FROM RESEARCH TO INDUSTRY

cea tech

leti

Feature Extraction for Side-Channel Attacks

Eleonora Cagli

05/12/2018, Paris

PhD Supervisor : Emmanuel Prouff
(ANSSI)
CEA Supervisor : Cécile Dumas
(CEA-Leti Grenoble)

Contents

Secure Component and Embedded Cryptography

A piece of hardware with security properties.
It usually embeds cryptography to provide security services (authentication, signature, secure messaging with terminals...)

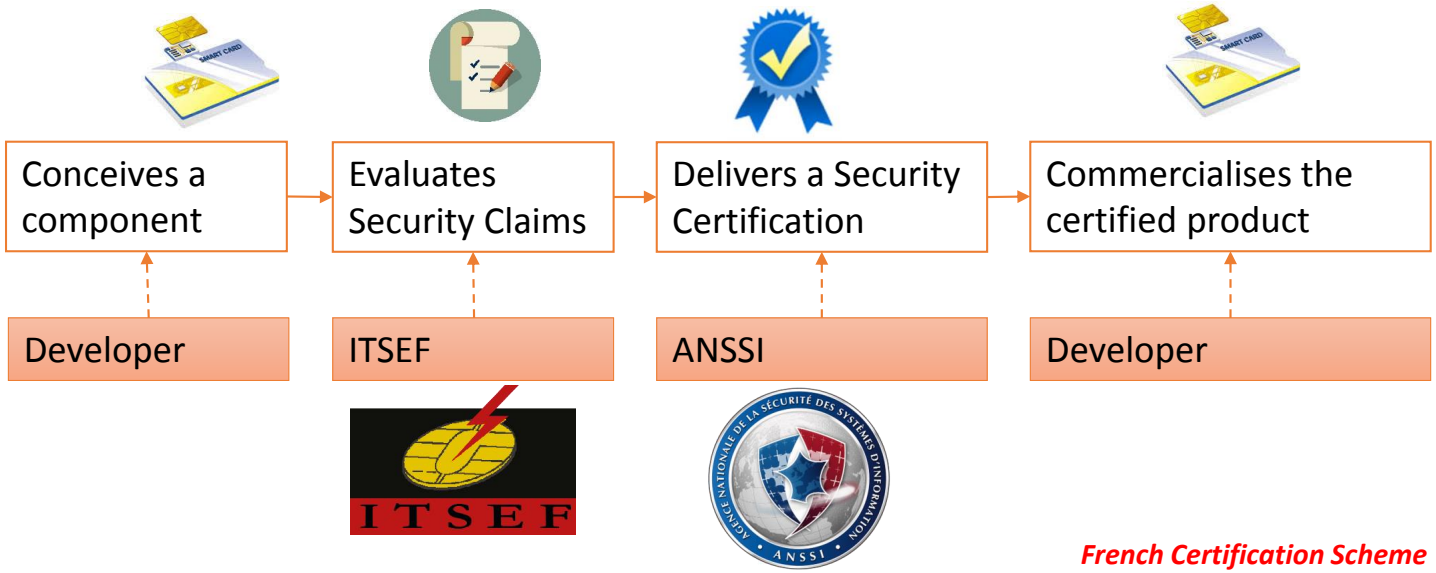
Secure Component and Embedded Cryptography

A piece of hardware with security properties.
It usually embeds cryptography to provide security services (authentication, signature, secure messaging with terminals...)

- Sensitive applications: ID cards, credit cards, transport cards, health cards, SIM
- Pervasive aspect: several billion smartcards sold par year
- Hard to update
- Hostile environment

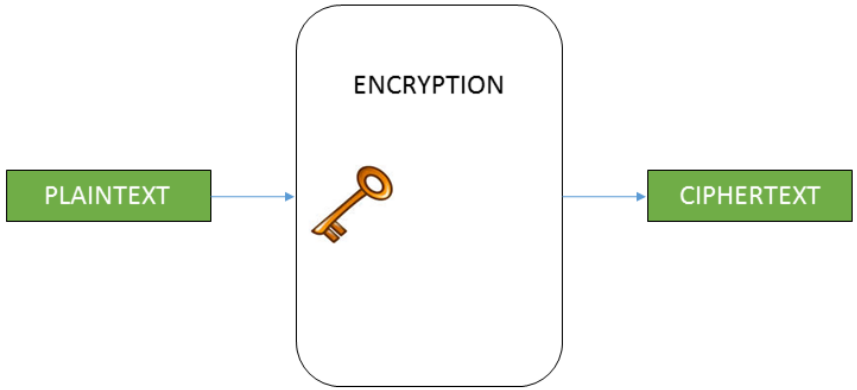
$\Rightarrow$ Requires protection against very high-level attacker

Security Certification



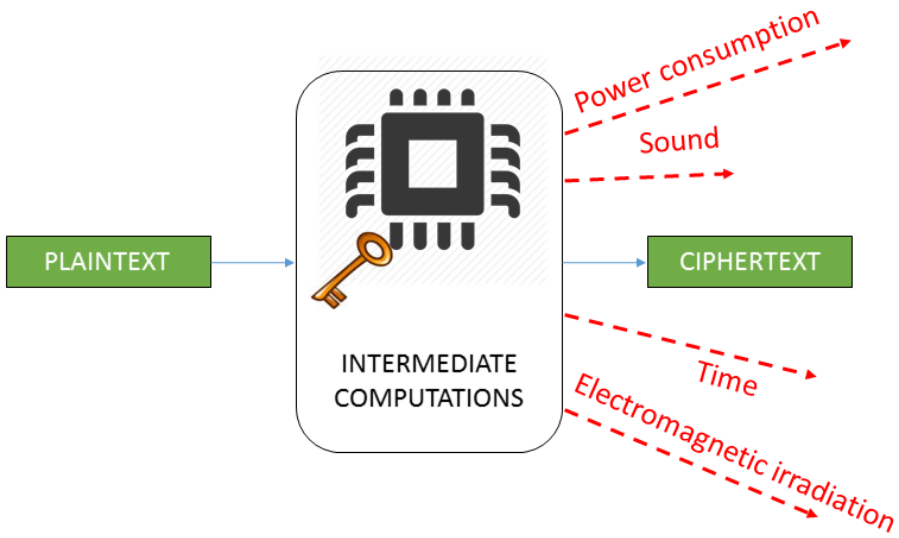
- Standardised Evaluation (*e.g.* ISO/IEC 15408 - Common Criteria)
- Assigns an Evaluation Assurance Level (EAL)
- The evaluator checks the Security Assurance Requirements (SAR), *e.g.* ADV, ALC, AVA, ...
- AVA: vulnerability assessment (penetration testing → attack potential rating)

Side-Channel Vulnerability of Embedded Cryptography



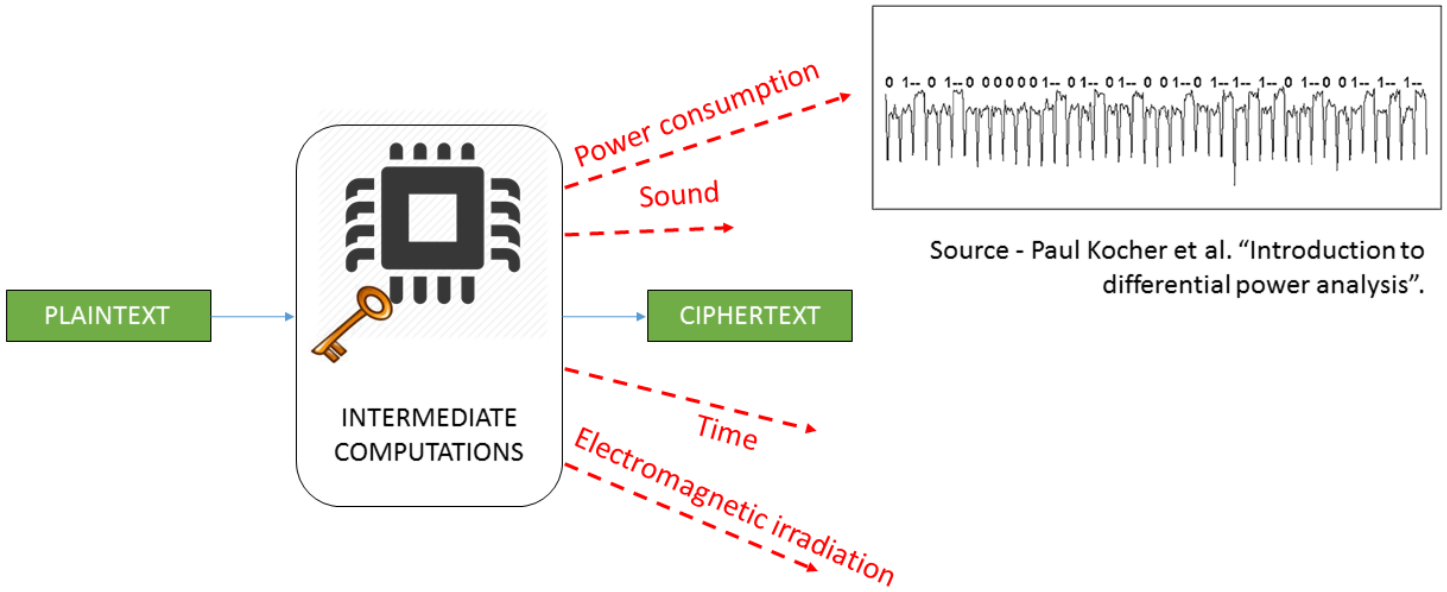
	Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability		
Black Box		

Side-Channel Vulnerability of Embedded Cryptography



Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

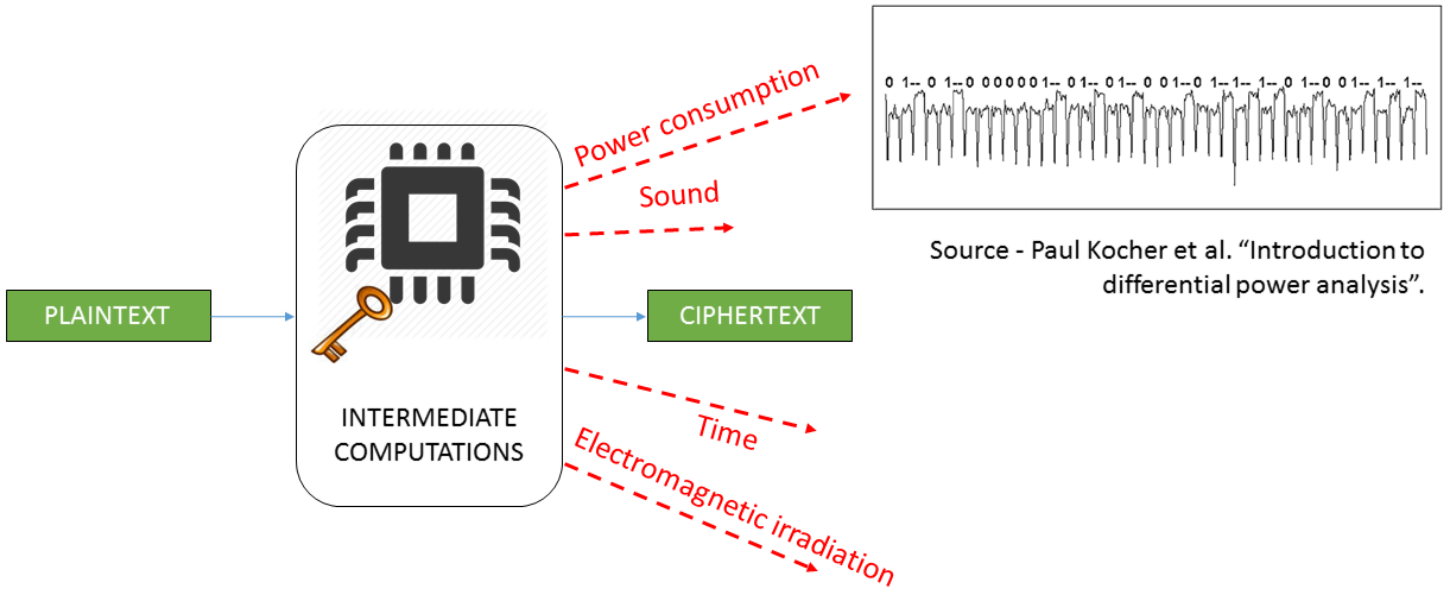
Side-Channel Vulnerability of Embedded Cryptography



Source - Paul Kocher et al. "Introduction to differential power analysis".

Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

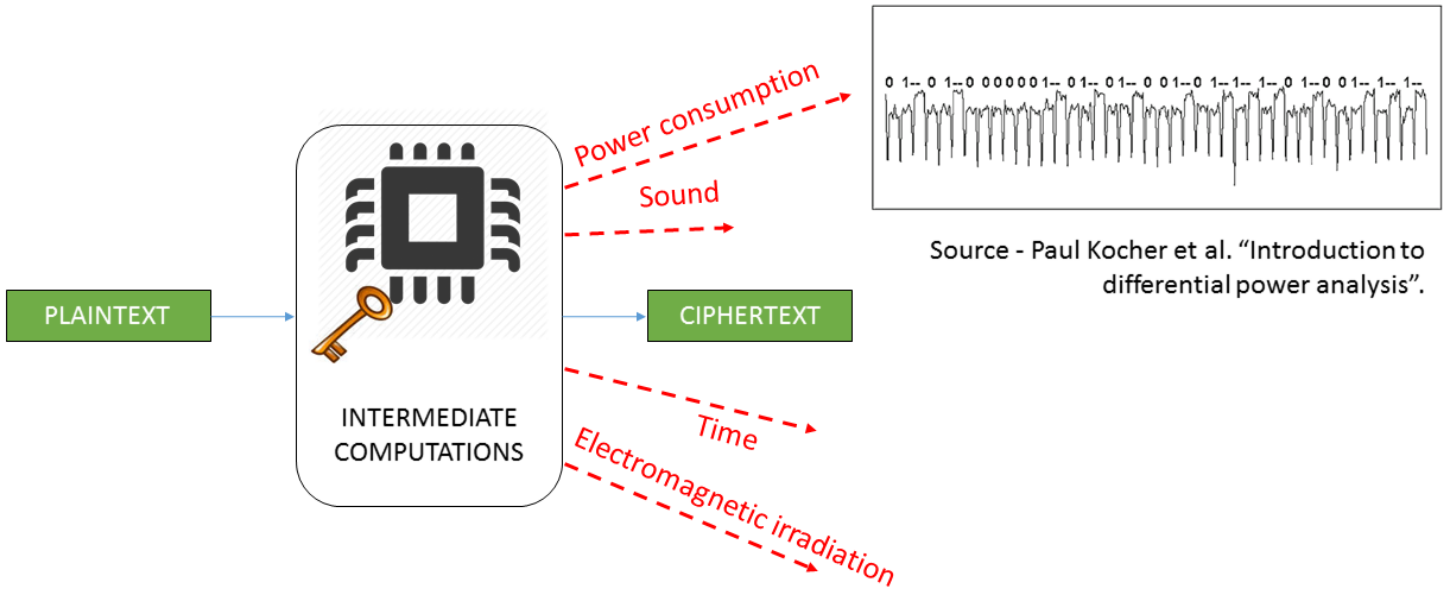
Side-Channel Vulnerability of Embedded Cryptography



Source - Paul Kocher et al. "Introduction to differential power analysis".

Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

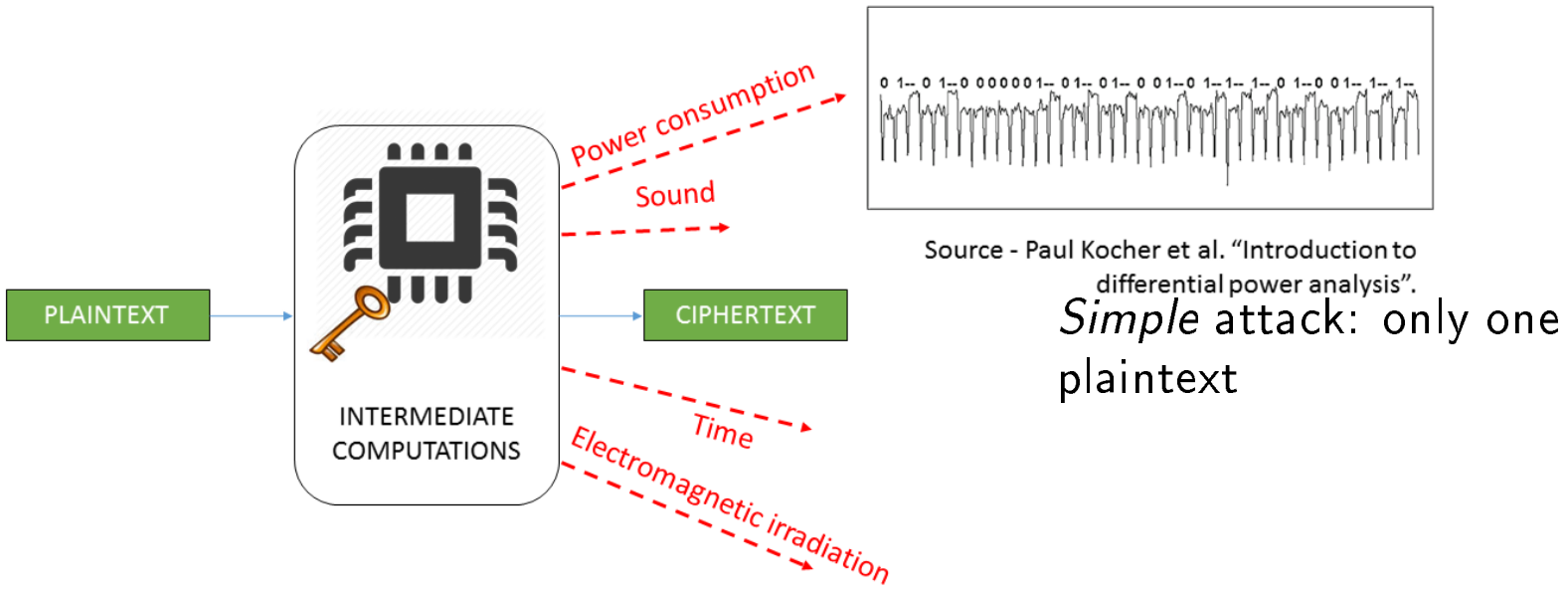
Side-Channel Vulnerability of Embedded Cryptography



Source - Paul Kocher et al. "Introduction to differential power analysis".

Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

Side-Channel Vulnerability of Embedded Cryptography

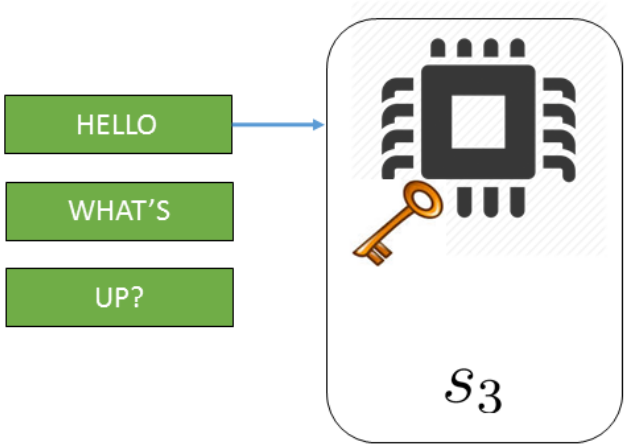


Source - Paul Kocher et al. "Introduction to differential power analysis".

