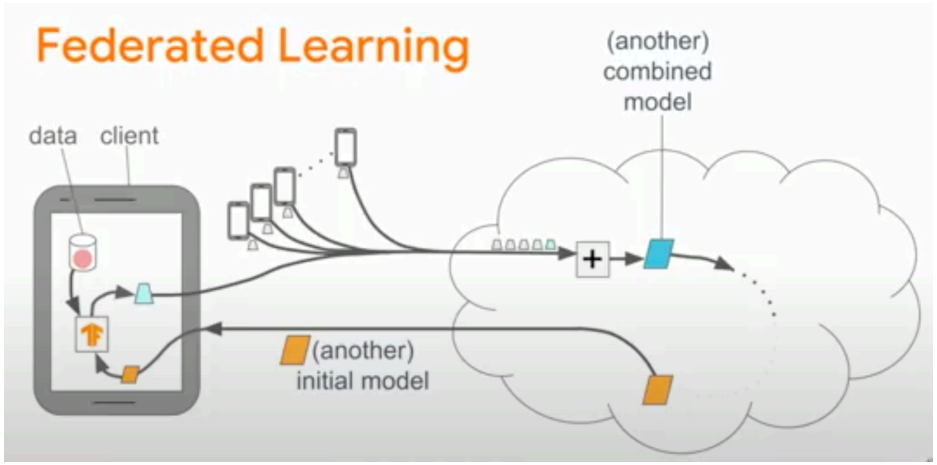
Kubeflow Katib

- K8S based system for hyperparameter optimization and NAS
- https://github.com/kubeflow/katib
- https://www.kubeflow.org/docs/components/hyperparameter-tuning/overview/
- https://www.kubeflow.org/docs/components/hyperparameter-tuning/hyperparameter/

Federated Learning

Federated Learning

- Learn together without sharing data



- “bringing the code to the data, instead of the data to the code”
- Addresses the fundamental problems of privacy, ownership, and locality of data

Federated Learning Challenges

- *Is FL another name for distributed DL ?*
- **Non-IID:** The training data on a given client is typically based on the usage of the mobile device by a particular user, and hence any particular user's local dataset will not be representative of the population distribution.
- **Unbalanced:** Similarly, some users will make much heavier use of the service or app than others, leading to varying amounts of local training data.
- **Massively distributed:** We expect the number of clients participating in an optimization to be much larger than the average number of examples per client.
- **Limited communication:** Mobile devices are frequently offline or on slow or expensive connections.

Federated Averaging Algorithm

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ **do**

$m \leftarrow \max(C \cdot K, 1)$

$S_t \leftarrow$ (random set of m clients)

for each client $k \in S_t$ **in parallel do**

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*

$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server

$$n_k = |\mathcal{P}_k|$$

McMahan et al. Communication-efficient learning of deep networks from decentralized data. AAAI 2017

Federated Learning Protocol

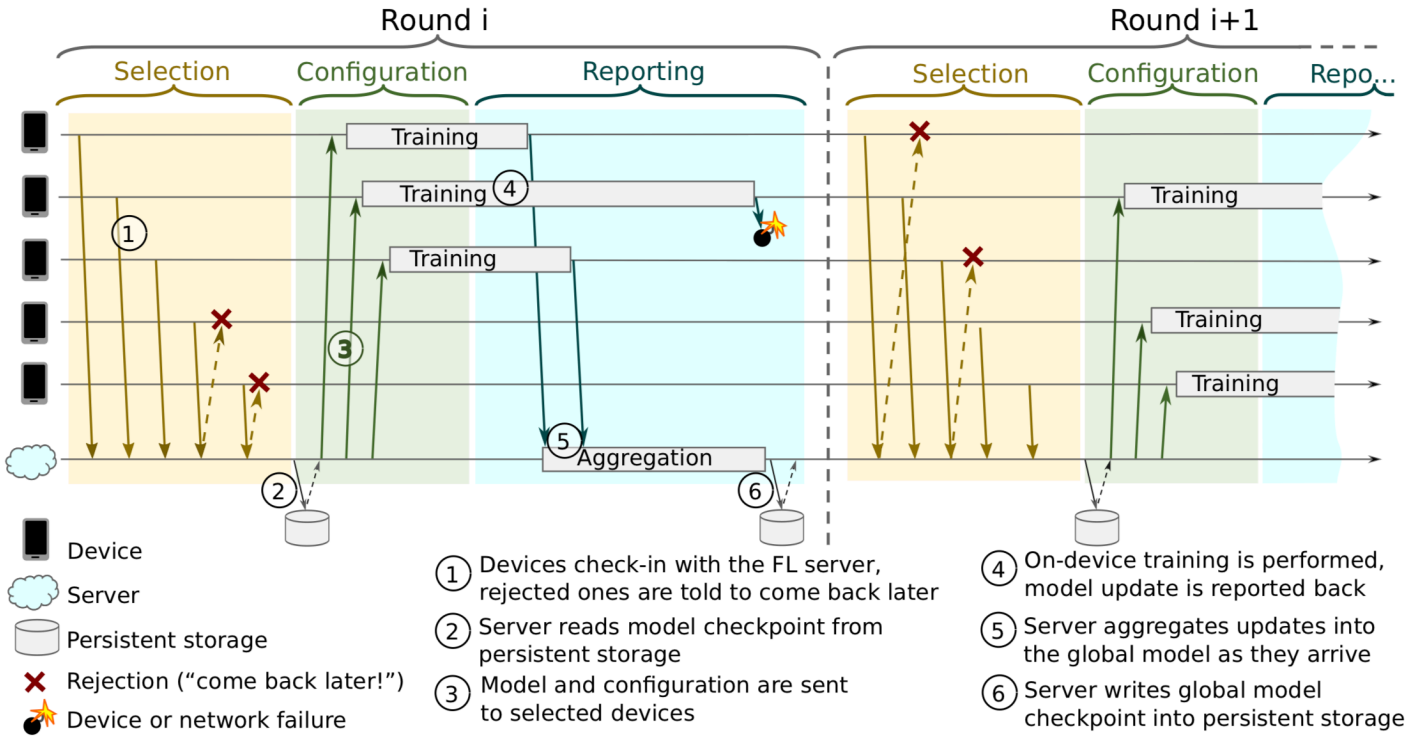


Figure 1: Federated Learning Protocol

Bonawitz et al. Towards federated learning at scale: System design. SysML 2019

Device Architecture for FL

Job only runs when phone is idle, charging, and connected to a free WiFI

Repository of locally collected data for model training and evaluation

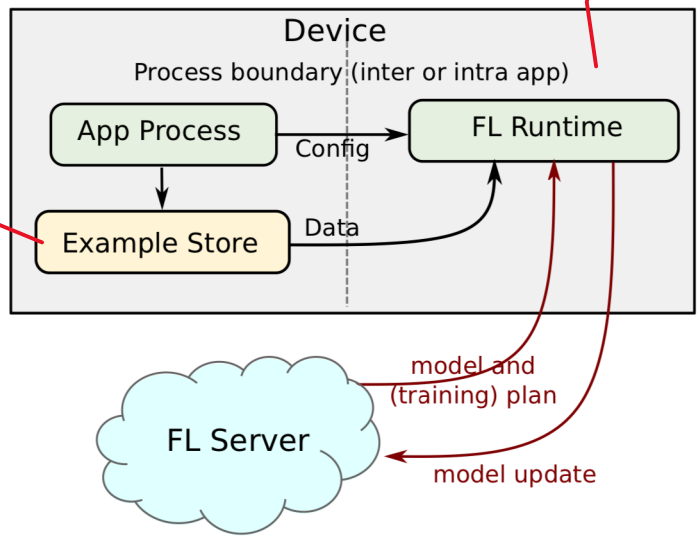


Figure 2: Device Architecture

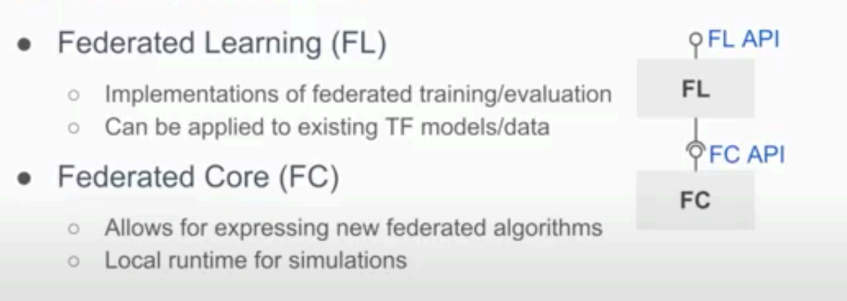
Control flow steps

- Programmatic configuration
  - Application configures the FL runtime by providing an FL population name and registering its example stores
  - Periodic FL runtime job is scheduled at the device
- Job invocation
  - FL runtime contacts the FL server to announce that it is ready to run tasks
- Task execution
  - If selected, the FL runtime receives the FL plan, queries the app's example store for data requested by the plan, and computes plan-determined model updates and metrics.
- Reporting
  - Computer model updates are reported to the FL server

Bonawitz et al. Towards federated learning at scale: System design. SysML 2019

TensorFlow Federated (TFF)

- TFF is an OSS framework for federated learning on decentralized data
- TFF enables users to use their own model and data for
  - Doing simulations with standard FL algorithms (supported by TFF)
  - Experimenting with novel algorithms for FL
- 2-layers of TFF Interfaces



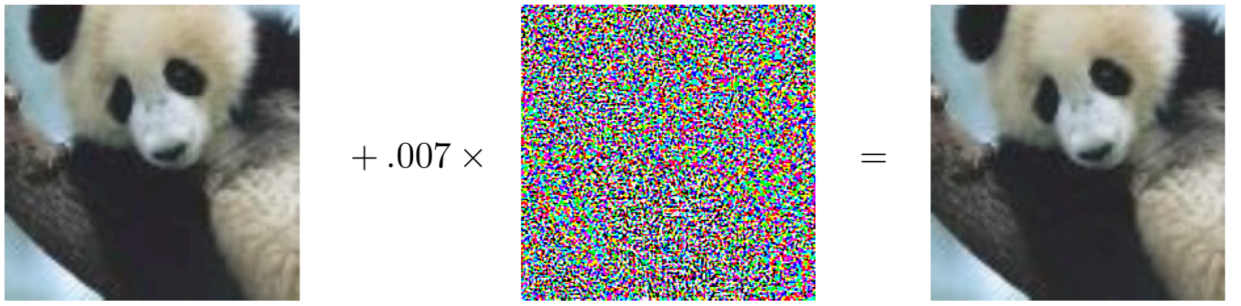
- TFF tutorials: https://www.tensorflow.org/federated/tutorials/
- TFF github repo: https://github.com/tensorflow/federated

FL Benchmarking

- https://arxiv.org/pdf/1812.01097.pdf
- Project page. LEAF: A benchmark for Federated Settings.

Adversarial Robustness

Brittleness of ML models