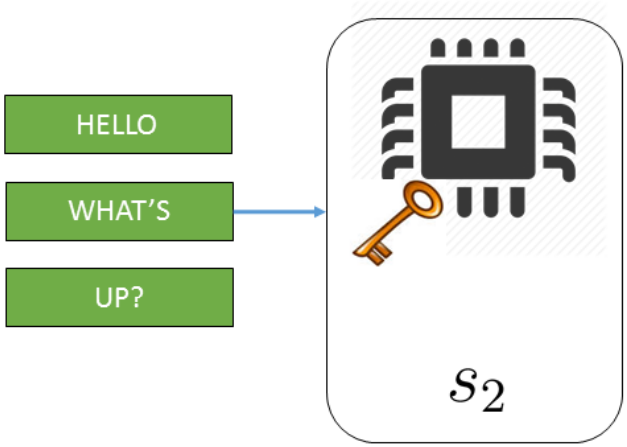
Simple attack: only one plaintext

Classical Cryptanalysis	Side-Channel Cryptanalysis
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

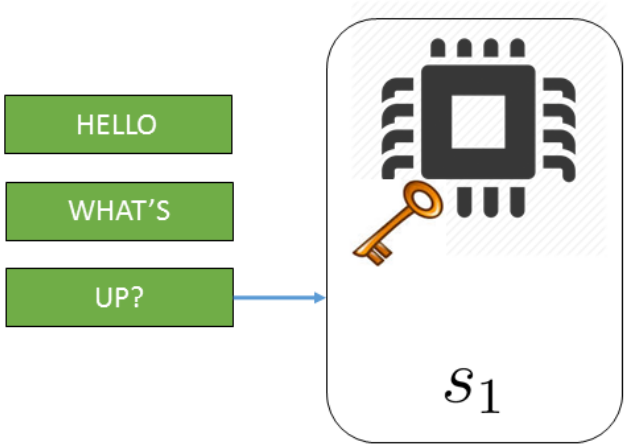
Advanced Side-Channel Attacks



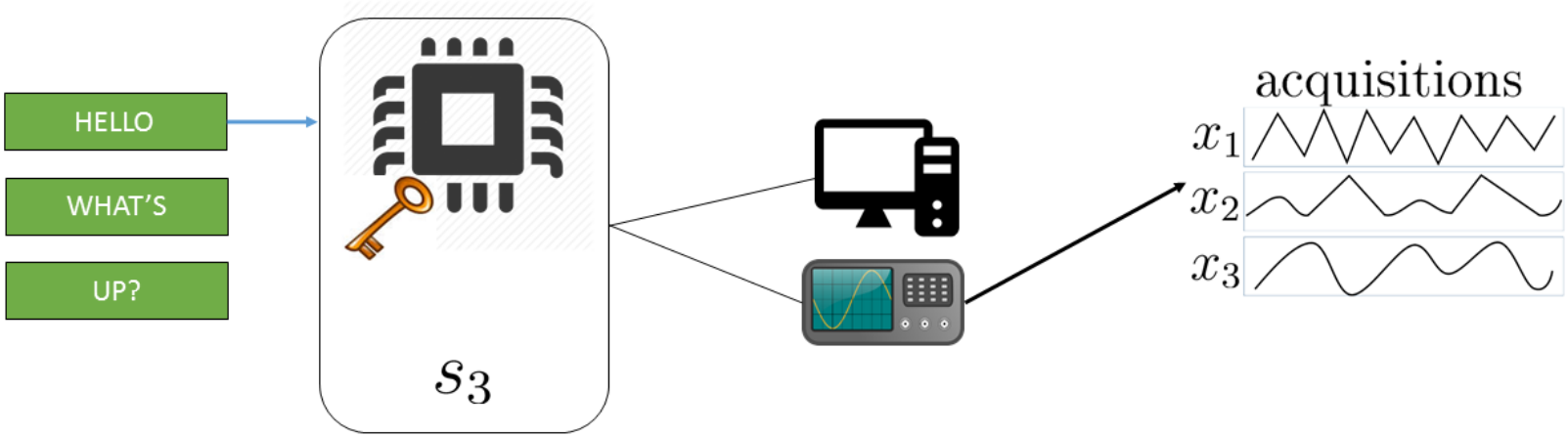
Advanced Side-Channel Attacks



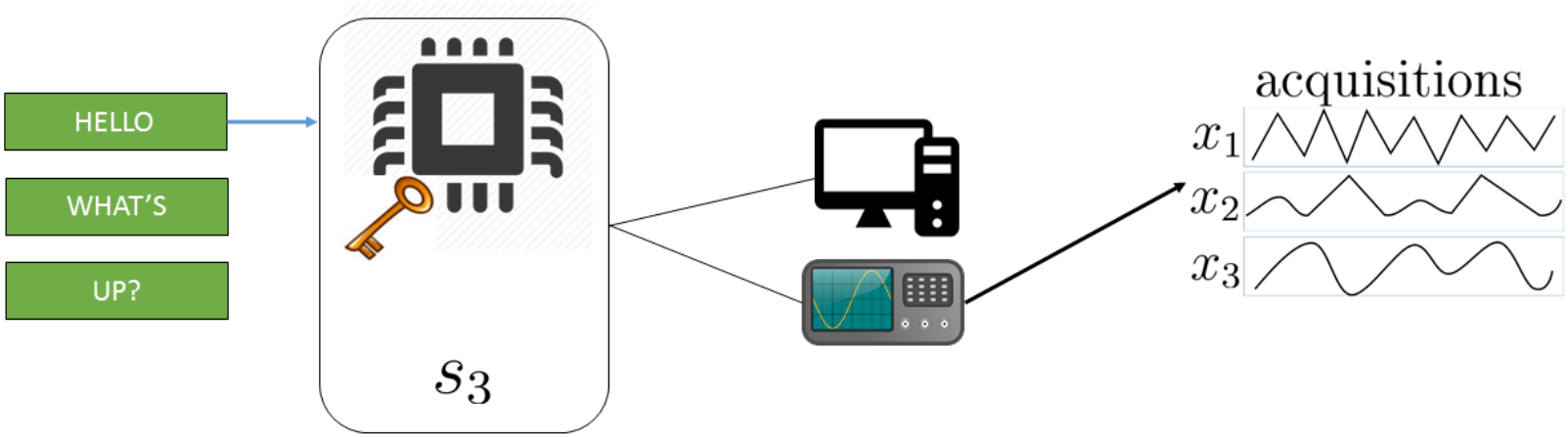
Advanced Side-Channel Attacks

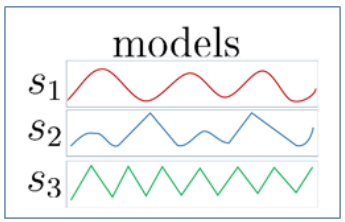


Advanced Side-Channel Attacks

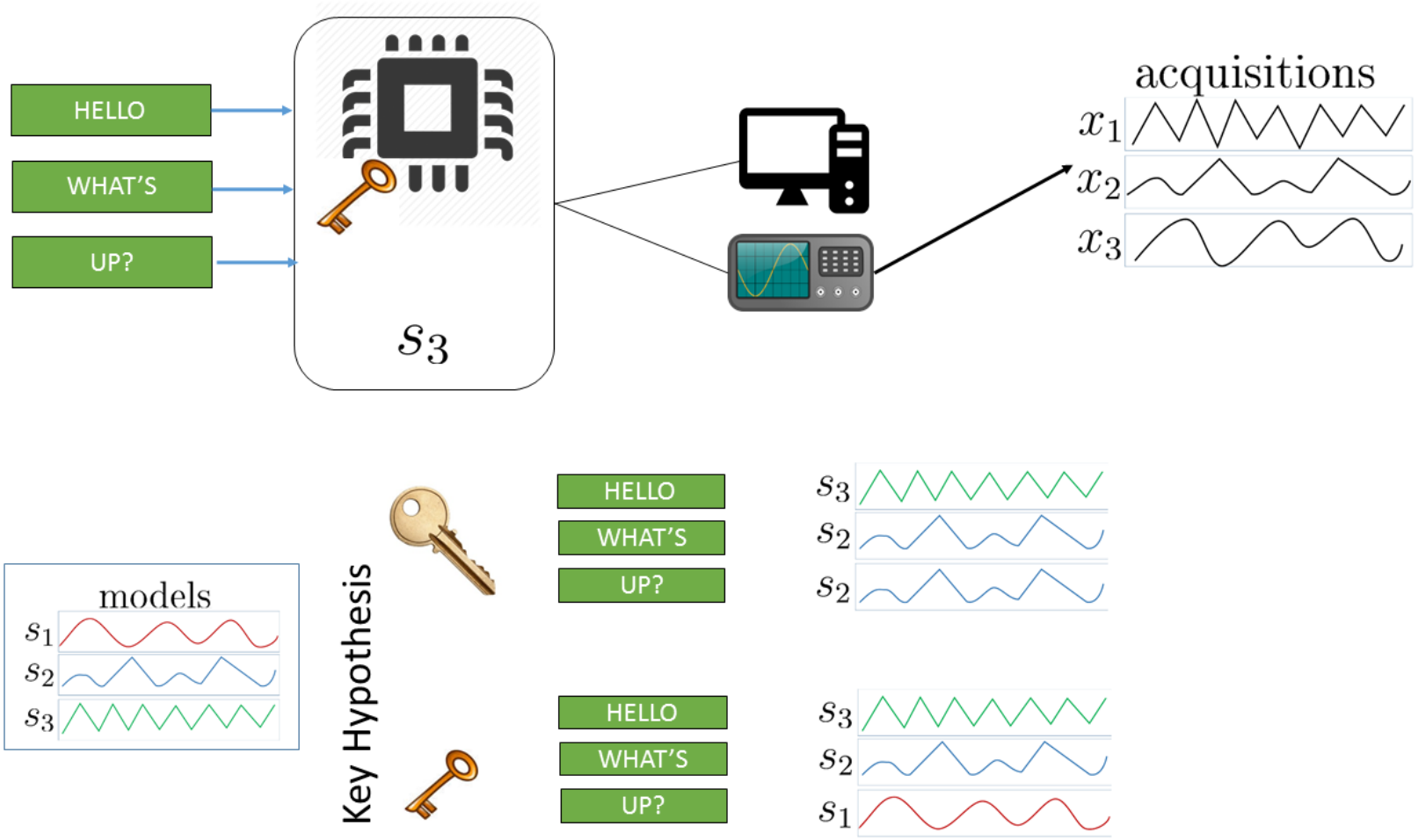


Advanced Side-Channel Attacks

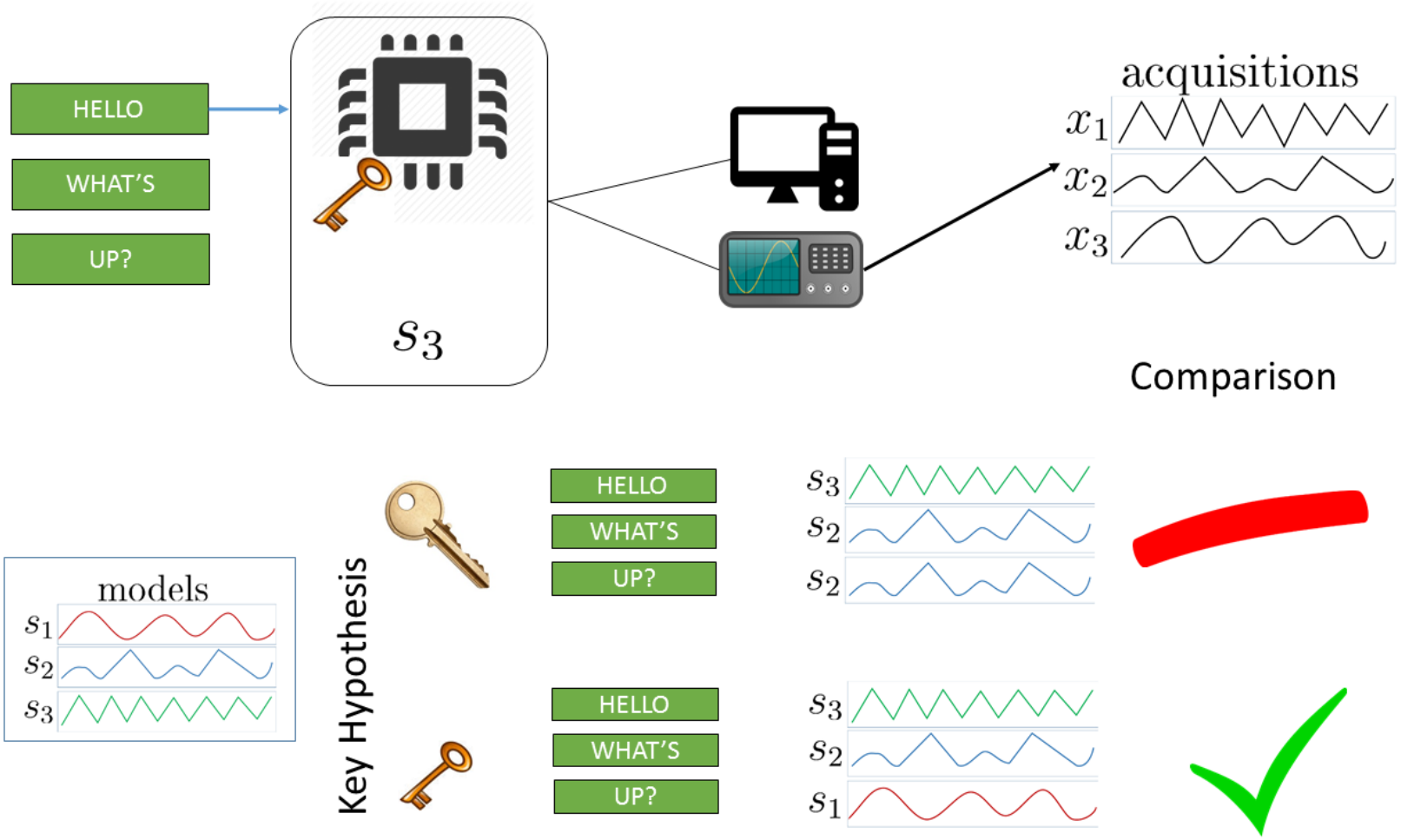




Advanced Side-Channel Attacks

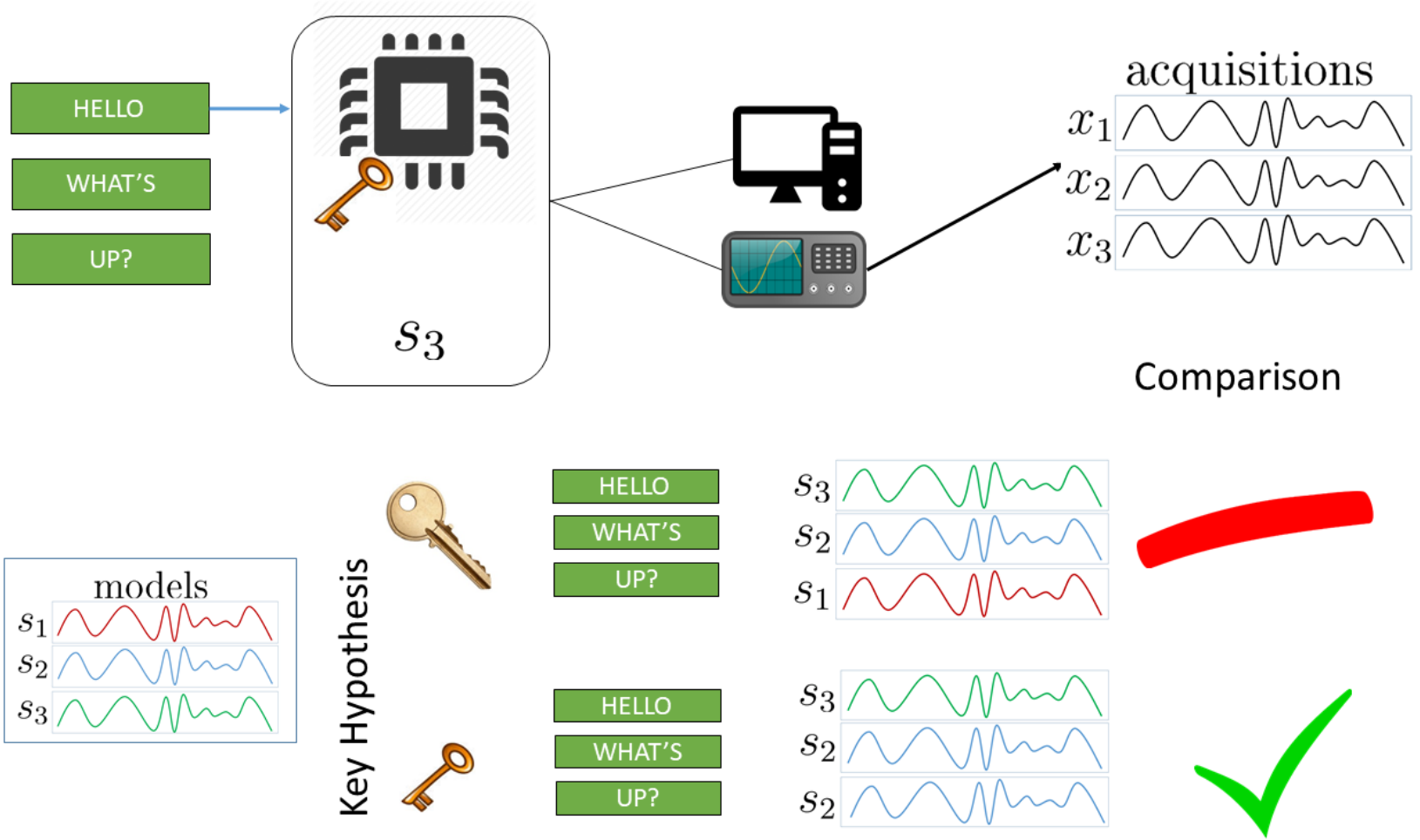


Advanced Side-Channel Attacks

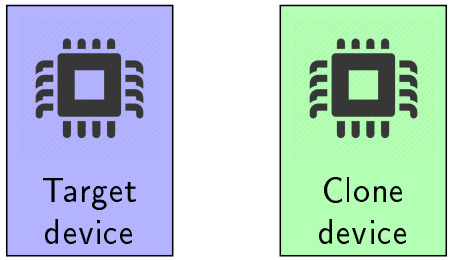


Advanced Side-Channel Attacks

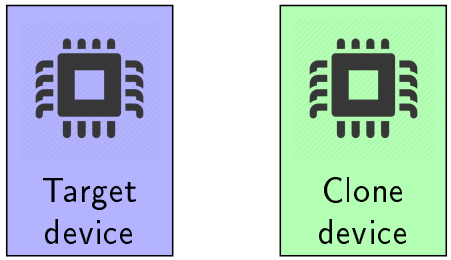
Don't panic!



Profiling Attacks...Supervised Learning



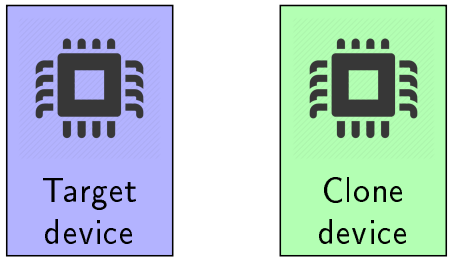
Profiling Attacks...Supervised Learning



Machine Learning

Supervised Learning

Profiling Attacks...Supervised Learning

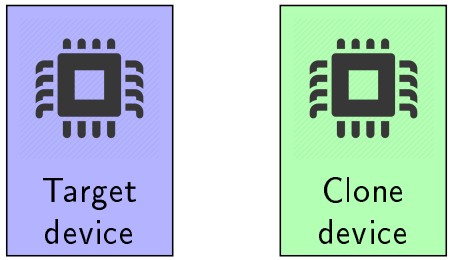


Machine Learning

"A computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance on T, as measured by P, improves with experience E. "[TM97]

Supervised Learning

Profiling Attacks...Supervised Learning



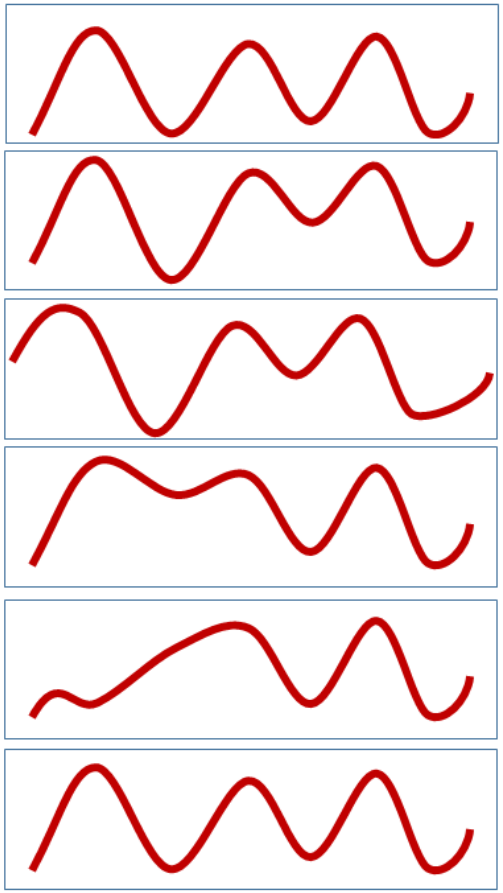
Machine Learning

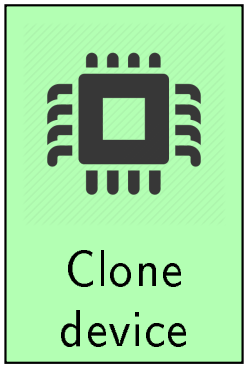
"A computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance on T, as measured by P, improves with experience E. "[TM97]

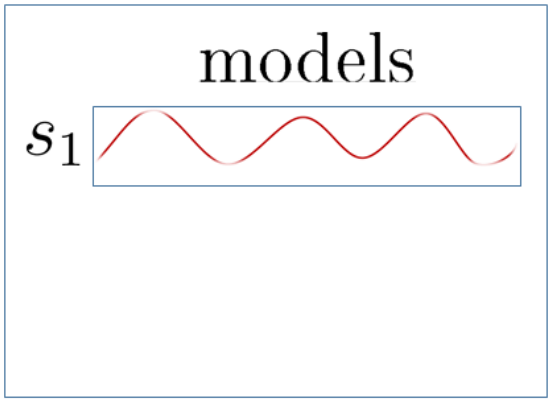
Supervised Learning

The *supervised* learning algorithms access to a dataset of examples, each associated in general to a *target* or *label*.