- ML models are extremely brittle
- Brittleness is a problem: security and safety implications

Goodfellow et al. Explaining and harnessing adversarial examples. 2014

Why are ML models brittle ?

- Artifact of the model training algorithm

$$\mathbb{E}_{(x,y)\sim D}\left[loss(\theta, x, y)\right]$$

- Research on adversarial ML is focused on two main questions:
  1. How can we produce strong adversarial examples, i.e., adversarial examples that fool a model with high confidence while requiring only a small perturbation ? (Adversarial attack generation)
  2. How can we train a model so that there are no adversarial examples, or at least so that an adversary cannot find them easily? (Adversarial training)
- Can we develop ML models robust to adversaries ?
  - Cannot be fooled by imperceptible perturbations to the input

Adversarial Training

- To create adversarially robust models
- Need to modify the model training objective

~~Standard~~ generalization: $\mathbb{E}_{(x,y)\sim D}\left[\max_{\boldsymbol{\delta}\in\boldsymbol{\Delta}} loss(\theta, x+\boldsymbol{\delta}, y)\right]$

Adversarially robust

- Train using adversarial examples
  - Delta needs to be imperceptible
- How to generate adversarial examples ?
  - How can we find perturbatrion which is imperceptible
- Bounding the LP-norm of the change of each individual pixel
- Train models to perform well in expectation on the worst case examples from this distribution

Adversarial Training Algorithm

- Adversarial training was developed to try to train models to be robust [Madry et al.].
- Composition of an *inner maximization* problem and an *outer minimization* problem

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right] .$$

- Computationally expensive and slow !

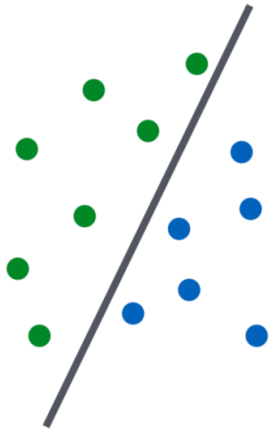
	Natural training	Adversarial training
ResNet 50	1.1 hours	6.8 hours
WideResNet 28x10	2.2 hours	14.7 hours

Table: Training time with 10 step adversary and with $\epsilon = \frac{8}{255}$.

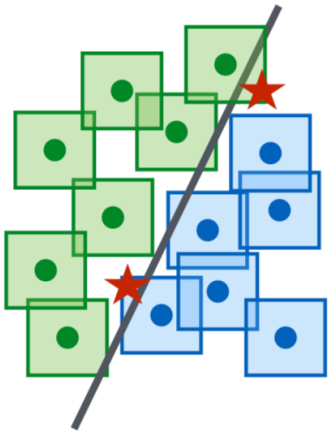
Madry et al. Towards deep learning models resistant to adversarial attacks. 2017

Adversarially trained model learns complex decision boundaries

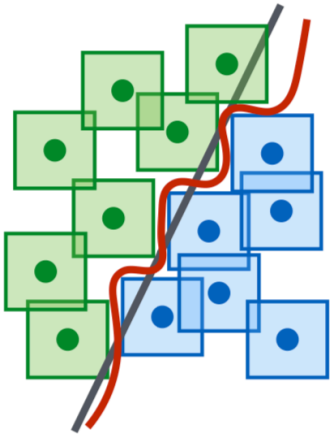
Linear decision boundary separates original training data



Linear decision boundary cannot separate separate the l_∞-balls (here, squares) around the data points



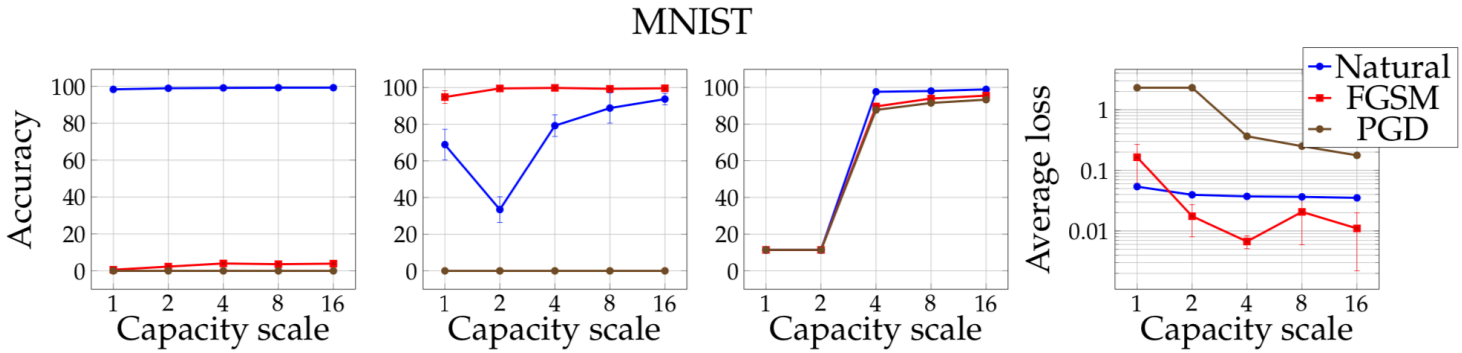
Classifier with a complicated decision boundary is robust to adversarial examples with bounded l_∞-norm perturbations