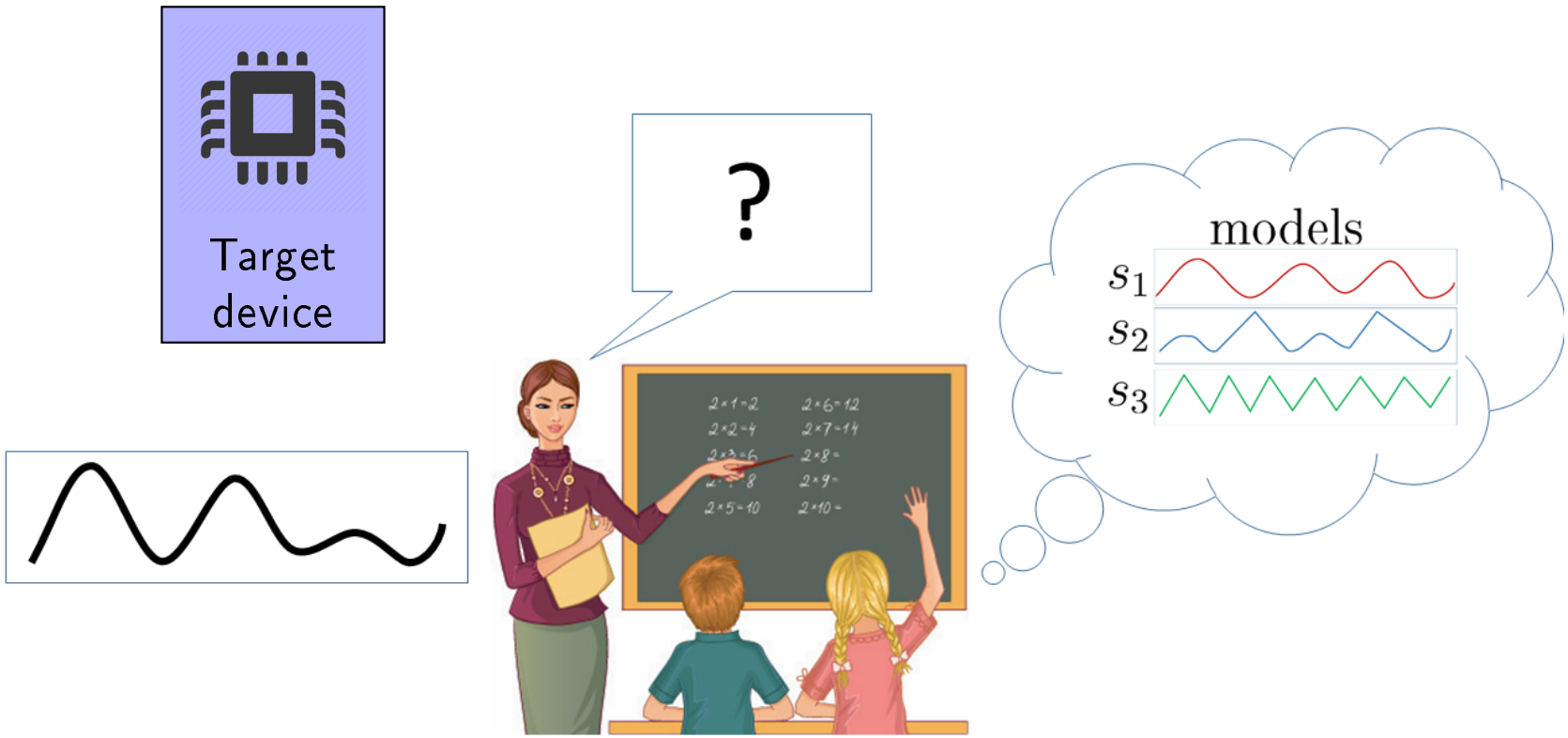
Classroom Side-Channel Attacks







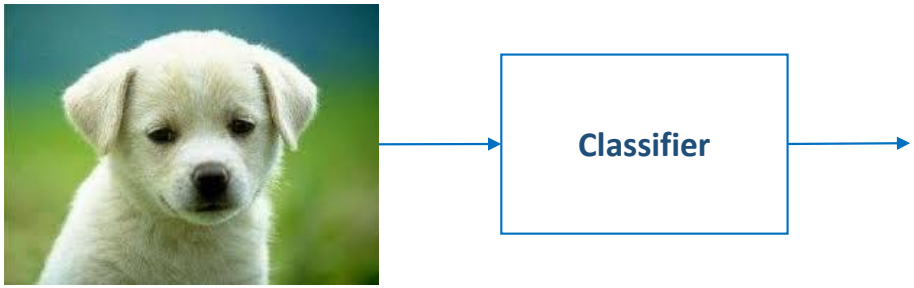
Classroom Side-Channel Attacks



Classification

Classification problem

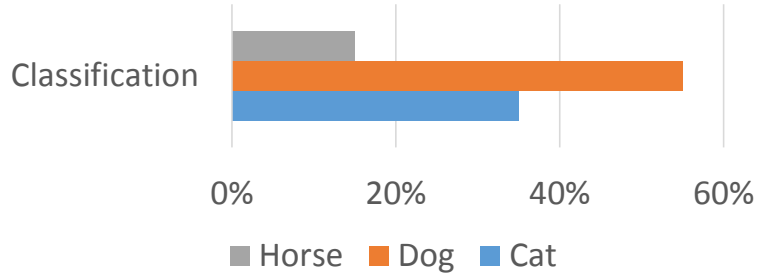
Assign to a datum $\vec{X}$ (e.g. an image) a label Z among a set of possible labels $\mathcal{Z} = \{\mathrm{Cat}, \mathrm{Dog}, \mathrm{Horse}\}$



Classification

Classification problem

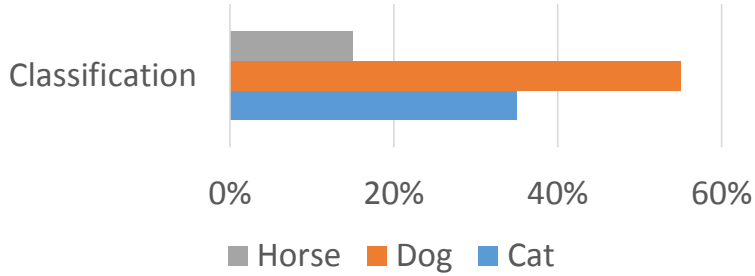
Assign to a datum $\vec{X}$ (e.g. an image) a label Z among a set of possible labels $\mathcal{Z} = \{\text{Cat}, \text{Dog}, \text{Horse}\}$



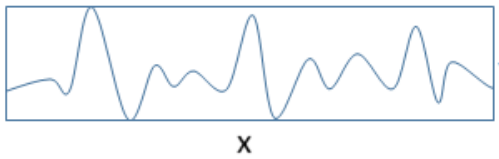
Classification

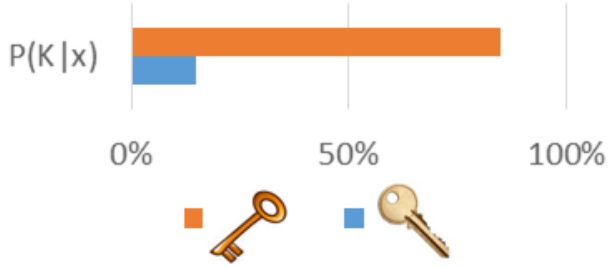
Classification problem

Assign to a datum $\vec{X}$ (e.g. an image) a label Z among a set of possible labels $\mathcal{Z} = \{\text{Cat}, \text{Dog}, \text{Horse}\}$



Simple Attack as a Classification Problem

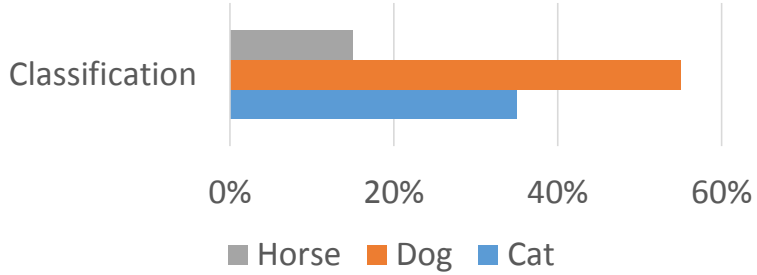




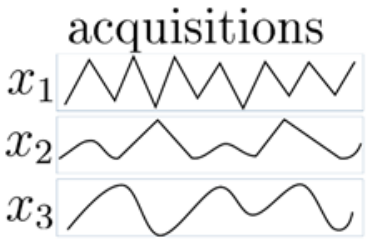
Classification

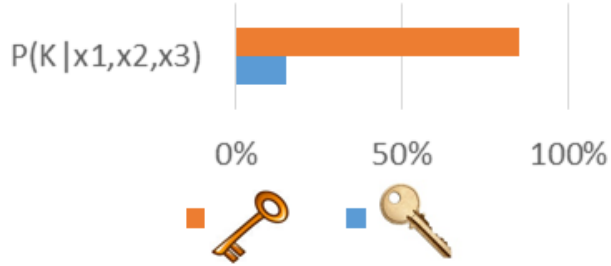
Classification problem

Assign to a datum $\vec{X}$ (e.g. an image) a label Z among a set of possible labels $\mathcal{Z} = \{\text{Cat}, \text{Dog}, \text{Horse}\}$



Advanced Attack as a Classification Problem

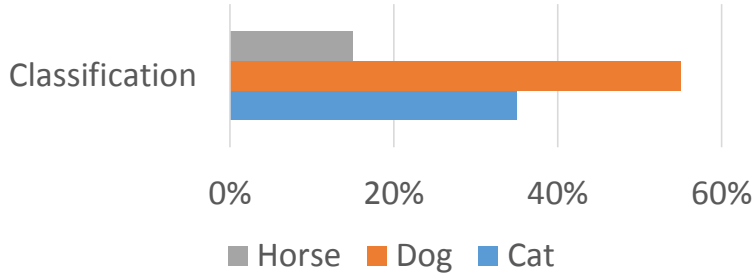




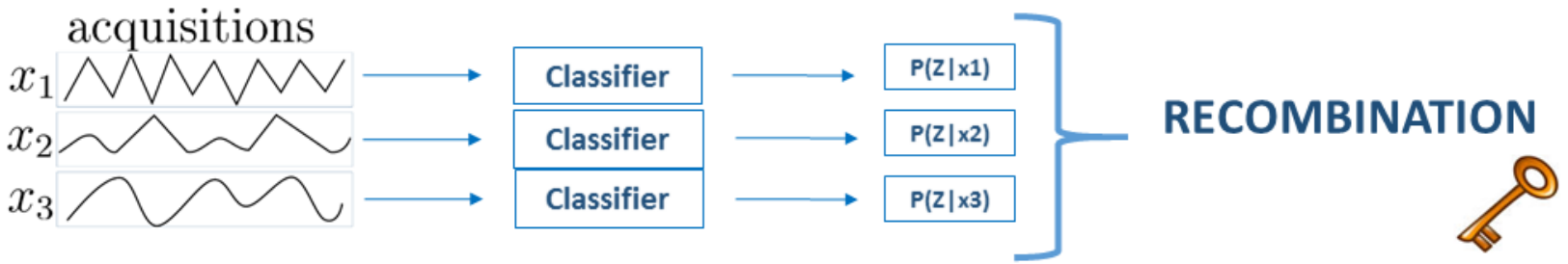
Classification

Classification problem

Assign to a datum $\vec{X}$ (e.g. an image) a label Z among a set of possible labels $\mathcal{Z} = \{\text{Cat}, \text{Dog}, \text{Horse}\}$



Advanced Attack as Multiple Classification Problems



Contents

Notations

Notations and generalities

- Side-channel traces: realizations of a random vector $\vec{X} \in \mathbb{R}^D$
- D is the number of time samples (or features)
- Target: a *sensitive* variable $Z = f(\text{plaintext}, \text{key})$ in $\mathcal{Z} = \{s_1, \ldots, s_{|\mathcal{Z}|}\}$
- each label s_i identifies a *class*

Profiling attack scenario

- labelled traces $\mathcal{D}_{\text{train}} = (\vec{x}_i, z_i)_{i=1}^{N}$, acquired under known Z, to characterise the signals
- attack traces $\mathcal{D}_{\text{attack}} = (\vec{x}_i, p_i)_{i=1}^{N_a}$ acquired under known plaintext

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\vec{X} \mid Z=f(p_i,k)}(\vec{x}_i)$$

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$, $p_{\vec{X}}$, p_Z

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\vec{X} \mid Z=f(p_i,k)}(\vec{x}_i)$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \vec{X}=\vec{x}}(z) = \frac{p_{\vec{X} \mid Z=z}(\vec{x}) p_Z(z)}{p_{\vec{X}}(\vec{x})} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \vec{X}=\vec{x}_i}(f(k, e_i)) \ ,$$

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$, $p_{\vec{X}}$, p_Z (generative model)
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\vec{X} \mid Z=f(p_i,k)}(\vec{x}_i)$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \vec{X}=\vec{x}}(z) = \frac{p_{\vec{X} \mid Z=z}(\vec{x}) p_Z(z)}{p_{\vec{X}}(\vec{x})} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \vec{X}=\vec{x}_i}(f(k, e_i)) \ ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$, $p_{\vec{X}}$, p_Z (generative model)
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\vec{X} \mid Z=f(p_i,k)}(\vec{x}_i)$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \vec{X}=\vec{x}}(z) = \frac{p_{\vec{X} \mid Z=z}(\vec{x}) p_Z(z)}{p_{\vec{X}}(\vec{x})} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \vec{X}=\vec{x}_i}(f(k, e_i)) \ ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$, $p_{\vec{X}}$, p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\vec{X} \mid Z=f(p_i,k)}(\vec{x}_i)$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \vec{X}=\vec{x}}(z) = \frac{p_{\vec{X} \mid Z=z}(\vec{x}) p_Z(z)}{p_{\vec{X}}(\vec{x})} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \vec{X}=\vec{x}_i}(f(k, e_i)) \ ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon : \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $p_{Z \mid \epsilon(\vec{X})=\epsilon(\vec{x})}$ (discriminative model)

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\epsilon(\vec{X}) \mid Z=f(p_i,k)}(\epsilon(\vec{x}_i))$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \epsilon(\vec{X})=\epsilon(\vec{x})}(z) = \frac{p_{\epsilon(\vec{X}) \mid Z=z}(\epsilon(\vec{x}))p_Z(z)}{p_{\epsilon(\vec{X})}(\epsilon(\vec{x}))} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \epsilon(\vec{X})=\epsilon(\vec{x}_i)}(f(k, e_i)) \ ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho \colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon \colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X})) \mid Z=z}$, $p_{\epsilon(\rho(\vec{X}))}$, p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $p_{Z \mid \rho(\epsilon(\vec{X}))=\epsilon(\rho(\vec{x}))}$ (discriminative model)

Attack phase

- Log-likelihood score for each key hypothesis k

$$d_k = \prod_{i=1}^{N_a} p_{\epsilon(\rho(\vec{X})) \mid Z=f(p_i,k)}(\epsilon(\rho(\vec{x}_i)))$$

- A-posteriori probability score for each key hypothesis k

$$p_{Z \mid \rho(\epsilon(\vec{X}))=\epsilon(\rho(\vec{x}))}(z) = \frac{p_{\epsilon(\rho(\vec{X})) \mid Z=z}(\epsilon(\rho(\vec{x})))p_Z(z)}{p_{\epsilon(\rho(\vec{X}))}(\epsilon(\rho(\vec{x})))} \text{ Bayes' theorem}$$

$$d_k = \prod_{i=1}^{N_a} p_{Z \mid \epsilon(\rho(\vec{X}))=\epsilon(\rho(\vec{x}_i))}(f(k, e_i)) \ ,$$

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho : \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon : \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho : \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon : \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

- Ameliorate the template attack routine by proposing efficient dimensionality reduction techniques

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho : \mathbb{R}^D \rightarrow \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon : \mathbb{R}^D \rightarrow \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

- Ameliorate the template attack routine by proposing efficient dimensionality reduction techniques
- Consider the presence of most-commonly-implemented SCA countermeasures (masking, hiding)

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho : \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon : \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

- Ameliorate the template attack routine by proposing efficient dimensionality reduction techniques
- More generally, ameliorate the profiling attack strategy
- Consider the presence of most-commonly-implemented SCA countermeasures (masking, hiding)

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho \colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon \colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
  - $Z \mid \vec{X} = \vec{x}$ (discriminative model)

DEEP LEARNING

Objectives

- Ameliorate the template attack routine by proposing efficient dimensionality reduction techniques
- More generally, ameliorate the profiling attack strategy
- Consider the presence of most-commonly-implemented SCA countermeasures (masking, hiding)

Dimensionality Reduction: State of the Art

Dimensionality Reduction

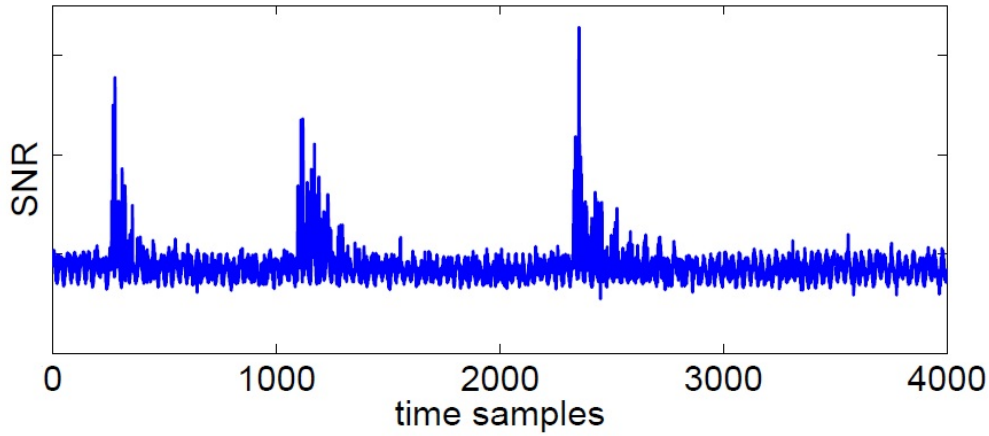
$$\epsilon : \mathbb{R}^D \to \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]



Dimensionality Reduction: State of the Art

Dimensionality Reduction

$$\epsilon : \mathbb{R}^D \to \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

Linear feature extraction

$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

Contributions

- **Linear Dimensionality Reduction**([CARDIS 2015]):
  - PCA, choise of components ELV
  - LDA in case of undersampling
- **Kernel Discriminant Analysis**([CARDIS 2016]): application of an appropriate kernel trick to LDA, in order to manage masking countermeasure
- **Convolutional Neural Networks**([CHES 2017]) :
  - discriminative model by means of neural network classifiers
  - convolutional layers to manage desyncrhonisation (a form of hiding)
  - Data Augmentation techniques to reduce overfitting

Contributions

- **Linear Dimensionality Reduction**([CARDIS 2015]):
  - PCA, choise of components ELV
  - LDA in case of undersampling
- **Kernel Discriminant Analysis**([CARDIS 2016]): application of an appropriate kernel trick to LDA, in order to manage masking countermeasure
- **Convolutional Neural Networks**([CHES 2017]) :
  - discriminative model by means of neural network classifiers
  - convolutional layers to manage desyncrhonisation (a form of hiding)
  - Data Augmentation techniques to reduce overfitting

Contents

Model-based SNR and Fisher's Criterion

Signal-to-Noise Ratio (SNR)

- Independent Noise Assumption: $\vec{X} = \varphi(Z) + \vec{B}$
- $\vec{\mu}_s = \hat{\mathbb{E}}[\vec{X} \mid Z = s] = \frac{1}{N_s} \sum_{i:\ z_i = s} \vec{x}_i$ sample mean per class ($\approx \varphi(Z)$)
- $\vec{\varrho}_s = \hat{\mathrm{Var}}(\vec{X} \mid Z = s) = \frac{1}{N_s - 1} \sum_{i:\ z_i = s} (\vec{x}_i - \vec{\mu}_s)^2$ sample variance per class ($\approx \mathrm{var}(\vec{B})$)
-

$$\mathrm{SNR}(t) = \frac{\hat{\mathrm{Var}}(\vec{\mu}_Z(t))}{\hat{\mathbb{E}}[\vec{\varrho}_Z(t)]} \qquad \frac{\textit{variance inter-class}}{\textit{variance intra-class}}$$

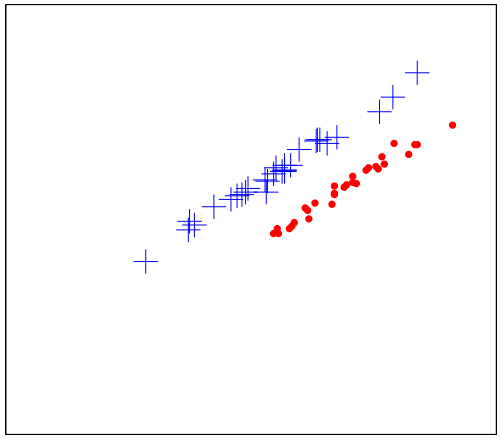
Fisher's Criterion for Linear Dimensionality Reduction

- $\mathbf{S_B} = \sum_{s \in \mathcal{Z}} N_s (\vec{\mu}_s - \overline{\vec{x}})(\vec{\mu}_s - \overline{\vec{x}})^\intercal$ (inter-class scatter matrix)
- $\mathbf{S_W} = \sum_{s \in \mathcal{Z}} \sum_{i=1:\ z_i = s} (\vec{x}_i - \vec{\mu}_s)(\vec{x}_i - \vec{\mu}_s)^\intercal$ (intra-class scatter matrix)
- Fisher's criterion

$$\hat{\vec{\alpha}} = \mathrm{argmax}_{\vec{\alpha}} \frac{\vec{\alpha}^\intercal \mathbf{S_B} \vec{\alpha}}{\vec{\alpha}^\intercal \mathbf{S_W} \vec{\alpha}}\ , \tag{1}$$

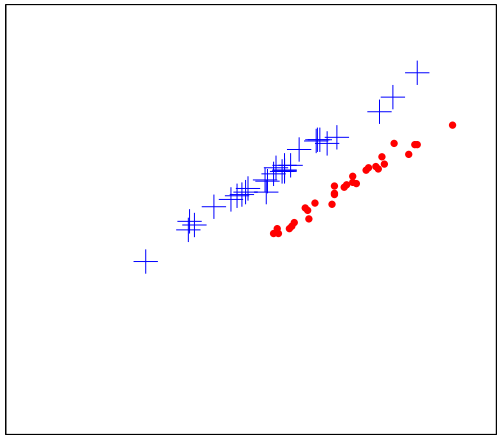
$\epsilon(\vec{x}) = A\vec{x}$ where $A = [\vec{\alpha}_1, \ldots, \vec{\alpha}_{|\mathcal{Z}|-1}]$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$

LDA: an optimal binary linear classifier



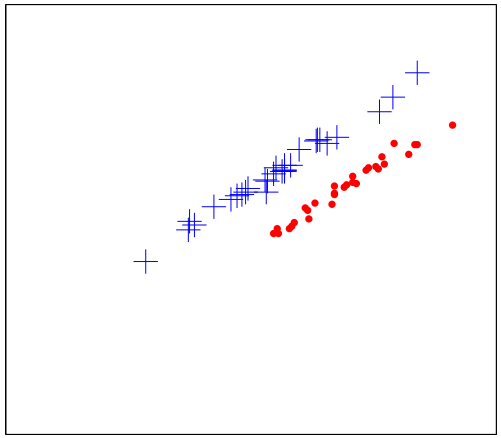
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$

LDA: an optimal binary linear classifier



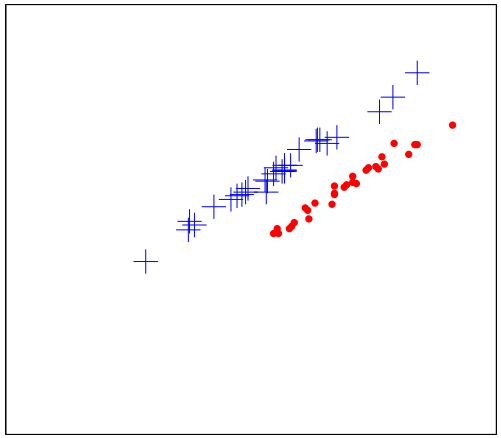
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$

LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

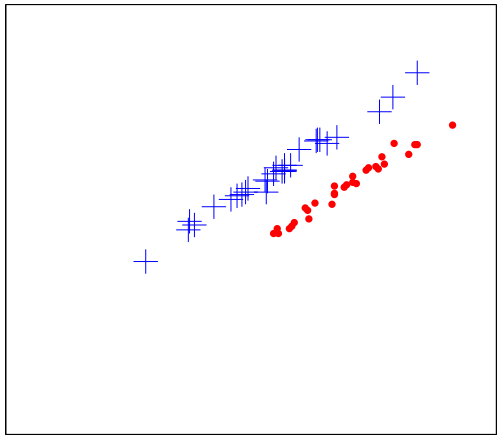
LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j

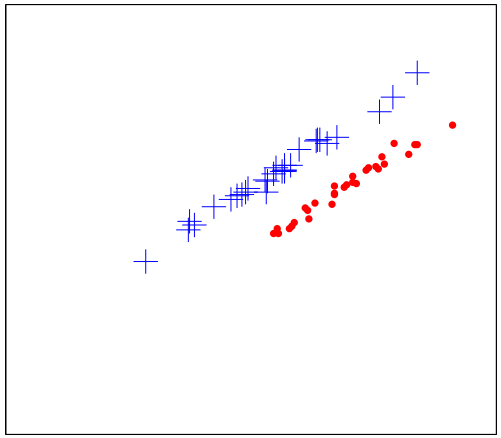
LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j
- $\Rightarrow a = \vec{w}^\intercal \vec{x} + w_0$ (linear decision boundary, $\vec{w}$ and w_0 functions of Σ, μ_j)