Adversarial robustness demands higher network capacity

Madry et al. Towards deep learning models resistant to adversarial attacks. 2017

Network Capacity and Adversarial Robustness



CIFAR10

	Simple	Wide
Natural	92.7%	95.2%
FGSM	27.5%	32.7%
PGD	0.8%	3.5%

(a) Standard training

	Simple	Wide
Natural	87.4%	90.3%
FGSM	90.9%	95.1%
PGD	0.0%	0.0%

(b) FGSM training

	Simple	Wide
Natural	79.4%	87.3%
FGSM	51.7%	56.1%
PGD	43.7%	45.8%

(c) PGD training

	Simple	Wide
Natural	0.00357	0.00371
FGSM	0.0115	0.00557
PGD	1.11	0.0218

(d) Training Loss

Madry et al. Towards deep learning models resistant to adversarial attacks. 2017

Delayed Adversarial Training (DAT)

Adversarial samples are a function of the network parameters which are changing.

Start with natural training. Switch to adversarial training after some number of epochs.

Parameters change a lot at the start of training and are far from their final state and so initial adversarial samples are not helpful. Initial samples may also be influencing the decision boundary when they are not important.

Delayed adversarial training: Introduce adversarial samples in the middle of training.

- Computational cost reduced because fewer adversaries are generated during training.
- Initial adversaries are removed from training which should help overfitting.

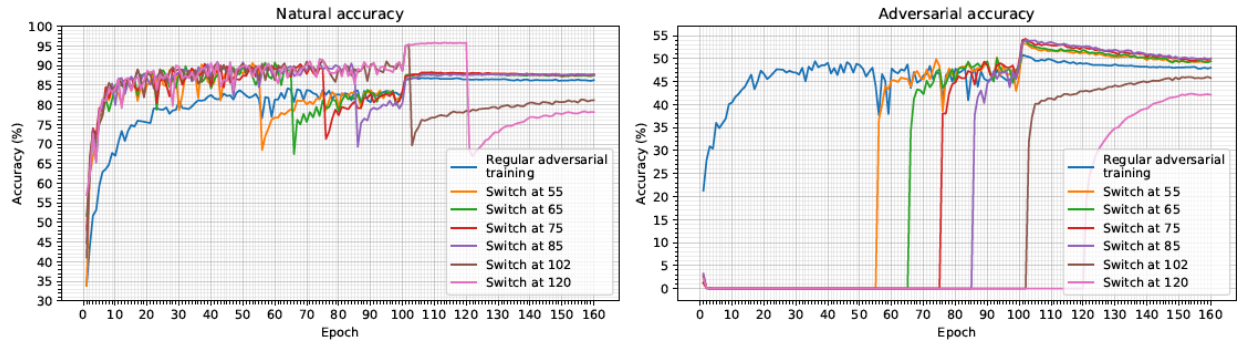


Figure 2. Natural and adversarial test accuracy during regular adversarial training and adversarial training with different switches. CIFAR-10 images are classified using the WideResNet-28x10. Adversarial samples with $T = 10, \epsilon = \frac{8}{255}$ and $\alpha = \frac{2}{255}$ are used. SGD learning rate drops are after epochs 100, 105 and 150.

Gupta et al. Improving the affordability of robustness training for DNNs. 2020

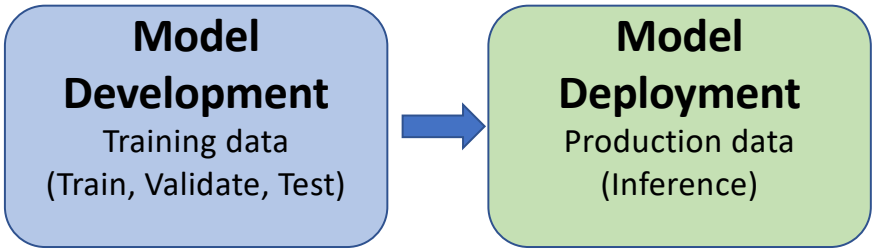
DAT Time Savings

		CIFAR-10			
		Training time	*Time saved*	$T = 10, \epsilon = 8/255$	*Natural accuracy*
WideResNet-28x10	*RAT*	14.7 HOURS	46.9%	48.5%	86.8%
	DAT	7.8 HOURS		49.7%	87.9%
	RAT early stop	10.9 HOURS	62.4%	49.2%	87.1%
	DAT early stop	4.1 HOURS		53.6%	87.9%
ResNet-50	*RAT*	6.8 HOURS	45.6%	41.0%	75.2%
	DAT	3.7 HOURS		41.1%	74.4%
	RAT early stop	5.0 HOURS	64.0%	42.3%	73.2%
	DAT early stop	1.8 HOURS		41.6%	72.2%
ResNet-18	*RAT*	2.5 HOURS	36.0%	37.0%	73.8%
	DAT	1.6 HOURS		40.4%	72.8%
	RAT early stop	1.9 HOURS	52.6%	40.6%	71.0%
	DAT early stop	0.9 HOURS		41.1%	69.9%
		CIFAR-100			
		Training time	*Time saved*	$T = 10, \epsilon = 8/255$	*Natural accuracy*
ResNet-50	*RAT*	6.9 HOURS	42.0%	15.2%	44.2%
	DAT	4.0 HOURS		15.2%	46.6%
Resnet-18	*RAT*	2.6 HOURS	46.2%	14.2%	44.7%
	DAT	1.4 HOURS		14.2%	46.7%
		MNIST			
		Training time	*Time saved*	$T = 40, \epsilon = 0.3$	*Natural accuracy*
Two-layer CNN	*RAT*	2.2 HOURS	13.6 %	91.4%	98.2%
	DAT	1.9 HOURS		91.9%	98.2%

Gupta et al. Improving the affordability of robustness training for DNNs. 2020

Model Drift and Detection

ML Model: Development to Deployment



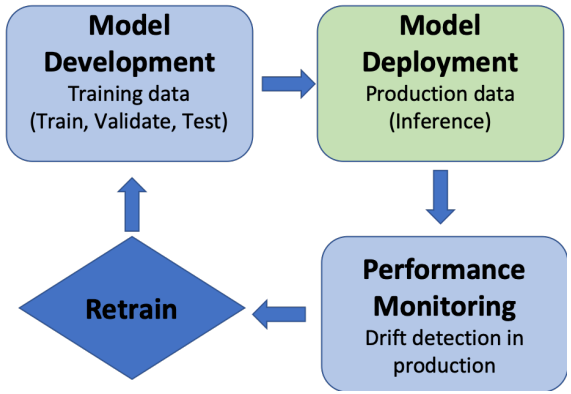
- The assumption underlying statistical ML
  - Training data distribution is representative of production data
  - Performance of ML model in production is same as predicted in training
- Will the model performance always remain same in production ?

Model Drift: Change in Model Performance Over Time

- Production data gradually changes over time
- Performance of ML model may deteriorate over time
- Causes
  - Data drift: change in distribution, emergence of new variety
    - Cars, fashion items, baby names
  - Concept drift: change in statistical properties of the target variable
    - Data does not change but assigned classes changes
    - Socioeconomic reasons, change in laws
- Not all data drift will cause model performance deterioration