LDA: an optimal binary linear classifier



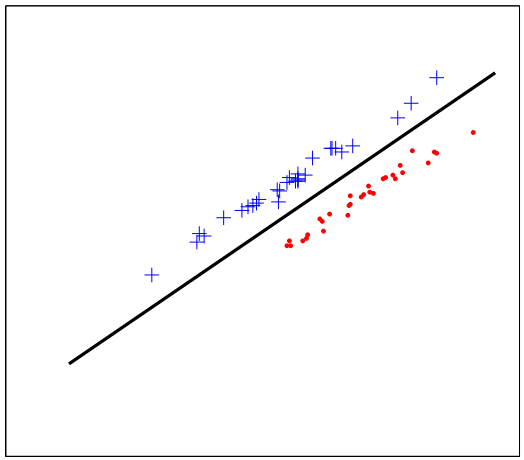
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j
- $\Rightarrow a = \vec{w}^\intercal \vec{x} + w_0$ (linear decision boundary, $\vec{w}$ and w_0 functions of Σ, μ_j)

Generalised linear discriminative model

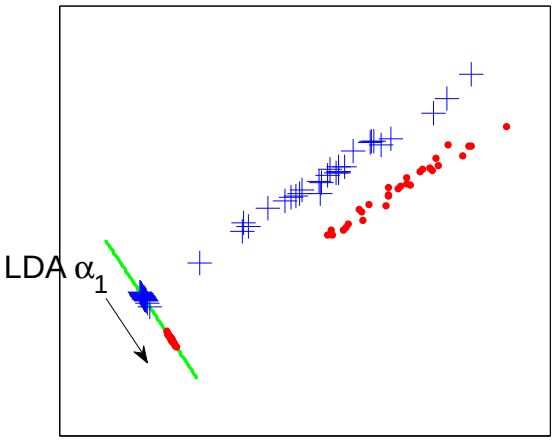
$$\Pr(s_1 \mid \vec{x}) = \sigma(\vec{w}^\intercal \vec{x} + w_0) \text{ ,where } \sigma(a) = \frac{1}{1+e^{-a}} \textit{ logistic sigmoid} \qquad (2)$$

LDA and Fisher's Discriminant



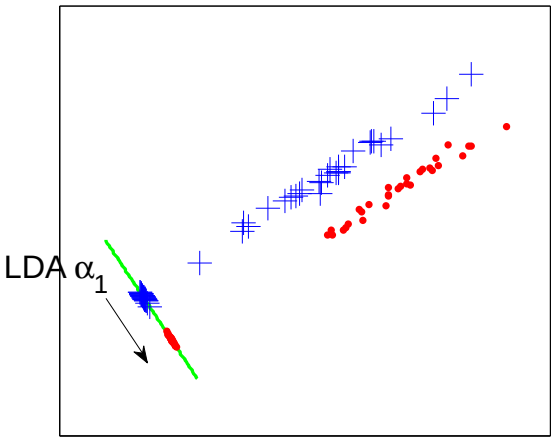
- LDA: linear decision boundary
  $a = \vec{w}^{\intercal}\vec{x} + w_0$

LDA and Fisher's Discriminant



- LDA: linear decision boundary $a = \vec{w}^\intercal \vec{x} + w_0$
- Equivalently, project data onto $\vec{w}^\intercal \vec{x}$ (orthogonally to the decision boudary), than classify by a real threshold (optimally w_0).

LDA and Fisher's Discriminant



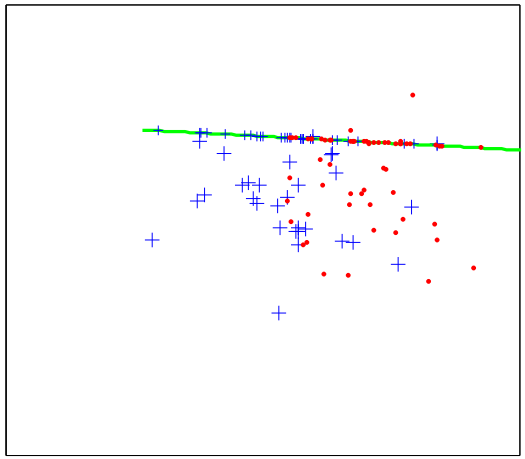
- LDA: linear decision boundary $a = \vec{w}^\intercal \vec{x} + w_0$
- Equivalently, project data onto $\vec{w}^\intercal \vec{x}$ (orthogonally to the decision boudary), than classify by a real threshold (optimally w_0).

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j

Fact, abuse and preference for the dimensionality reduction formulation

- When LDA assumptions are met, the solution $\vec{\alpha}_1$ of the Fisher's criterion is orthogonal to $\vec{w}$.
- assumption not required
- naturally multi-class

Linear separability



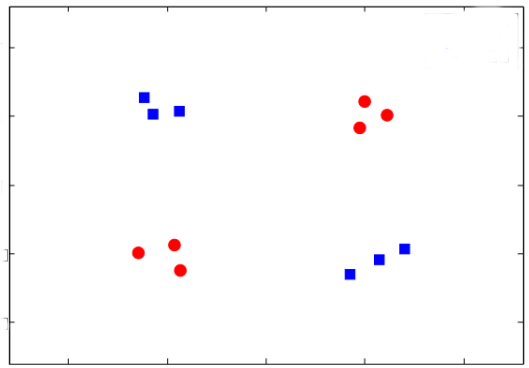
LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?

Linear separability

LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?

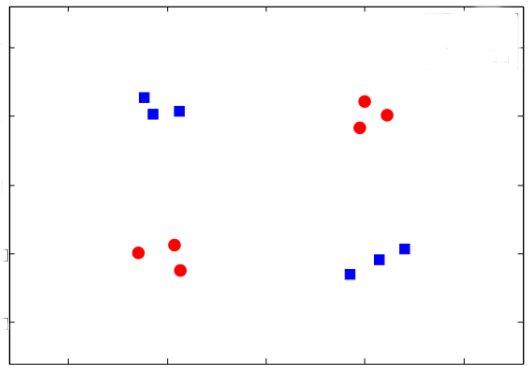


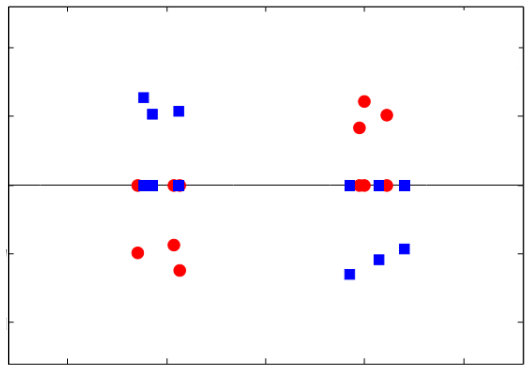
$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant

Linear separability

LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?





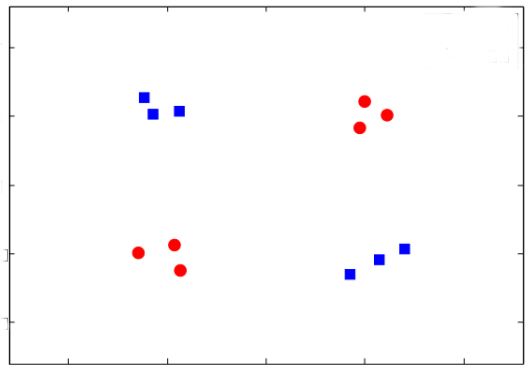
$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant

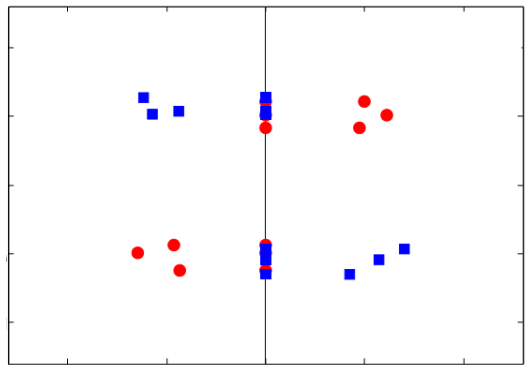
$\mathbb{E}[A\vec{X}|Z = z]$ is constant as well $\Rightarrow$ Projecting extractors can't help!

Linear separability

LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?





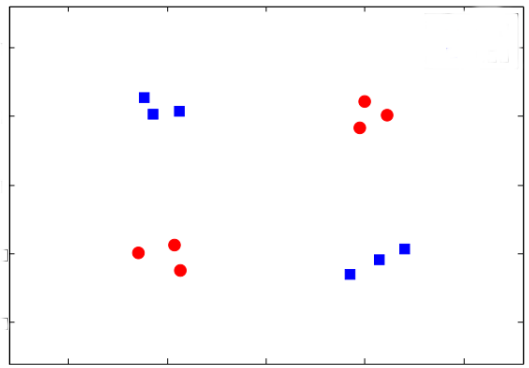
$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant

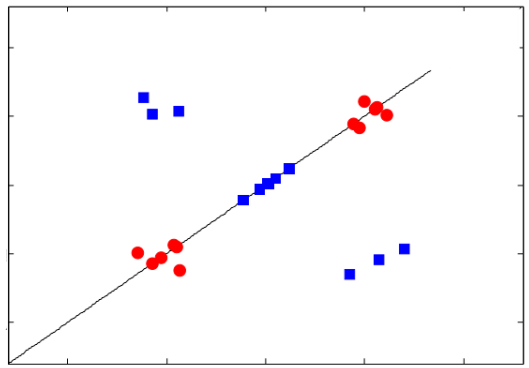
$\mathbb{E}[A\vec{X}|Z = z]$ is constant as well $\Rightarrow$ Projecting extractors can't help!

Linear separability

LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?





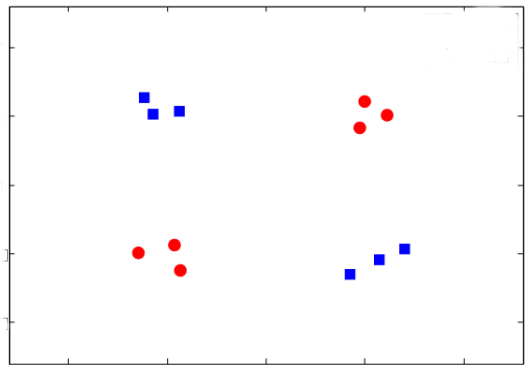
$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant

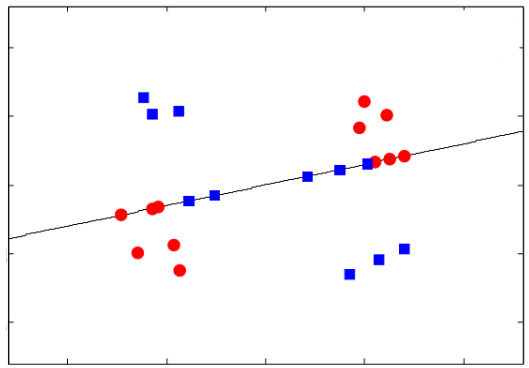
$\mathbb{E}[A\vec{X}|Z = z]$ is constant as well $\Rightarrow$ Projecting extractors can't help!

Linear separability

LDA: linear decision boundary $a = \vec{w}^{\intercal}\vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?





$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant

$\mathbb{E}[A\vec{X}|Z = z]$ is constant as well $\Rightarrow$ Projecting extractors can't help!

Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares $Z = M_1 \star \cdots \star M_d$
with $M_1, \ldots, M_{d-1}$ random shares (or masks)
and $M_d = Z \star M_1^{-1} \star \cdots \star M_{d-1}^{-1}$
Software implementations: shares are handled at different time samples

$$t_1, \ldots, t_d$$

$\Rightarrow$ each time sample is independent from Z.

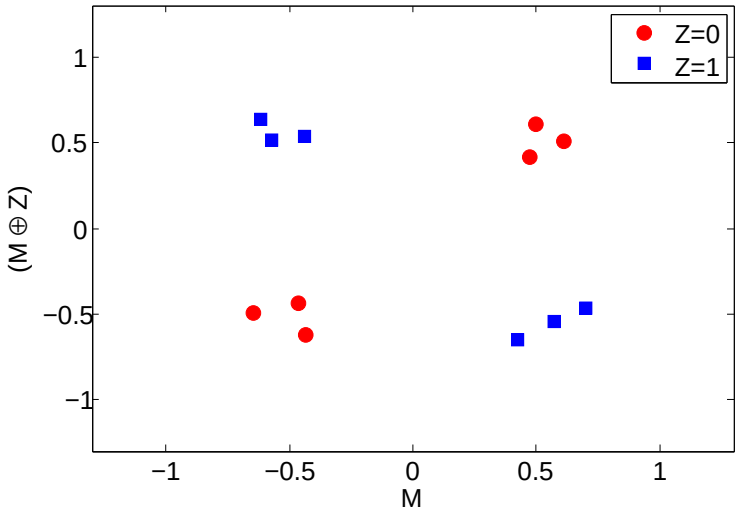
Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares $Z = M_1 \star \cdots \star M_d$
with $M_1, \ldots, M_{d-1}$ random shares (or masks)
and $M_d = Z \star M_1^{-1} \star \cdots \star M_{d-1}^{-1}$
Software implementations: shares are handled at different time samples

$$t_1, \ldots, t_d$$

$\Rightarrow$ each time sample is independent from Z.



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

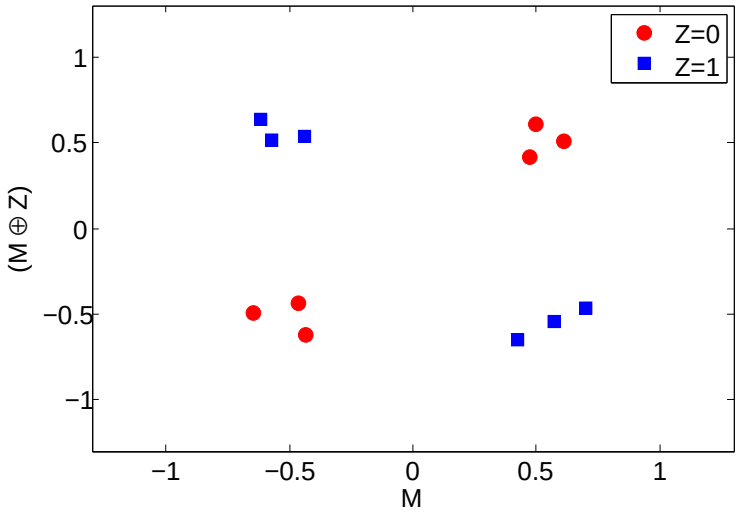
Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares $Z = M_1 \star \cdots \star M_d$
with $M_1, \ldots, M_{d-1}$ random shares (or masks) UNKNOWN
and $M_d = Z \star M_1^{-1} \star \cdots \star M_{d-1}^{-1}$
Software implementations: shares are handled at different time samples

$$t_1, \ldots, t_d \qquad \text{UNKNOWN}$$

$\Rightarrow$ each time sample is independent from Z.



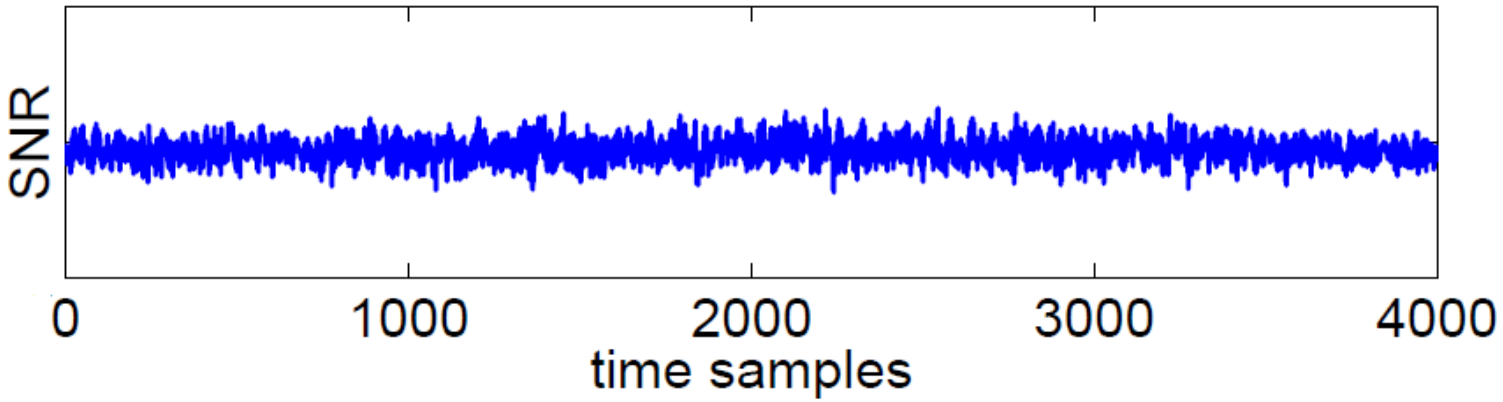
Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

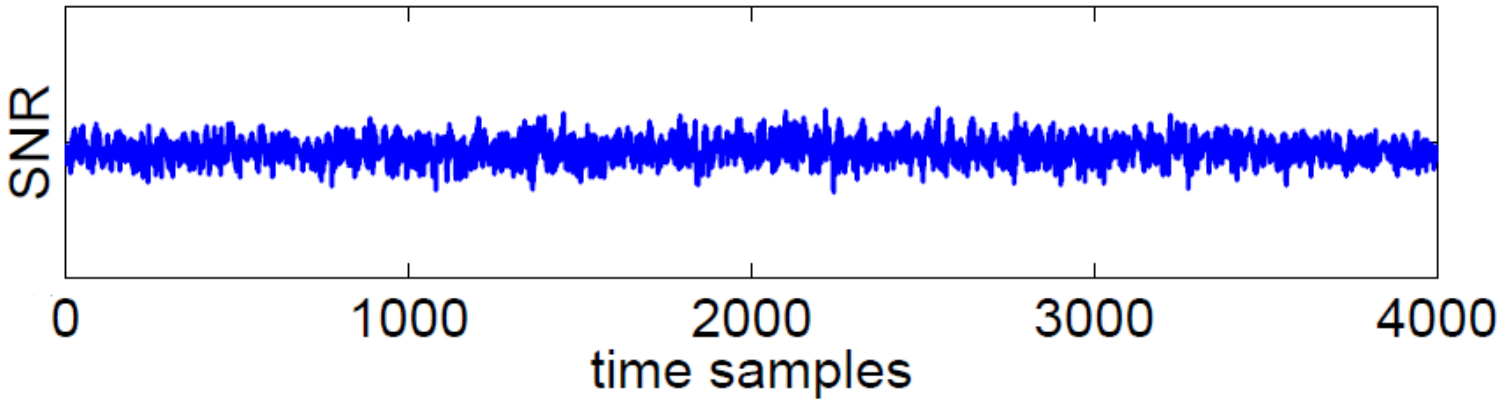
In high-dimensional traces

$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant $\Rightarrow$ SNR, SOD, t-statistic are null!



In high-dimensional traces

$f(z) = \mathbb{E}[\vec{X}|Z = z]$ is constant $\Rightarrow$ SNR, SOD, t-statistic are null!



$\mathbb{E}[A\vec{X}|Z = z]$ is constant as well

Higher-Order Side-Channel Attacks

Exploit $f(z) = \mathbb{E}\left[\vec{X}[t_1]\vec{X}[t_2]\ldots\vec{X}[t_d]|Z = z\right]$ non-constant.

Necessary condition

[Car+14] The statistic extracted from measurements must contain

$$\vec{X}[t_1]\vec{X}[t_2]\ldots\vec{X}[t_d]$$

PoIs Research

How to detect the d-tuple $t_1, \ldots, t_d$?

A lacking literature

- many HO attacks papers assume the knowledge of $t_1, \ldots, t_d$
- PoI research exploiting the random shares knowledge (back to unprotected case using $M_1, \ldots, M_d$ instead of Z)
- naive strategy: infer over all possible d-tuples
- Hand selection via educated guess [Osw+06]

Generalizing extractors for higher-order context

- Selecting extractors $\longrightarrow$ Projection Pursuits [Dur+15]
- Projecting extractors $\longrightarrow$ Kernel Discriminant Analysis [CARDIS '16]

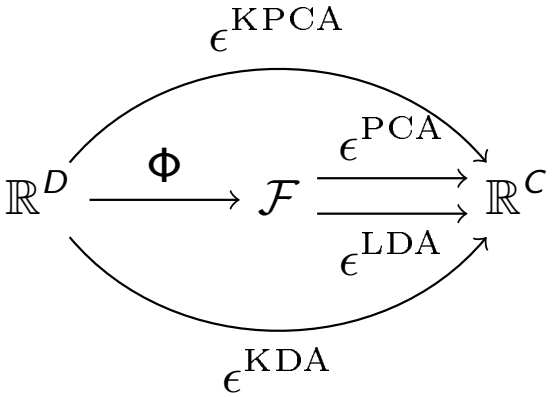
KDA: the purpose

Problem

Useful statistics lie in a high-dimensional *feature* space:

$$\mathcal{F} = \mathbb{R}^{\binom{D+d-1}{d}}$$

(all dth-degree monomials in the trace coordinates)



Φ non-linear function

What KDA provides

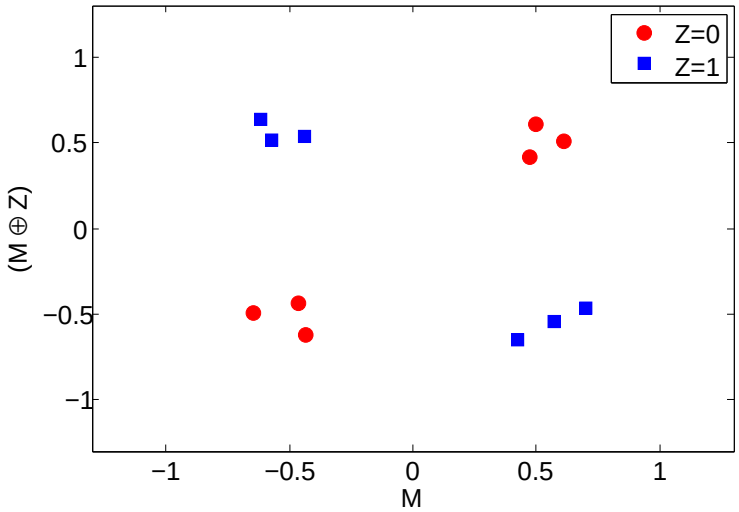
KDA allows performing LDA in $\mathcal{F}$, remaining in $\mathbb{R}^D$.

KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

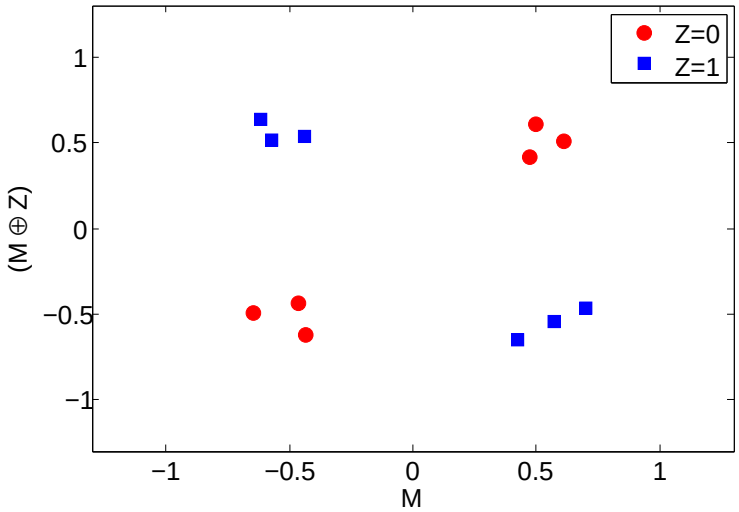
t_2: $M \oplus Z + n$ (Boolean masking)

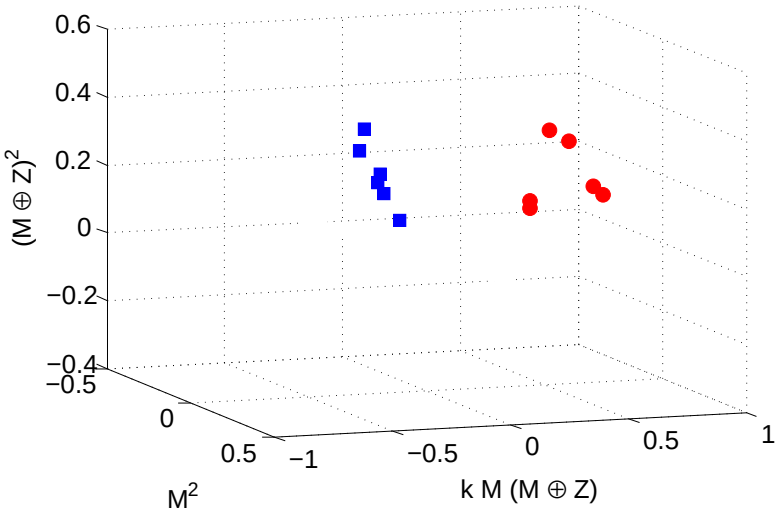


KDA: an intuition

Toy example: 2 time samples, 1-bit data

- t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$
- t_2: $M \oplus Z + n$ (Boolean masking)



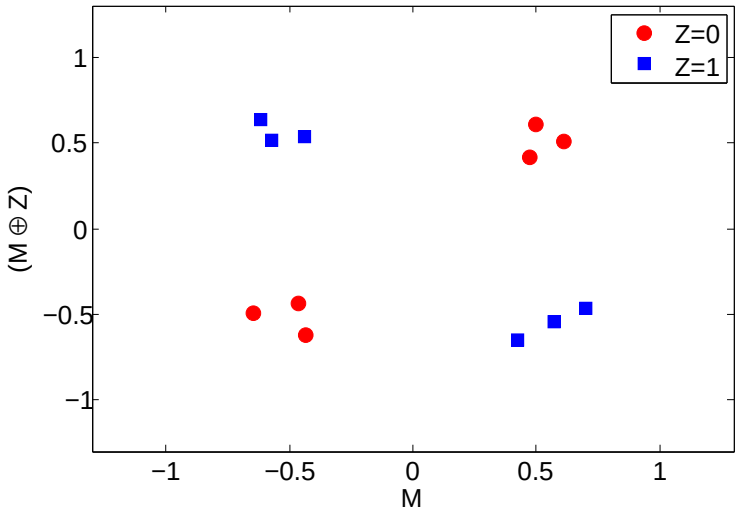


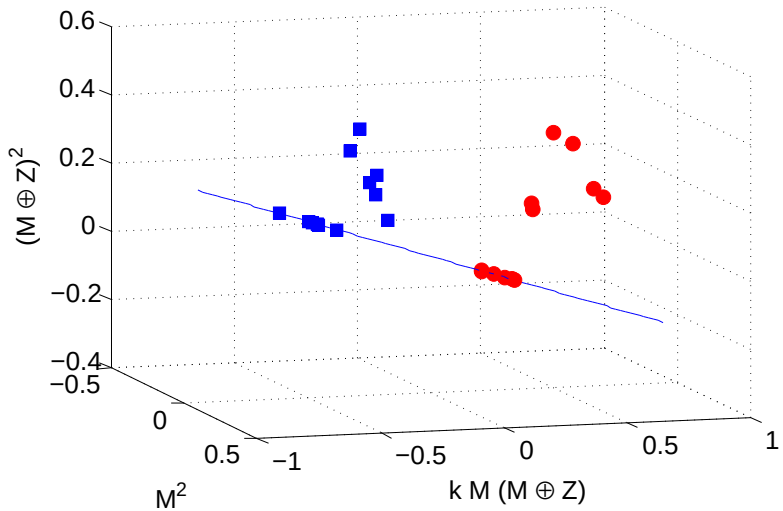
$$\Phi : \mathbb{R}^D \rightarrow \mathbb{R}^{\binom{D+d-1}{d}}$$
$$\Phi(t_1, t2) = (t_1^2, t_2^2, kt_1 t_2)$$

KDA: an intuition

Toy example: 2 time samples, 1-bit data

- t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$
- t_2: $M \oplus Z + n$ (Boolean masking)





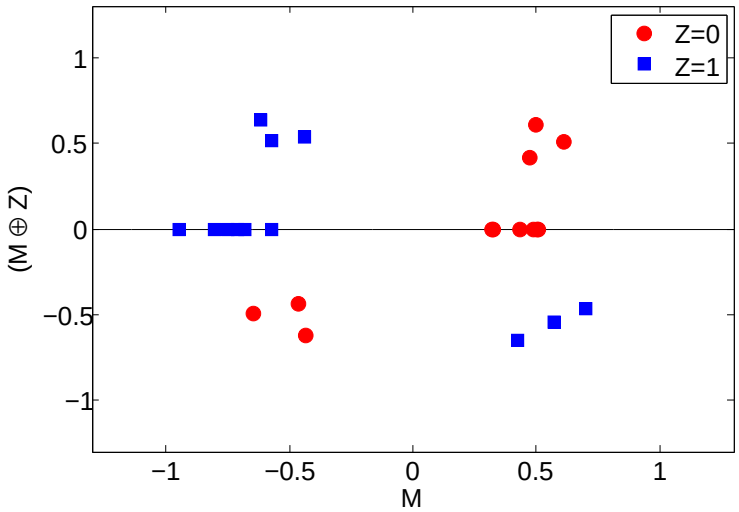
$\Phi \longrightarrow$ LDA

KDA: an intuition

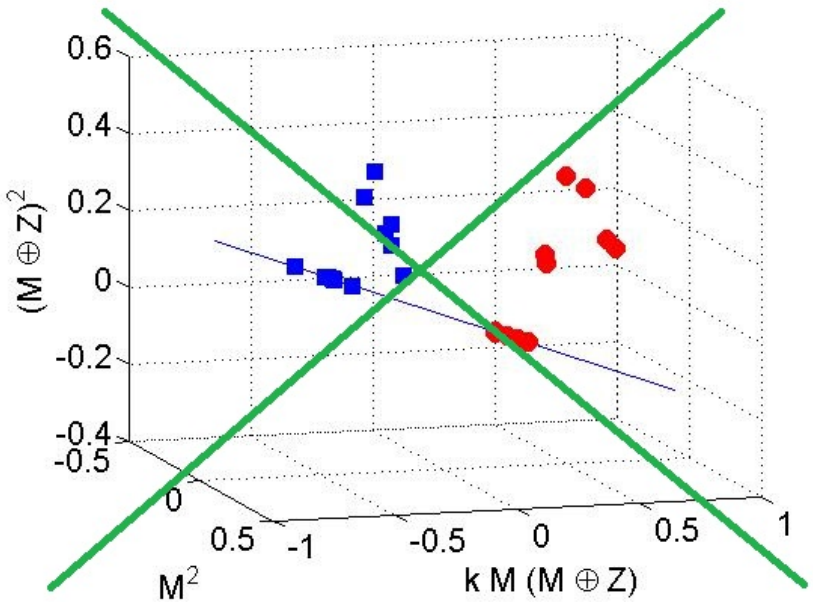
Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)



KDA
remains in $\mathbb{R}^D$

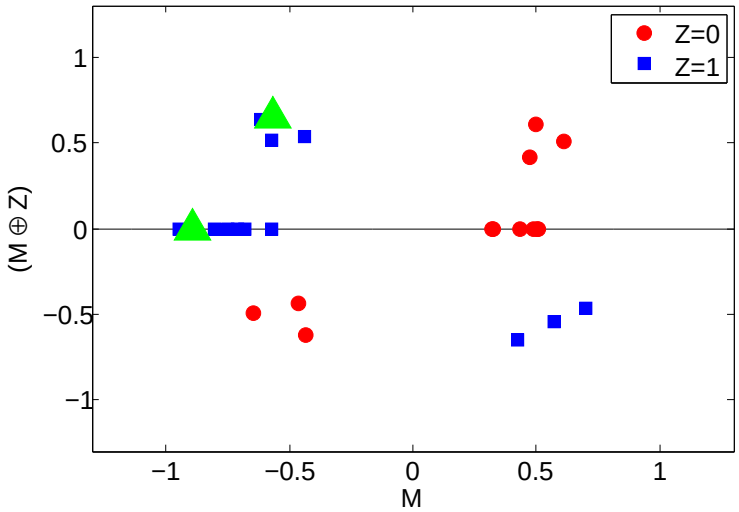


KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)



KDA procedure

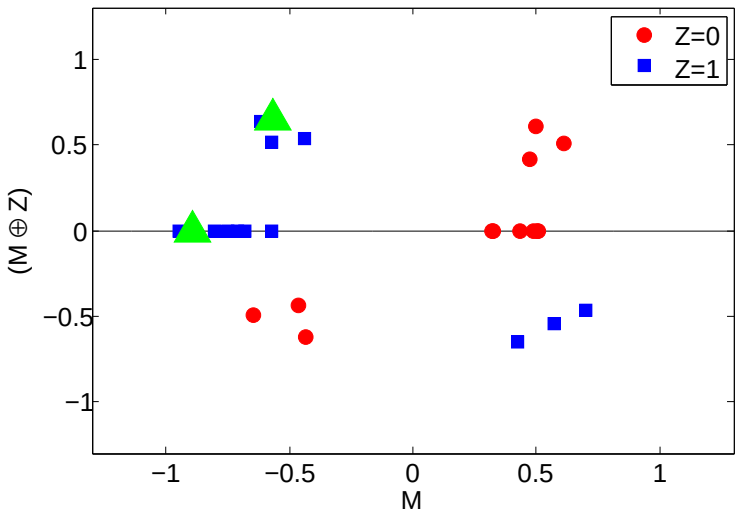
- Training $\rightarrow \boldsymbol{\nu}_\ell$, C vectors of coefficients $(\ell = 1, \ldots, C)$

KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)



KDA procedure

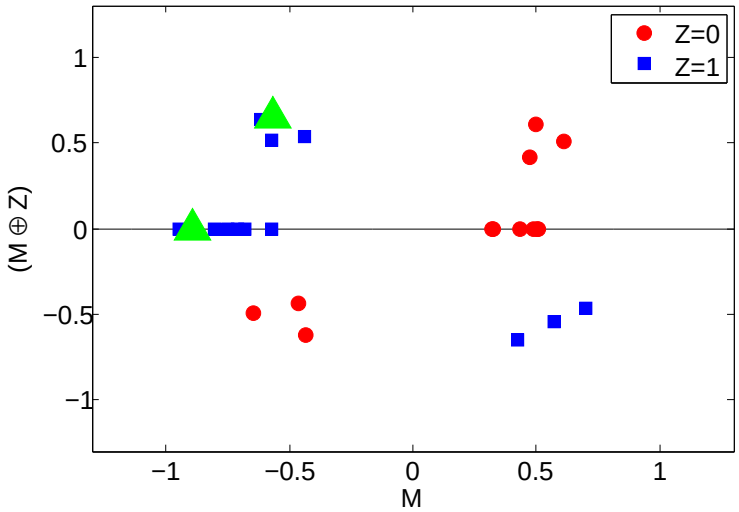
- Training $\rightarrow \boldsymbol{\nu}_\ell$, C vectors of coefficients $(\ell = 1, \ldots, C)$
- *Compare* the new trace with all training traces $K(\mathbf{x}_i^{z_i}, \mathbf{x})$, $K(\mathbf{x}_i^{z_i}, \mathbf{x})$

KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)



KDA procedure

- Training $\rightarrow \boldsymbol{\nu}_\ell$, C vectors of coefficients $(\ell = 1, \ldots, C)$
- *Compare* the new trace with all training traces $K(\mathbf{x}_i^{z_i}, \mathbf{x})$, $K(\mathbf{x}_i^{z_i}, \mathbf{x})$
- *KDA projection*

$$\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \boldsymbol{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x}) \ . \tag{3}$$

Kernel Function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{4}$$

Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

Kernel Function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{4}$$

Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

Example: 2nd Degree Polynomial Kernel Function

Toy example: $D = 2 \longrightarrow \mathbf{x}_i = [a, b]$, $\mathbf{x}_j = [c, d]$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (ac + bd)^2 = a^2c^2 + 2abcd + b^2d^2$$

Kernel Function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{4}$$

Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

Example: 2nd Degree Polynomial Kernel Function

Toy example: $D = 2 \longrightarrow \mathbf{x}_i = [a, b]$, $\mathbf{x}_j = [c, d]$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (ac + bd)^2 = a^2c^2 + 2abcd + b^2d^2$$

$K \longleftrightarrow \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$\Phi(u, v) = [u^2, \sqrt{2}uv, v^2]$$

Kernel Function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{4}$$

Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

Example: 2nd Degree Polynomial Kernel Function

Toy example: $D = 2 \longrightarrow \mathbf{x}_i = [a, b]$, $\mathbf{x}_j = [c, d]$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (ac + bd)^2 = a^2c^2 + 2abcd + b^2d^2$$

$K \longleftrightarrow \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$\Phi(a, b) = [a^2, \sqrt{2}ab, b^2]$$

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = a^2 \quad + \sqrt{2}ab \quad + b^2 \quad = K(\mathbf{x}_i, \mathbf{x}_j)$$

Kernel Function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{4}$$

Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

Example: 2nd Degree Polynomial Kernel Function

Toy example: $D = 2 \longrightarrow \mathbf{x}_i = [a, b]$, $\mathbf{x}_j = [c, d]$

$$K(\mathbf{x}_i, \mathbf{x}_j) = (ac + bd)^2 = a^2c^2 + 2abcd + b^2d^2$$

$K \longleftrightarrow \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

$$\Phi(c, d) = [c^2, \sqrt{2}cd, d^2]$$

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = a^2c^2 + \sqrt{2}ab\sqrt{2}cd + b^2d^2 = K(\mathbf{x}_i, \mathbf{x}_j)$$

KDA - the training

Between-class (inter-class) Scatter Matrix

LDA

- $\mathbf{S_B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \vec{\bar{x}})(\vec{\mu}_s - \vec{\bar{x}})^\intercal$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
-

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by: $\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i}), \qquad \vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

Within-class (inter-class) Scatter Matrix

LDA

- $\mathbf{S_B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \bar{\vec{x}})(\vec{\mu}_s - \bar{\vec{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z \in \mathcal{Z}} \sum_{i=1}^{N_z} (\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z (\mathbf{I} - \mathbf{I}_{N_z}) \mathbf{K}_z^\intercal$ [2]
-

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by: $\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

Eigenvector problem

Computational Complexity $O(D^3)$

LDA

- $\mathbf{S_B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \vec{\overline{x}})(\vec{\mu}_s - \vec{\overline{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z \in \mathcal{Z}} \sum_{i=1}^{N_z} (\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$
- $\vec{\alpha}_i$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$ $[D \times D]$

Computational Complexity $O(N^3)$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z (\mathbf{I} - \mathbf{I}_{N_z}) \mathbf{K}_z^\intercal$ [2]
- $\vec{\nu}_i$ eigenvectors of $\mathbf{N}^{-1}\mathbf{M}$ $[N \times N]$
-

---

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by: $\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

New trace projection

Computational Complexity $O(D^3)$

LDA

- $\mathbf{S_B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \vec{\bar{x}})(\vec{\mu}_s - \vec{\bar{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z \in \mathcal{Z}} \sum_{i=1}^{N_z} (\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$
- $\vec{\alpha}_i$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$ $[D \times D]$
- $\epsilon_\ell^{LDA}(\vec{x}) = \sum_{i=1}^{D} \vec{\alpha}_\ell[i]\vec{x}[i]$

Computational Complexity $O(N^3)$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z (\mathbf{I} - \mathbf{I}_{N_z}) \mathbf{K}_z^\intercal$ [2]
- $\vec{\nu}_i$ eigenvectors of $\mathbf{N}^{-1}\mathbf{M}$ $[N \times N]$
- $\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})$

---

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by: $\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i}), \qquad \vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

Computational Complexity $O(D^3)$

LDA

- $\mathbf{S_B} = \sum_{z\in\mathcal{Z}} N_z(\vec{\mu}_s - \vec{\bar{x}})(\vec{\mu}_s - \vec{\bar{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z\in\mathcal{Z}} \sum_{i=1}^{N_z}(\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$
- $\vec{\alpha}_i$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$ $[D \times D]$
- $\epsilon_\ell^{LDA}(\vec{x}) = \sum_{i=1}^{D} \vec{\alpha}_\ell[i]\vec{x}[i]$

Computational Complexity $O(N^3)$

KDA

- $\mathbf{M} = \sum_{z\in\mathcal{Z}} N_z(\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z\in\mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^\intercal \quad +\mu\mathbf{I}$
- $\vec{\nu}_i$ eigenvectors of $\mathbf{N}^{-1}\mathbf{M}$ $[N \times N]$
- $\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})$

μ regularization parameter — *Memory-based* machine

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by: $\vec{M}_z[j] = \frac{1}{N_z}\sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\vec{M}_T[j] = \frac{1}{N}\sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

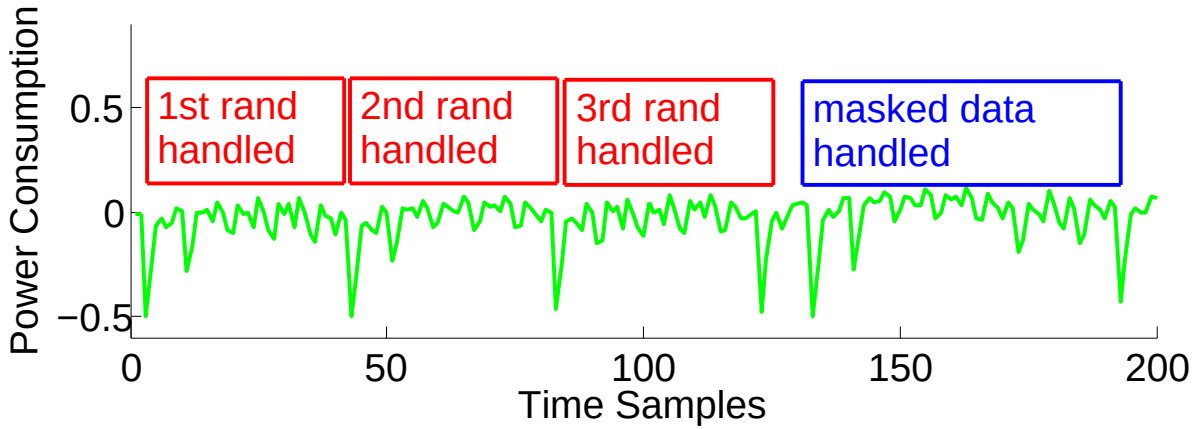
[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

Experimental setup

Target device and acquisitions:

- 8-bit AVR microprocessor Atmega328P
- power-consumption acquired via the ChipWhisperer [OC14] platform
- $D = 200$, 4 clock-cycles are selected

Sensitive variable: $Z = \mathrm{Sbox}_{\mathrm{AES}}(P \oplus K^{\star})$



Efficiency/Accuracy trade-off

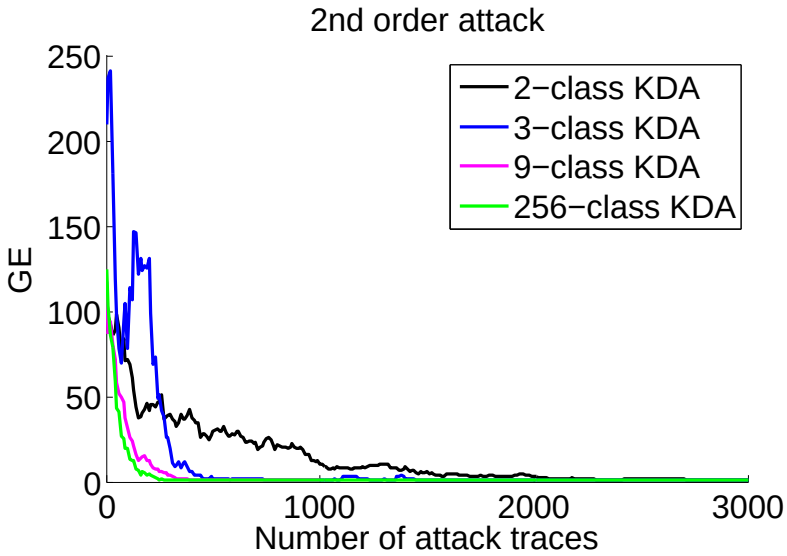
KDA training set size fixed ($\sim$ 9000 traces) to control efficiency

Adjust the number of classes to gain in accuracy

$Z = 0, \ldots, 255$

Model	number of classes	traces per class
Value	256	35
HW	9	1000
HW $<, >, = 4$	3	3000
HW $\leq, > 4$	2	4500