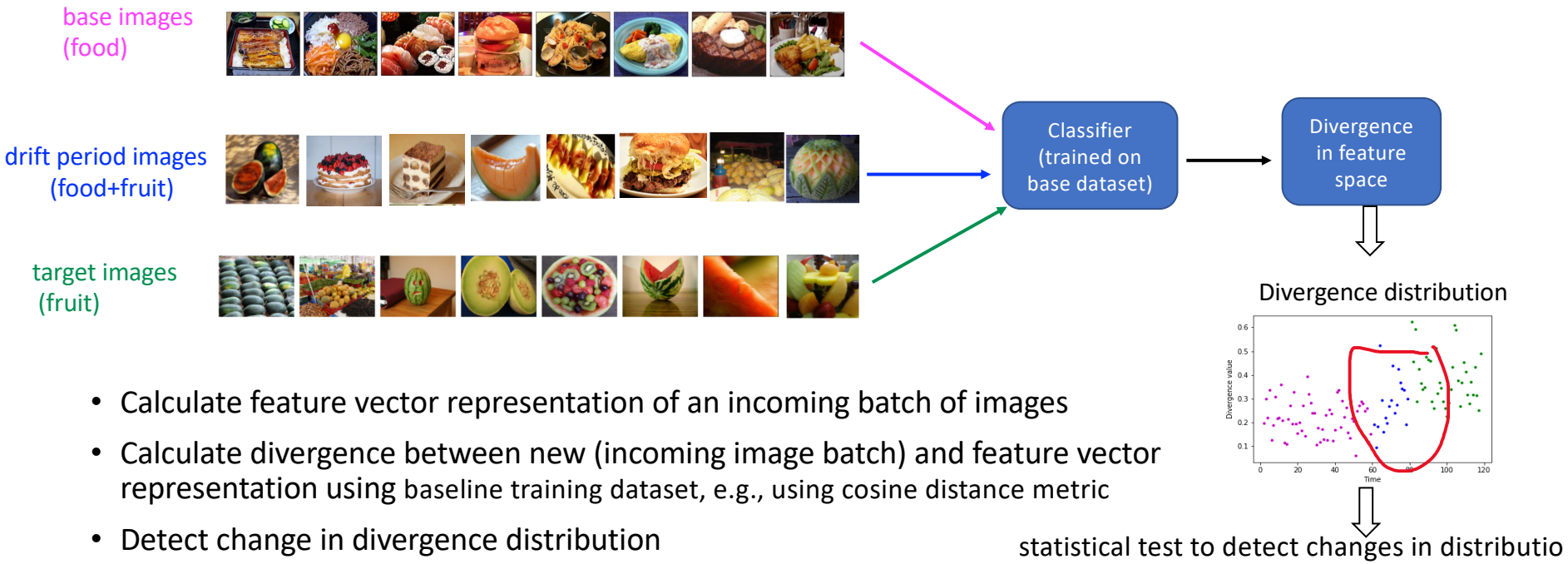
How to address data drift in ML ?

- Drift detection and retraining
  - Supervised drift detection: availability of labeled production data
    - Labeling is expensive and time consuming
  - Unsupervised drift detection: change detection in feature distribution of data
- Develop ML models robust to data drift



- *We are interested in detecting data drift in the absence of labels*
  - Detect changes in confidence distribution of classifier (Raz et al. 2019)
  - Exploit network structure and embeddings (Dube et al. 2020)