Attack of second order

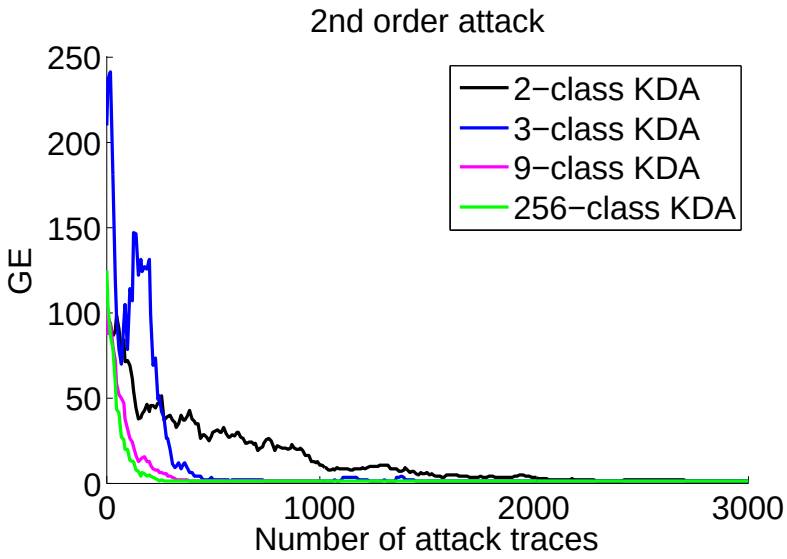


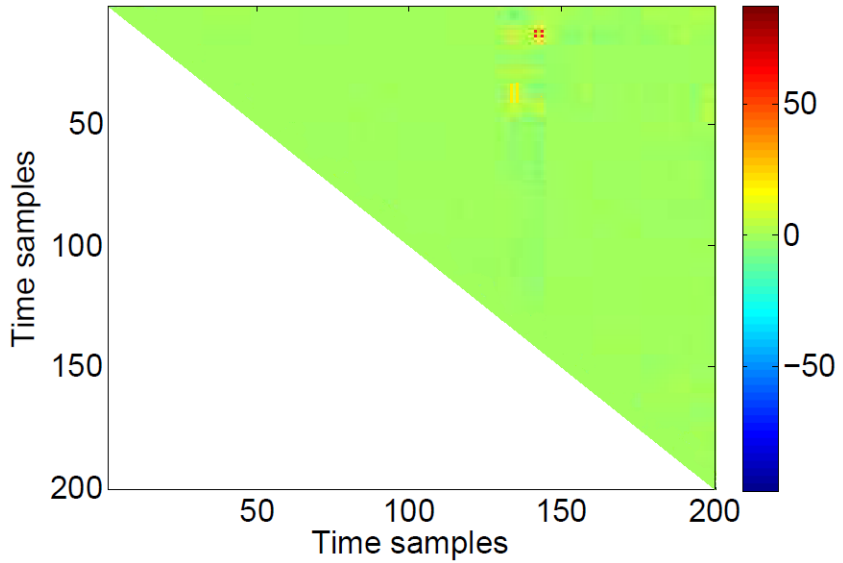
Implicit coefficients

$$\epsilon_\ell^{\mathrm{KDA}}(\mathbf{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x}) = \sum_{j=1}^{D}\sum_{k=1}^{D}[\ \underbrace{(\mathbf{x}[j]\mathbf{x}[k])}_{\text{multiplied time samples}}\ \underbrace{(\sum_{i=1}^{N} \boldsymbol{\nu}_\ell[i]\mathbf{x}_i[j]\mathbf{x}_i[k])}_{\text{implicit coefficients}}] \quad (5)$$

$d = 2 \longrightarrow \binom{200+2-1}{2} = 20.100$ implicit coefficients

Attack of second order



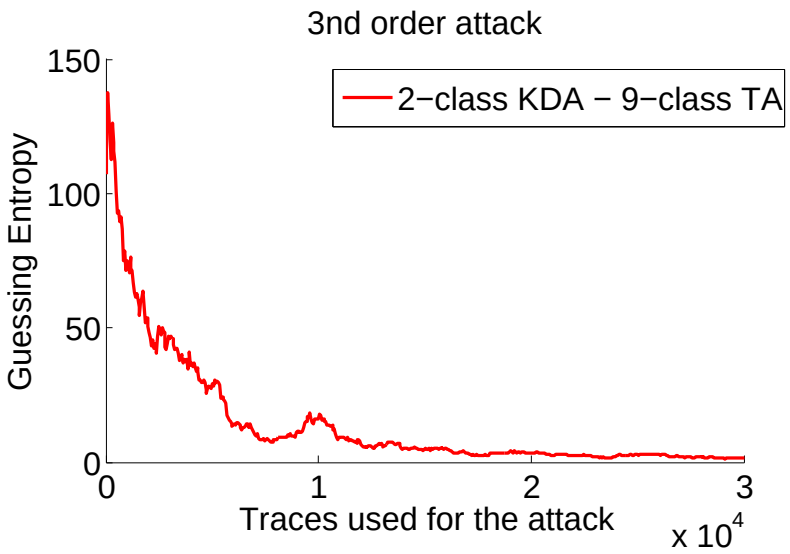


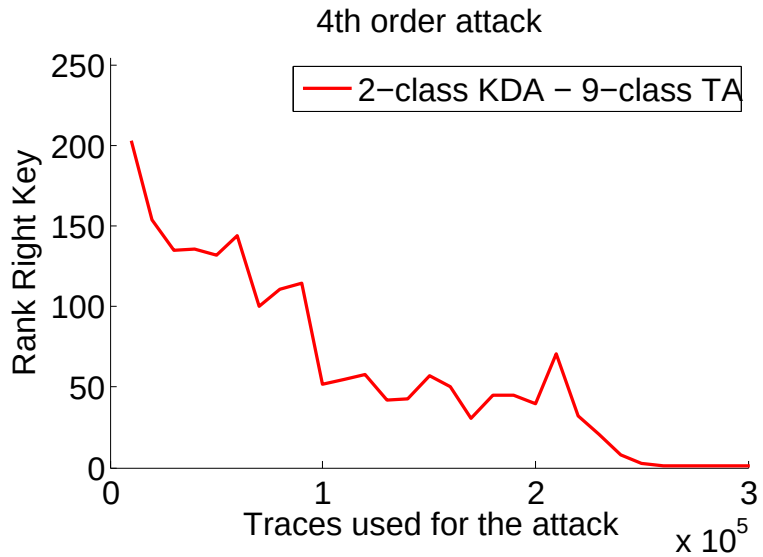
Implicit coefficients

$$\epsilon_\ell^{\mathrm{KDA}}(\mathbf{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x}) = \sum_{j=1}^{D}\sum_{k=1}^{D}[\underbrace{(\mathbf{x}[j]\mathbf{x}[k])}_{\substack{\text{multiplied} \\ \text{time samples}}} \underbrace{(\sum_{i=1}^{N} \boldsymbol{\nu}_\ell[i]\mathbf{x}_i[j]\mathbf{x}_i[k])}_{\text{implicit coefficients}}] \quad (5)$$

$d = 2 \longrightarrow \binom{200+2-1}{2} = 20.100$ implicit coefficients

Third and Fourth Order



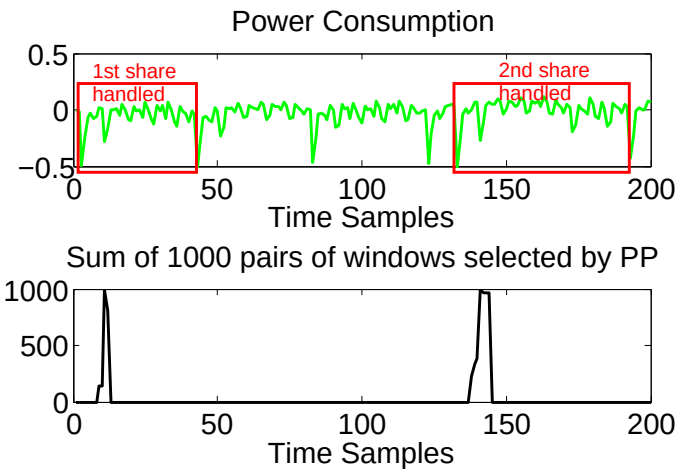


- $d = 3 \longrightarrow \binom{200+3-1}{3} = 1.353.400$ implicit coefficients
- $d = 4 \longrightarrow \binom{200+4-1}{4} = 68.685.050$ implicit coefficients

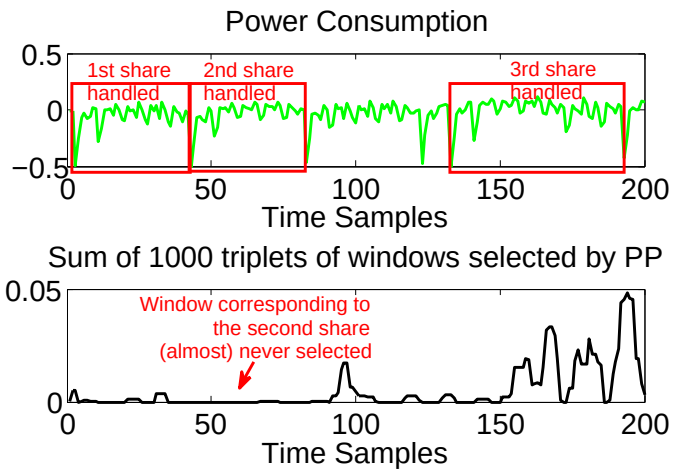
Same time of execution of the KDA algorithm!

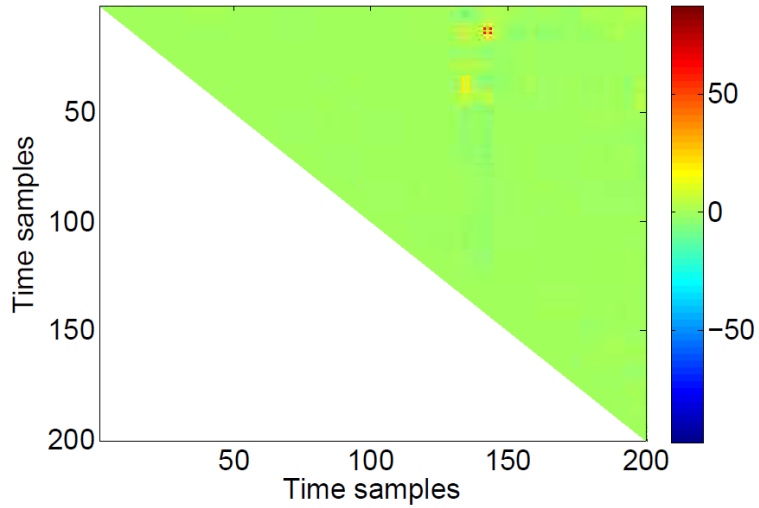
Qualitative comparison with PP

Second Order



Third Order





Conclusions on KDA

Strong points

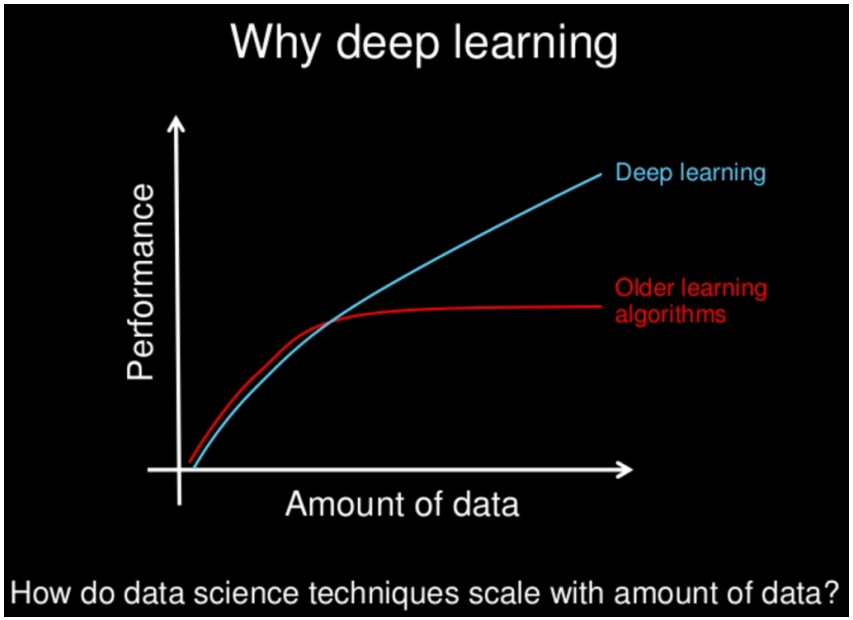
- KDA is suitable to attack $(d-1)$th-order masking
- Choice of the d-th degree polynomial kernel function
- KDA computational complexity is independent from the order d
- Tested and effective on a real case, positively compared to PP

Limits and drawbacks

- Memory-based + $O(N^3)$ complexity $\rightarrow$ Non-scalability to big training set $\rightarrow$ not suitable for highly-noisy higher-order masked signals
- Regularization hyper-parameter μ (and slow *validation*)

Contents

Motivations (1)



- not memory-based
- parallelizable computation(GPU optimizations)
- many hyper-parameters but faster validation

Motivations (2)

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X}))\mid Z=z}$, $p_{\epsilon(\rho(\vec{X}))}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
- $p_{Z\mid \epsilon(\rho(\vec{x}))}$ (discriminative model)

Many independent preprocessing steps and assumptions

Motivations (2)

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X}))\mid Z=z}$, $p_{\epsilon(\rho(\vec{X}))}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
- $p_{Z\mid\vec{X}}$ (discriminative model)
  by means of a neural network $F(\vec{X}) \approx p_{Z\mid\vec{X}}$ (Universal approximation theorem)

DEEP LEARNING

Many independent preprocessing steps and assumptions

Motivations (2)

Profiling phase

- manage de-synchronization problem [$\mathcal{D}_{\text{train}} \longrightarrow \rho \colon \mathbb{R}^D \to \mathbb{R}^D$]
- mandatory dimensionality reduction [$\mathcal{D}_{\text{train}} \longrightarrow \epsilon \colon \mathbb{R}^D \to \mathbb{R}^C$]
- estimate
  - $p_{\epsilon(\rho(\vec{X})) \mid Z=z}$, $p_{\epsilon(\rho(\vec{X}))}$, p_Z (generative model)
    - Gaussian hypothesis [CRR03]
    - Variants: *pooled* version [CK14], linear regression [SLP05]
- $p_{Z \mid \vec{X}}$ (discriminative model)
  by means of a neural network $F(\vec{X}) \approx p_{Z \mid \vec{X}}$ (Universal approximation theorem)

DEEP LEARNING

Many independent preprocessing steps and assumptions
$\longleftrightarrow$ integrated and agnostic approach

Multi-Layer Perceptron

Multi-Layer Perceptron (MLP)

$$F(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx \Pr[Z|\vec{X} = \vec{x}]$$

Multi-Layer Perceptron

Multi-Layer Perceptron (MLP)

$$F(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx \Pr[Z|\vec{X} = \vec{x}]$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

Multi-Layer Perceptron

Multi-Layer Perceptron (MLP)

$$F(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx \Pr[Z|\vec{X} = \vec{x}]$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions

Multi-Layer Perceptron

Multi-Layer Perceptron (MLP)

$$F(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx \Pr[Z|\vec{X} = \vec{x}]$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions

s normalizing *softmax* function

Multi-Layer Perceptron

Multi-Layer Perceptron (MLP)

$$F(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx \Pr[Z|\vec{X} = \vec{x}]$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions

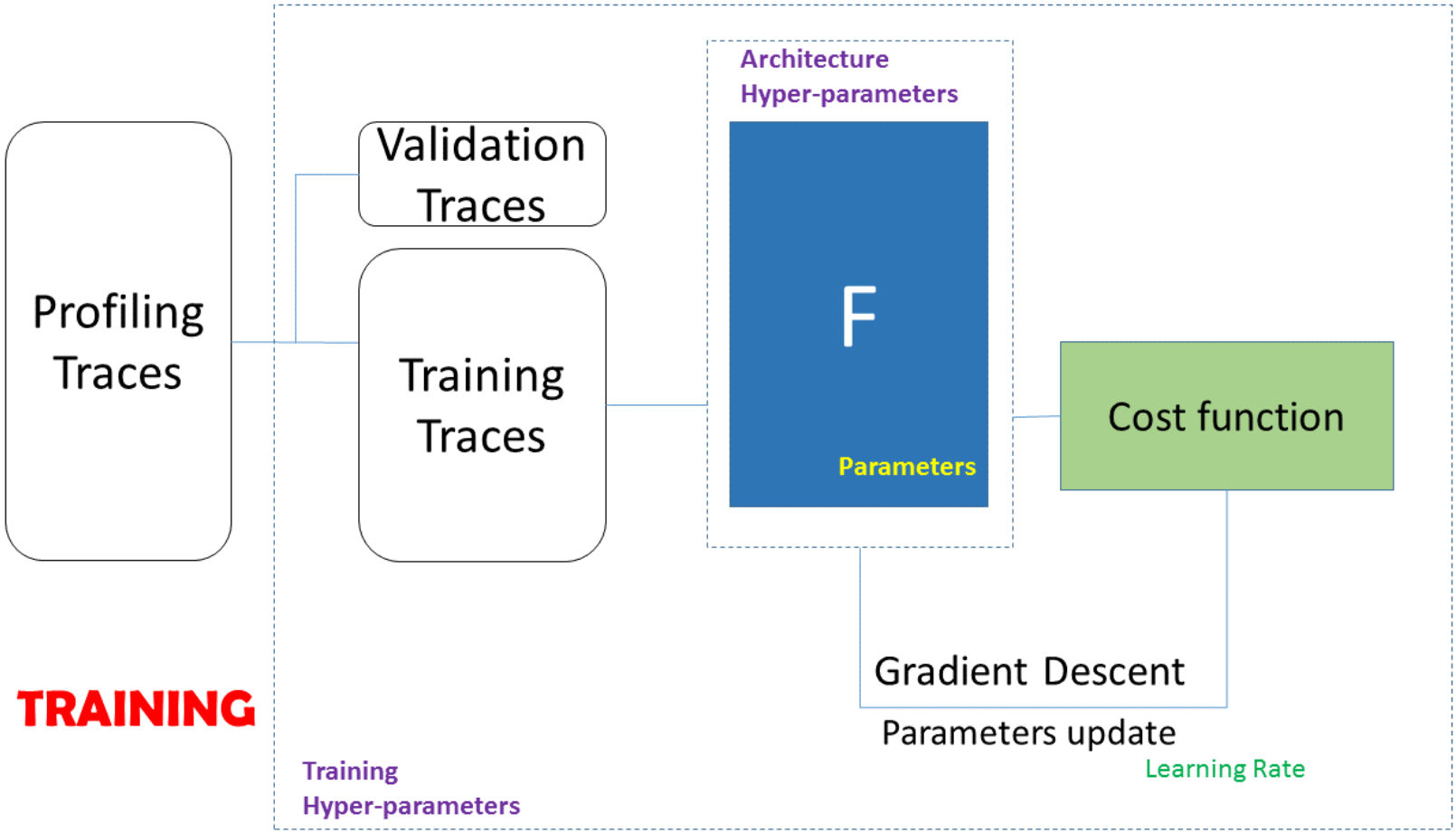
s normalizing *softmax* function

Softmax $\sim$ multi-class logistic sigmoid

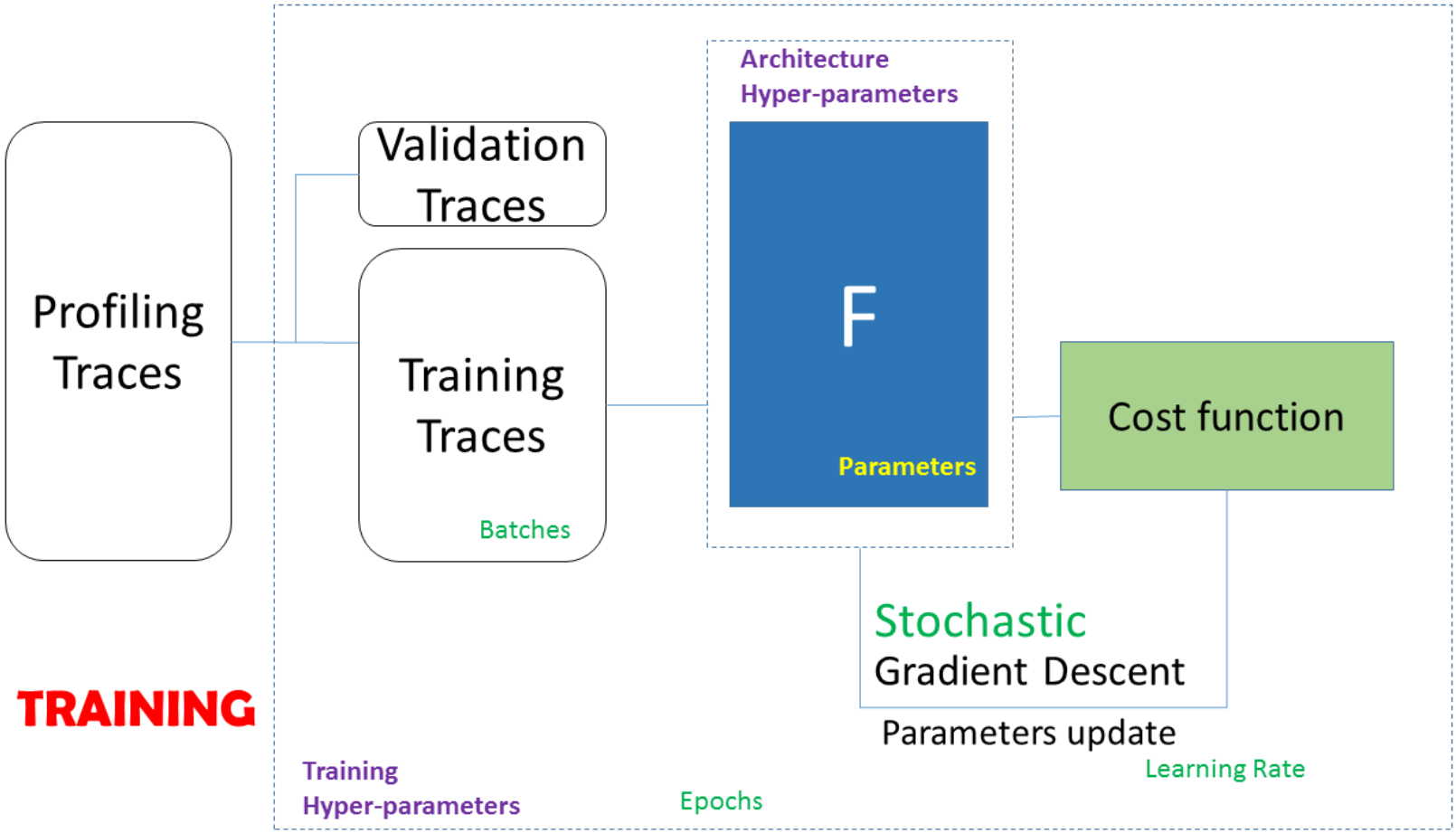
$$s(\mathbf{a})[k] = \frac{e^{\mathbf{a}[k]}}{\sum_{j=1}^{|\mathcal{Z}|} e^{\mathbf{a}[j]}} \ . \tag{6}$$

$$\Pr(s_j \mid \vec{x}) = \frac{\Pr(\vec{x} \mid s_j)\Pr(s_j)}{\Pr(\vec{x})} = \frac{\Pr(\vec{x} \mid s_j)\Pr(s_j)}{\sum_{k=1}^{|\mathcal{Z}|} \Pr(\vec{x} \mid s_k)\Pr(s_k)} = s(\mathbf{a})[j] \ , \tag{7}$$

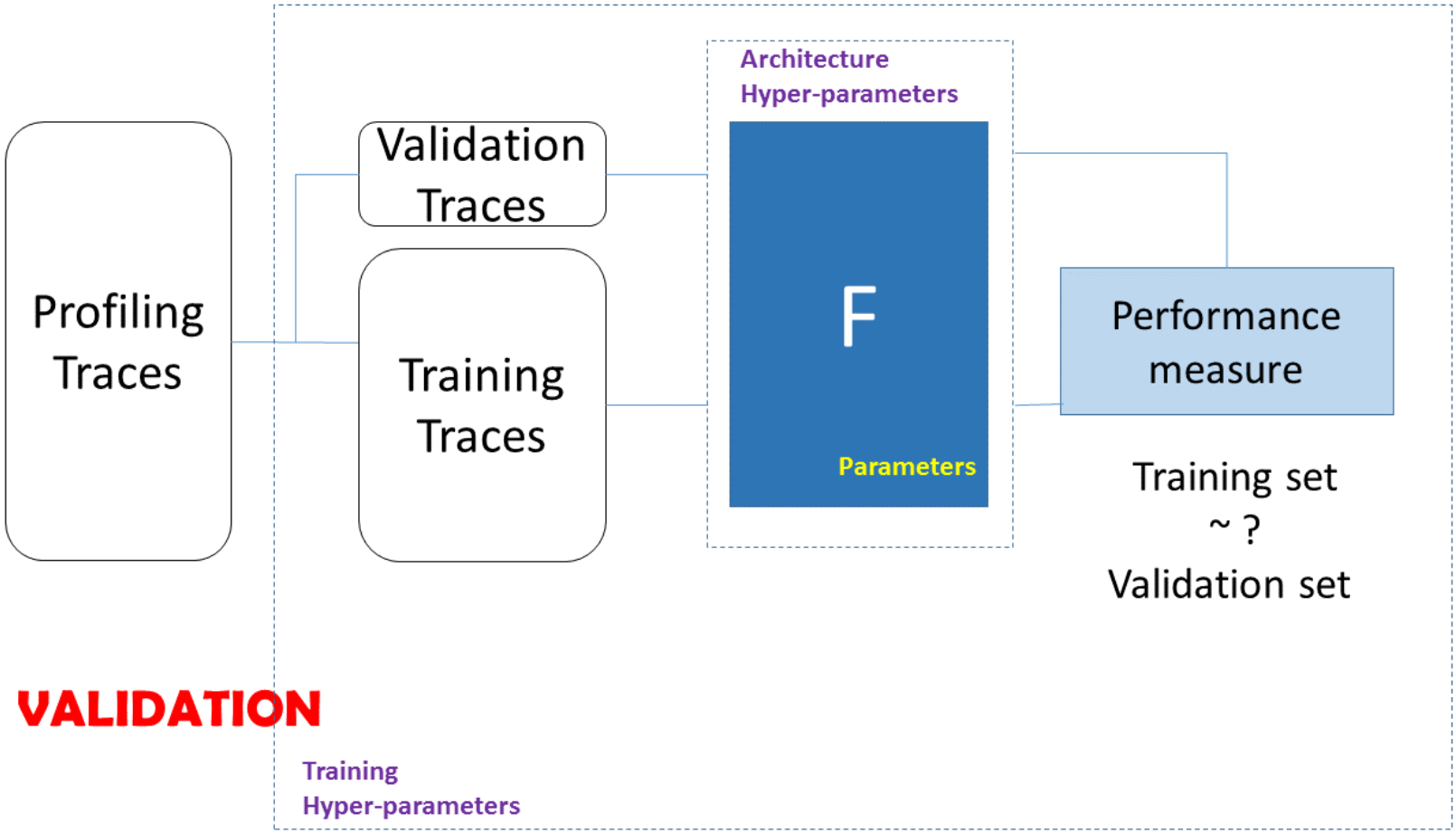
Training-Validation-Test



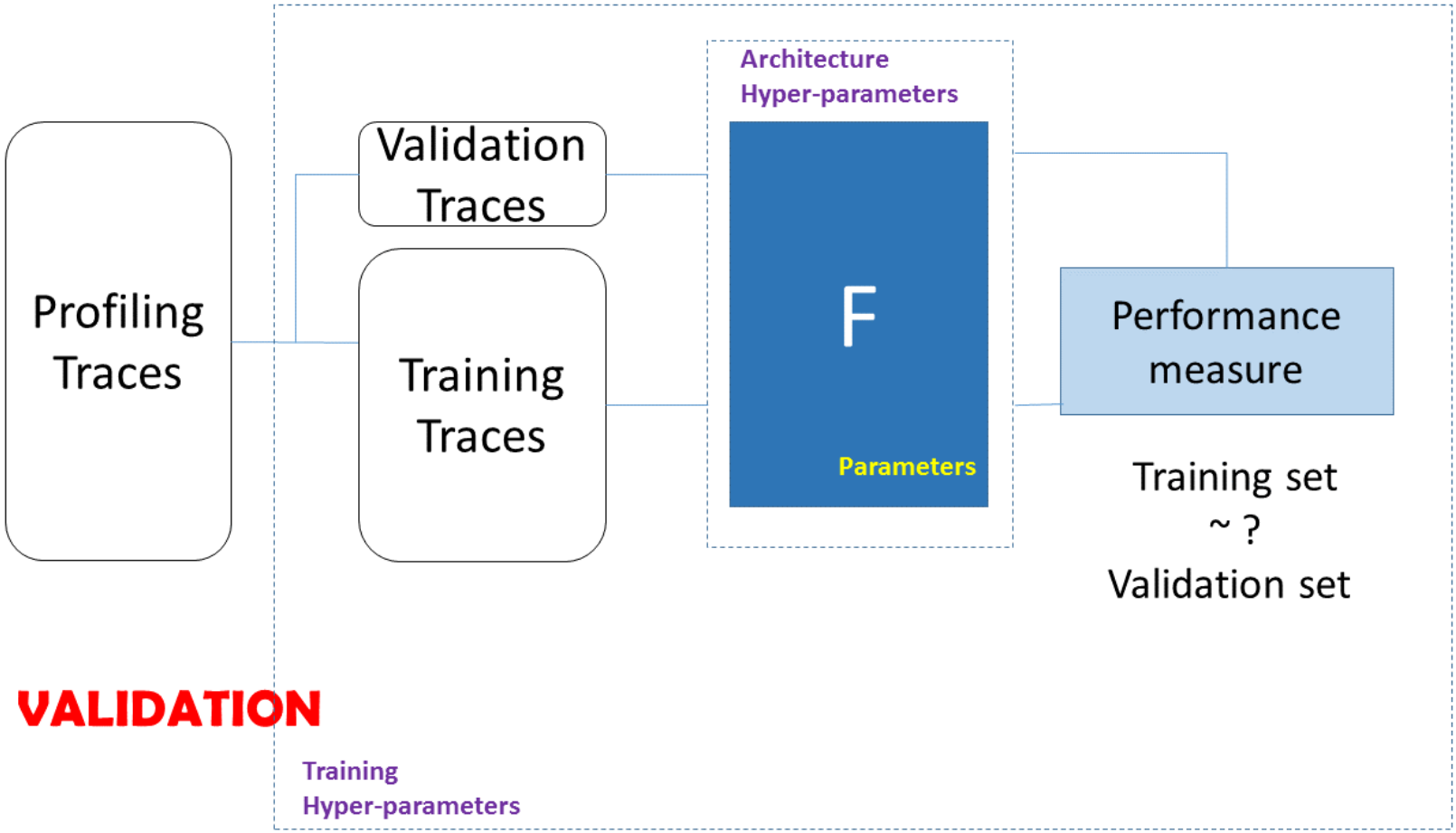
Training-Validation-Test



Training-Validation-Test



Training-Validation-Test



Cost function - Cross-entropy

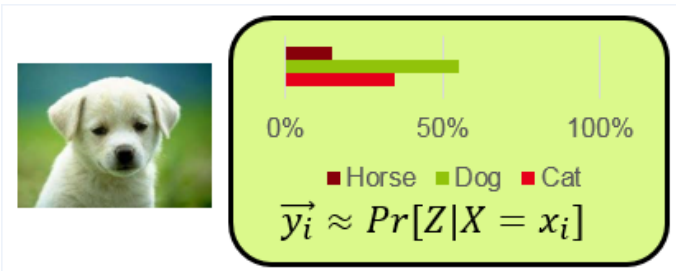
- batch of training data $(\vec{x}_i, z_i)_{i\in I}$, outputs of the current model $(\vec{y}_i)_{i\in I}$
- labels $z_i = s_j$ are *one-hot encoded*: $\vec{z}_i = \vec{s}_j = (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$

Loss function

$$\mathcal{L} = -\frac{1}{|I|} \sum_{i\in I} \sum_{t=1}^{|\mathcal{Z}|} \vec{z}_i[t] \log \vec{y}_i[t] \tag{8}$$

Maximum-*a-posteriori* or Cross-entropy

- $\vec{y}_i \approx \Pr[Z \mid \vec{X} = \vec{x}_i]$



Cost function - Cross-entropy

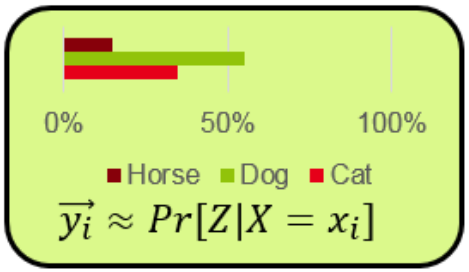
- batch of training data $(\vec{x_i}, z_i)_{i \in I}$, outputs of the current model $(\vec{y_i})_{i \in I}$
- labels $z_i = s_j$ are *one-hot encoded*: $\vec{z_i} = \vec{s_j} = (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$

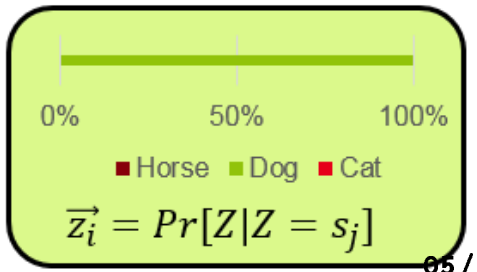
Loss function

$$\mathcal{L} = -\frac{1}{|I|} \sum_{i \in I} \sum_{t=1}^{|\mathcal{Z}|} \vec{z_i}[t] \log \vec{y_i}[t] \tag{8}$$

Maximum-*a-posteriori* or Cross-entropy

- $\vec{y_i} \approx \Pr[Z \mid \vec{X} = \vec{x_i}]$
- $\vec{z_i} \approx \Pr[Z \mid Z = \vec{s_j}]$
- $\mathbb{H}(\vec{z_i}, \vec{y_i}) = \mathbb{H}(\vec{z_i}) + D_{KL}(\vec{z_i} || \vec{y_i}) = \mathbb{E}_{\vec{z_i}}[-\log \vec{y_i}] = -\sum_{t=1}^{|\mathcal{Z}|} \vec{z_i}[t] \log \vec{y_i}[t]$





Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=44.228280, MSE_test=330.984916

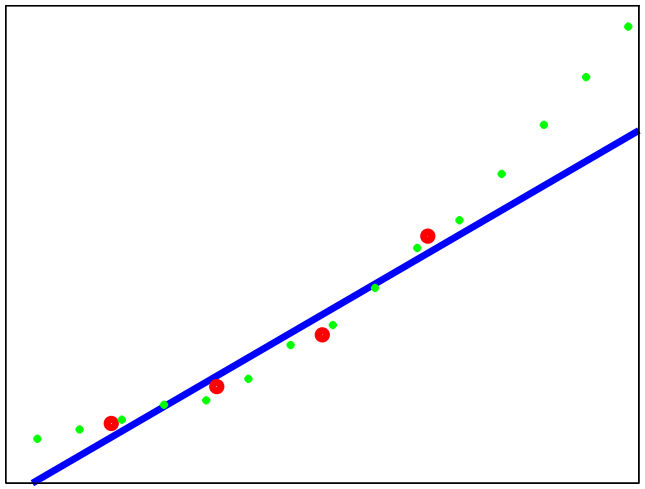


Figure: Linear regression → underfitting

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=44.228280, MSE_test=330.984916

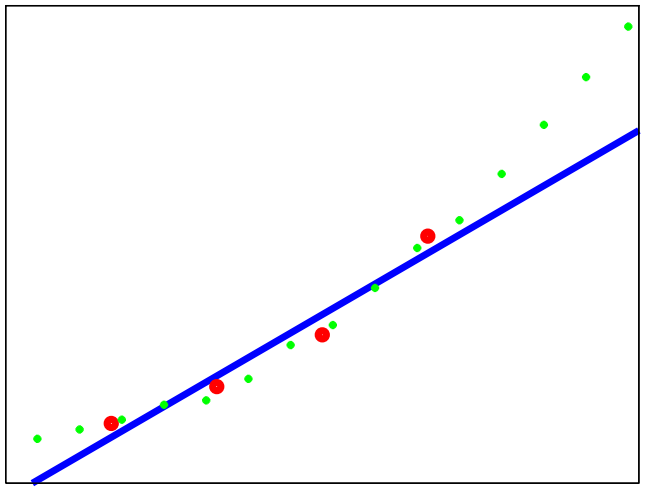


Figure: Linear regression → underfitting

MSE_train=2.243097, MSE_test=61.891672

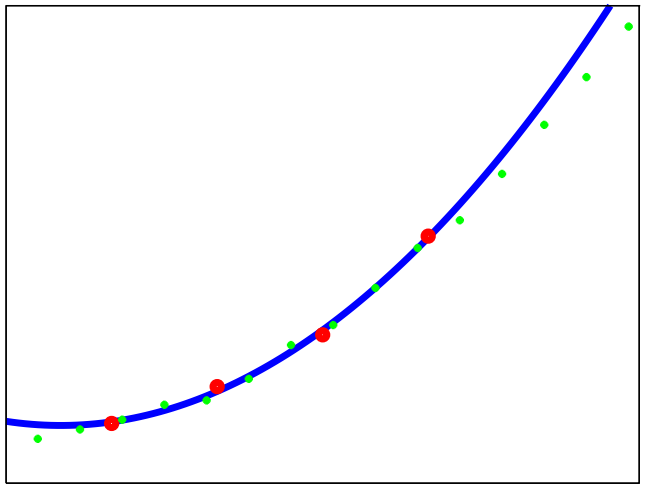


Figure: Quadratic regression → fits

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=0, MSE_test=970.081580

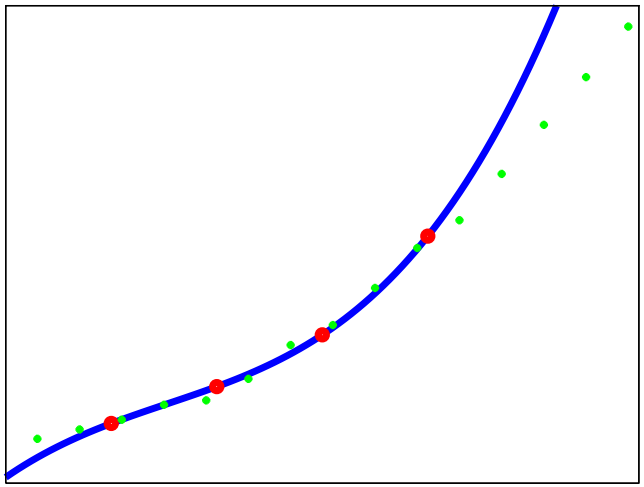


Figure: Cubic regression → overfitting

MSE_train=2.243097, MSE_test=61.891672

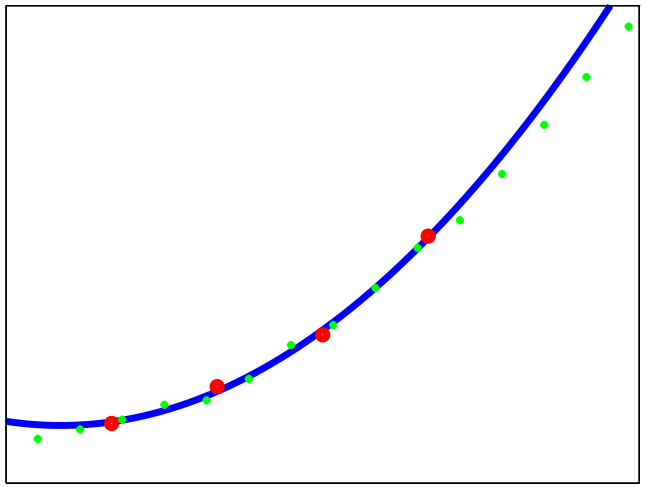


Figure: Quadratic regression → fits

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=0, MSE_test=970.081580

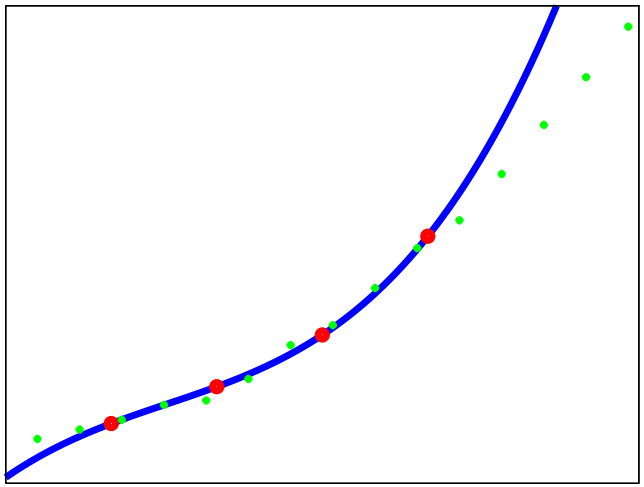


Figure: Cubic regression → overfitting

MSE_train=3.040333, MSE_test=58.377719

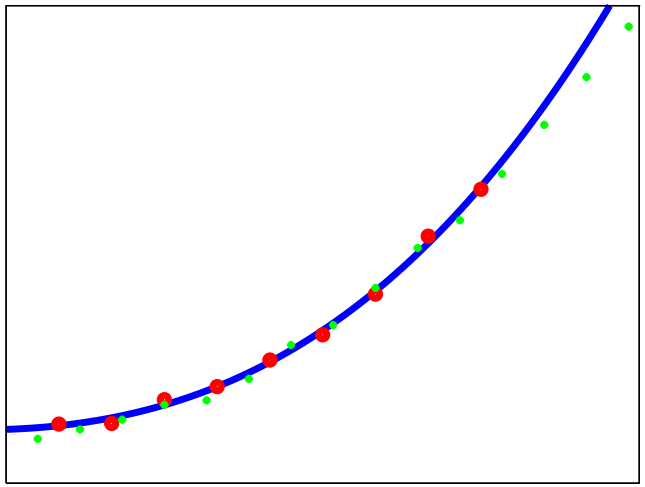


Figure: Cubic regression with more training data

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

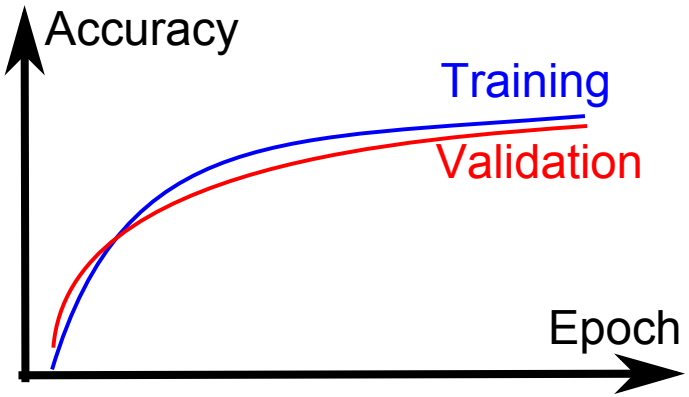
Classification via Neural Network

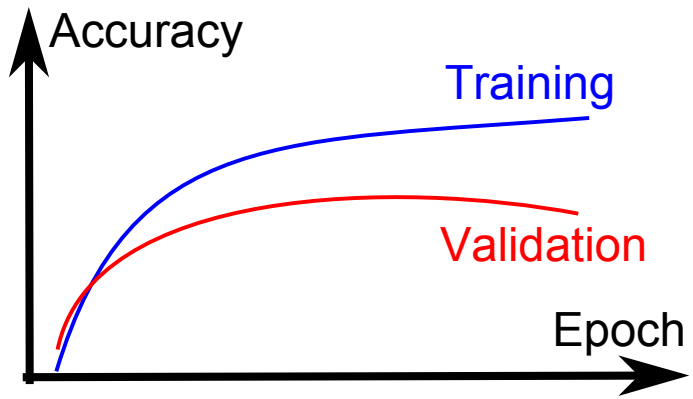
Performance measure: Accuracy (Classification rate)

Evaluate and compare training and validation accuracy

Understand significant features

Learn by heart (**OVERFITTING**)





Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

Classification via Neural Network

Performance measure: Accuracy (Classification rate)

Evaluate and compare training and validation accuracy

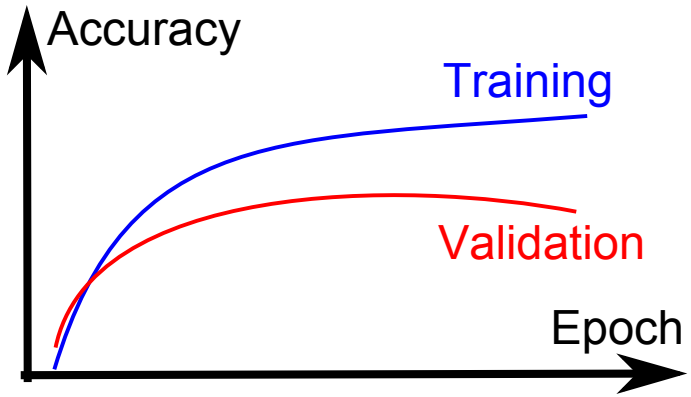
Learn by heart (**OVERFITTING**)

Why?