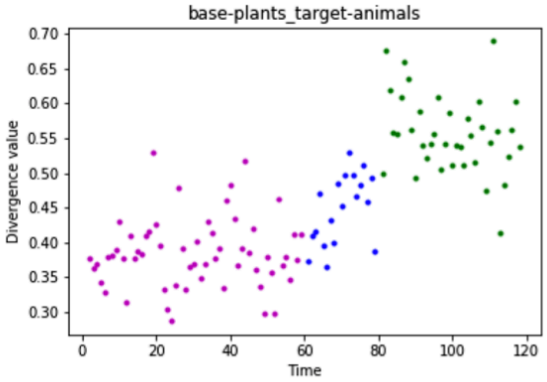
Drift Detection Methodology

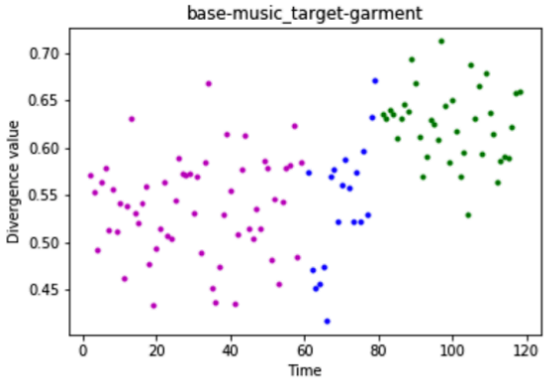


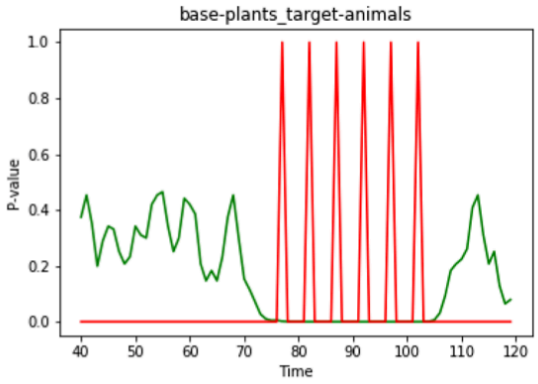
- Calculate feature vector representation of an incoming batch of images
- Calculate divergence between new (incoming image batch) and feature vector representation using baseline training dataset, e.g., using cosine distance metric
- Detect change in divergence distribution
  - Divergence distribution as a surrogate for production data distribution

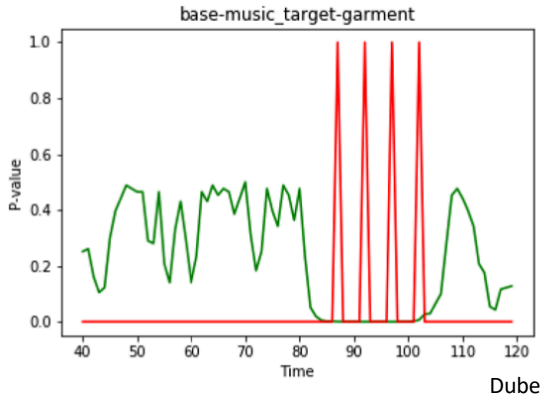
Dube et al.. Automated detection of drift in deep learning based classifiers performance using network emdeddings. EDSMLS 2020.

Algorithm at work: Successful detection









Drift starts at t = 60 and finishes at t = 80. At each epoch calculated P value is shown in green.

The epoch(s) when the algorithm detects the change are marked by a change in value of red curve from 0 to 1. When a change is detected the algorithm is reset ("found" parameter set to 0) resulting in multiple change detection during the drift period.

Dube et al.. Automated detection of drift in deep learning based classifiers performance using network emdeddings. EDSMLS 2020.

Reference Papers

- Hyperparameter optimization
  - Li et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2018
  - Li et al. A System for Massively Parallel Hyperparameter Tuning. 2020
- Federated Learning
  - McMahan et al. Communication-efficient learning of deep networks from decentralized data. AAAI 2017.
  - Bonawitz et al. Towards federated learning at scale: System design. SysML 2019.
  - Gupta et al. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 2018.
  - Caldas et al. LEAF: A Benchmark for Federated Settings. 2019

Reference Papers

- Adversarial Robustness
  - Goodfellow et al. Explaining and harnessing adversarial examples. 2014
  - Madry et al. Towards deep learning models resistant to adversarial attacks. 2017
  - Gupta et al. Improving the affordability of robustness training for DNNs. 2020
- Model Drift
  - Raz et al. Automatically detecting data drift in machine learning based classifiers. EDSMLS 2019.
  - Dube et al. Automated detection of drift in deep learning based classifiers performance using network emdeddings. EDSMLS 2020.

Reference Materials

- Federated Learning
  - McMahan et al. Collaborative machine learning without centralized training data. *Google AI Blog*. April 2017.
  - CVPR Tutorial on On Distributed Private Machine Learning for Computer Vision: Federated Learning, Split Learning and Beyond. 2019
  - TensorFlow project on Federated Learning. TensorFlow Federated: Machine Learning on Decentralized Data
  - Project page. LEAF: A benchmark for Federated Settings.
  - Split Learning Project. Distributed deep learning and inference without sharing raw data

Reference Materials

- Adversarial Robustness
  - Kolter and Madry. Adversarial Robustness: Theory and Practice.
- AI Explanability
  - IBM AI Explainability 360 Open Source Toolkit.
  - Google Explainable AI.
  - ALIBI: Open source python library aimed at ML models inspection and interpretation.