- Too complex model
- Not enough training data

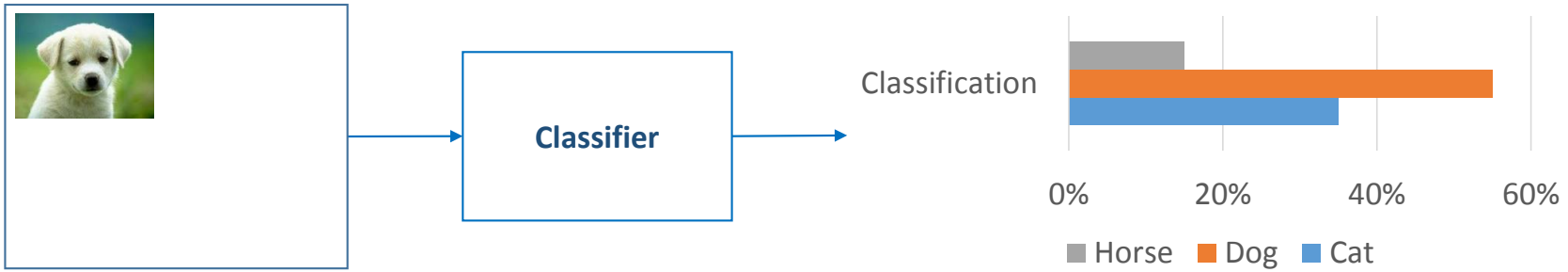
Solution?

- Data augmentation



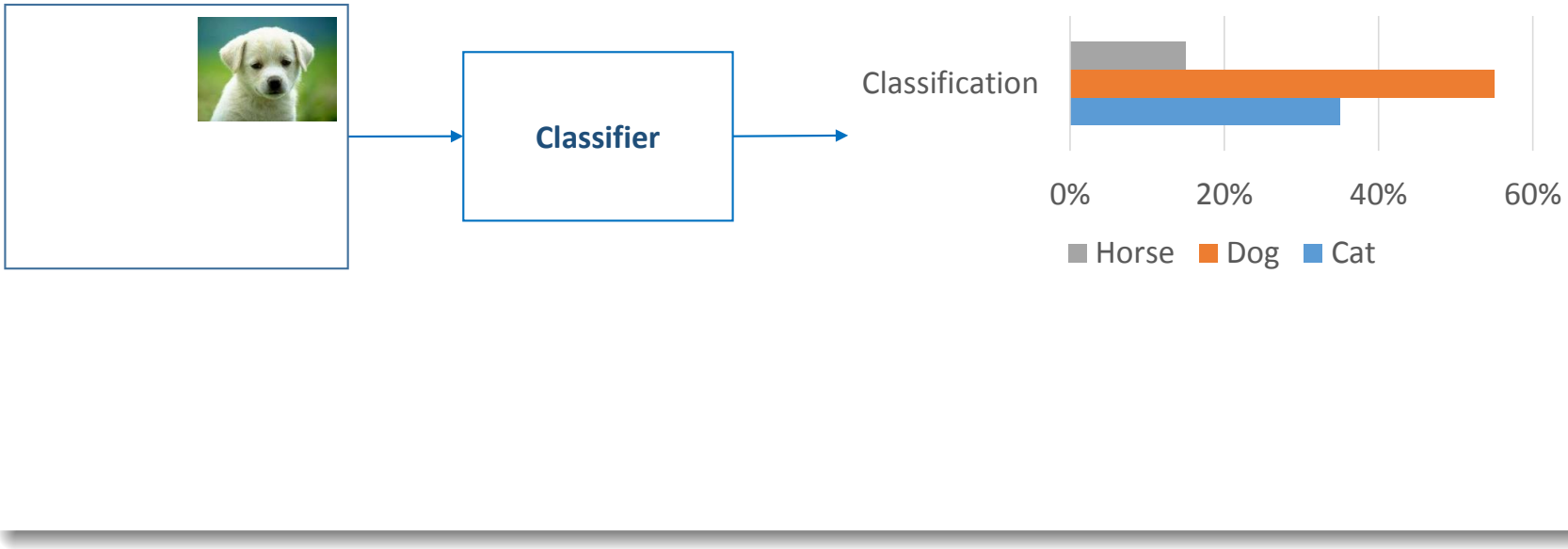
Convolutional Neural Networks

Translation-invariance



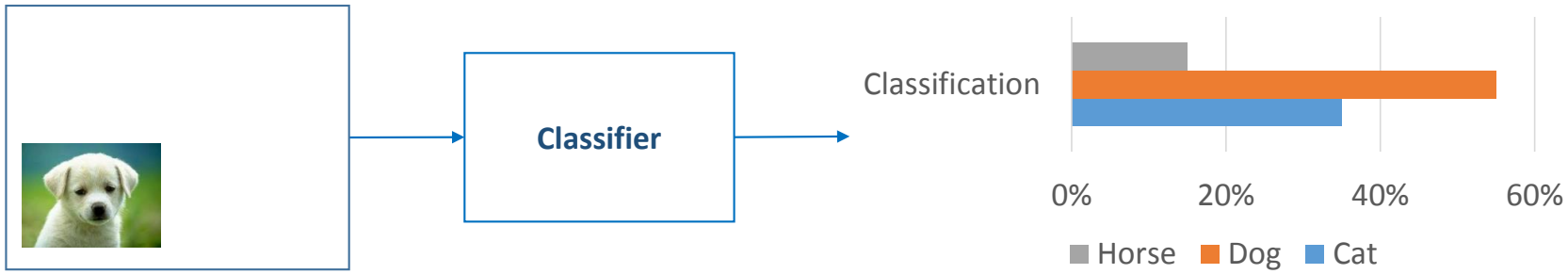
Convolutional Neural Networks

Translation-invariance



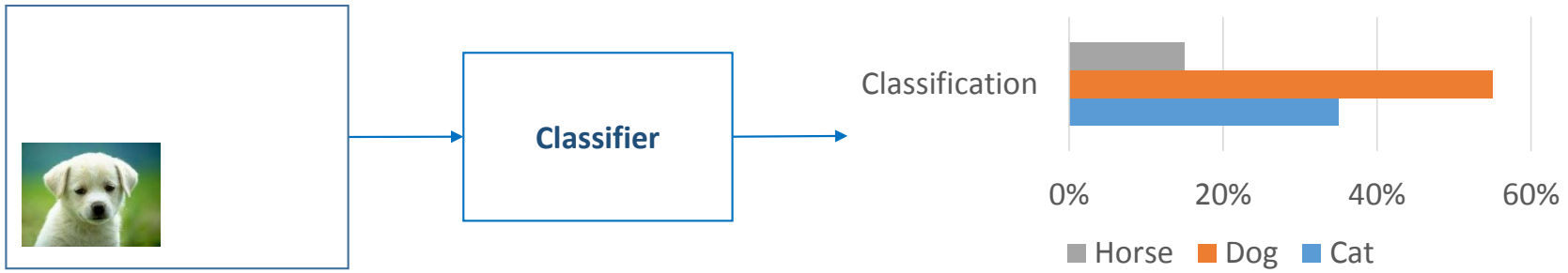
Convolutional Neural Networks

Translation-invariance



Convolutional Neural Networks

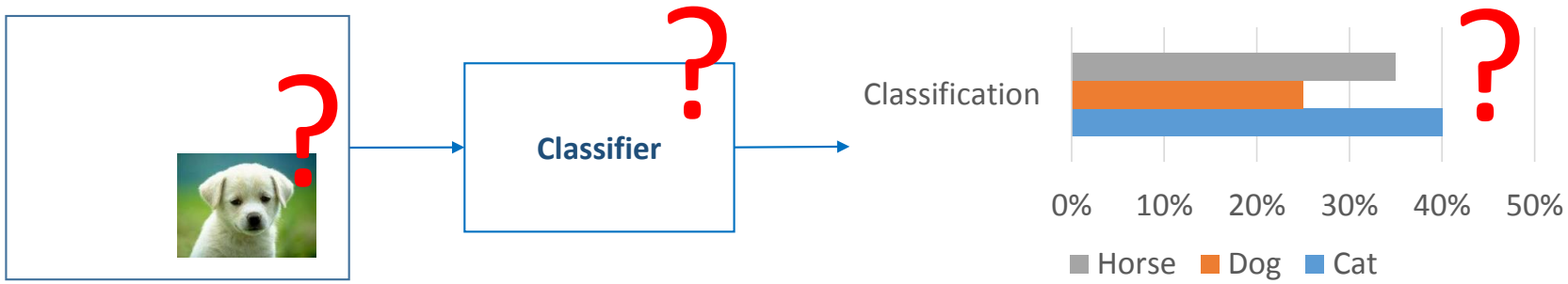
Translation-invariance



It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance



It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

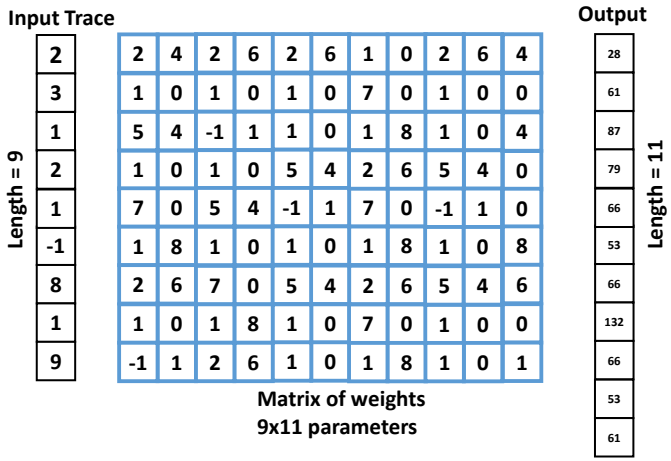
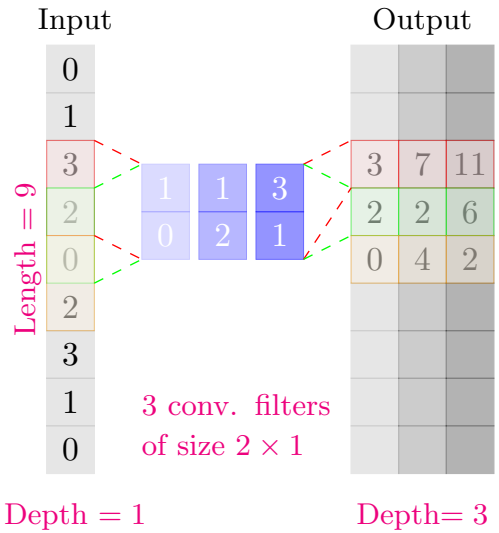


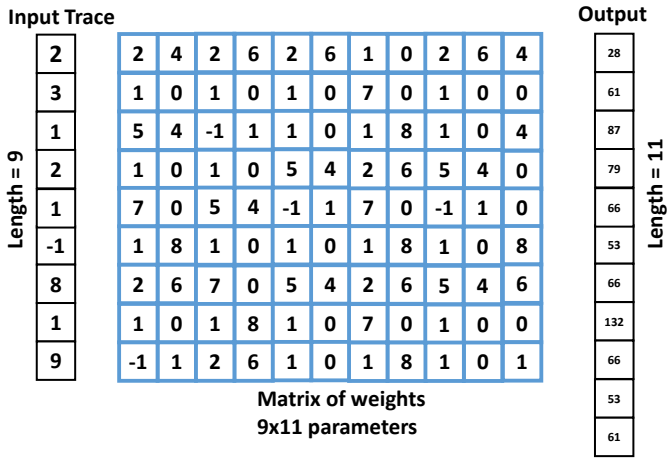
Figure: Linear layer in an MLP (Fully Connected)

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

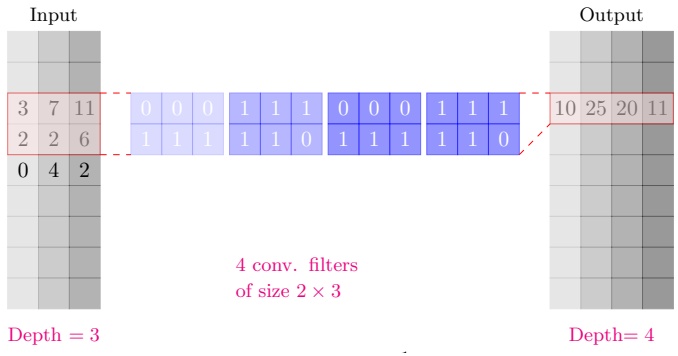




Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space



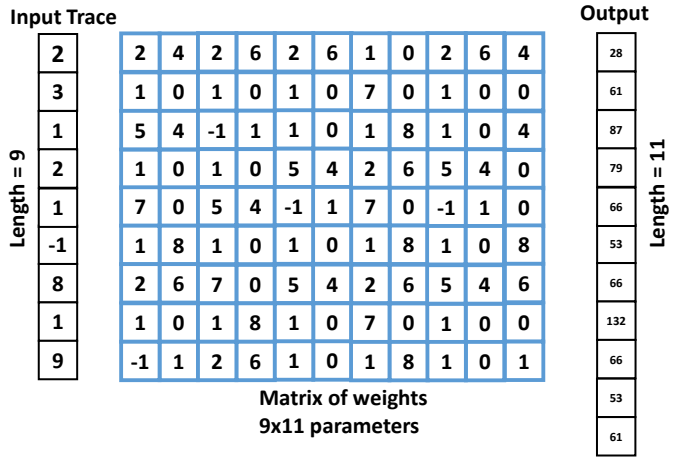
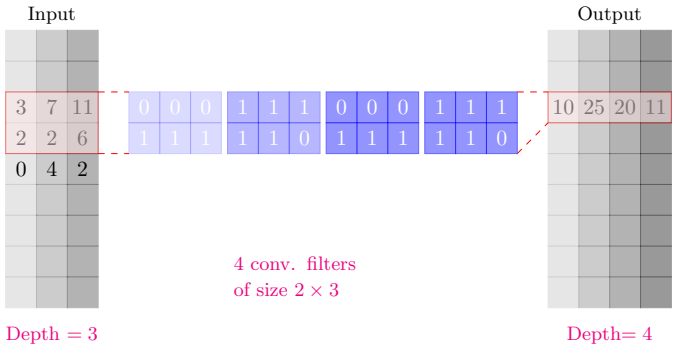


Figure: Linear layer in a ConvNet (*Convolutional Layer*)

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space



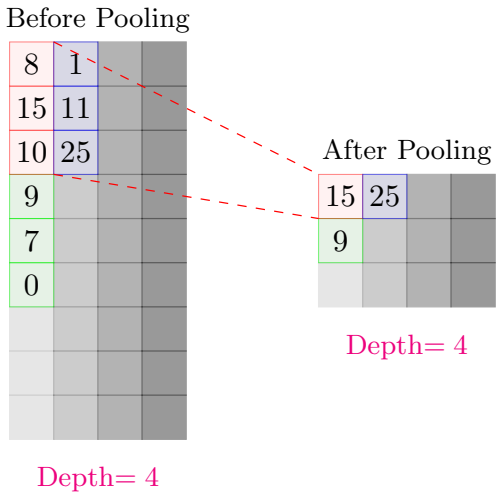
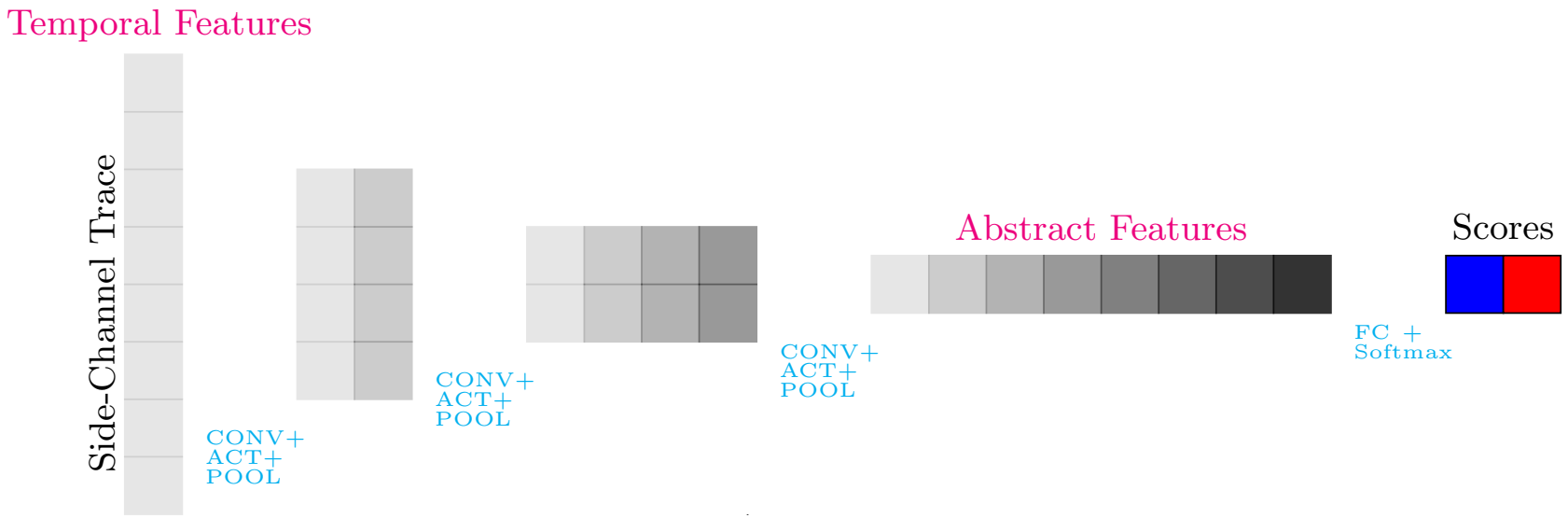


Figure: Linear layer in a ConvNet (*Convolutional Layer*)

A kind of CNN architecture



VGG-like [**simonyan2014very**]:

- Reduce temporal features to only one
- Maintain time complexity of each layer (one-half pooling when number of feature maps are doubled)
- Small filters

Model used in our experiments

$s \circ [\lambda]^1 \circ [\delta \circ [\sigma \circ \gamma]^1]^4$

Data Augmentation

Data Augmentation

Artificially generate new training data by deforming those previously acquired,
Applying transformations that preserve the label Z

Countermeasure Emulation Idea

Emulate the effects of misaligning countermeasures to generate new traces

SHIFTING

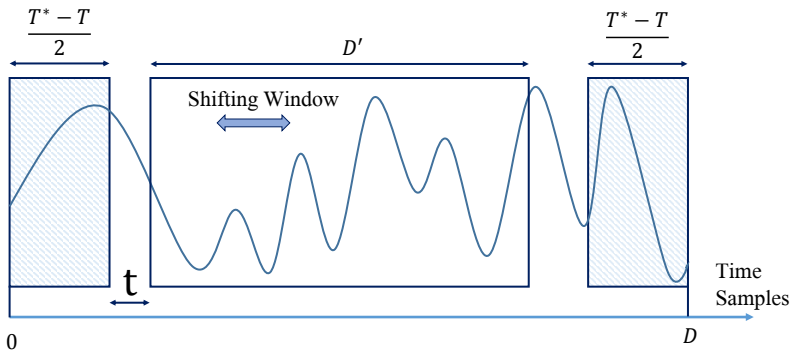


Figure: SH_T

ADD-REMOVE

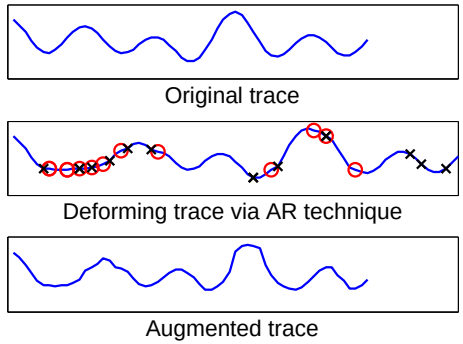


Figure: AR_R

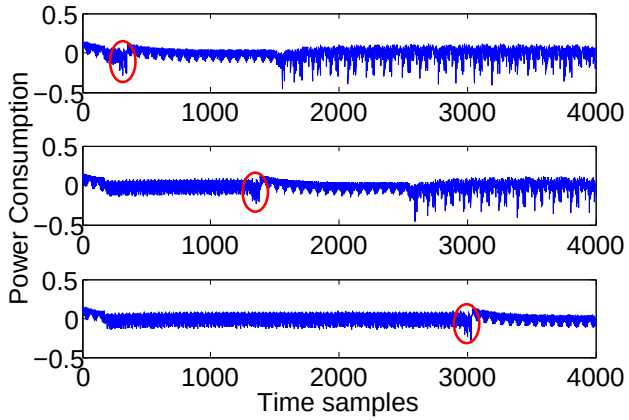
Parameter T: ♯ of possible positions
Parameter R: ♯ of added and removed points
Data Augmentation techniques are applied online during training phase

Experimental Results

- Random delays
- Artificial Jitter
- Real Jitter

Keras 1.2.1 library with Tensorflow backend [Cho+15] (open source, today 2.2.4)

Random delays



(a) One leaking operation

Setup

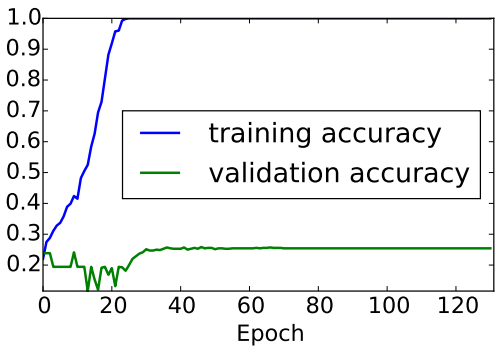
- Target Chip: Atmega328P
- Target Variable: $Z = \mathrm{HW}(\mathrm{Sbox}(P \oplus K))$
- Acquisition: through *ChipWhisperer*® platform, $\approx 4,000$ time samples
- Countermeasure: Random Delays - insertion of r *nop* operations, $r \in [0, 127]$ uniform random
- $1,000$ training traces

Random delays

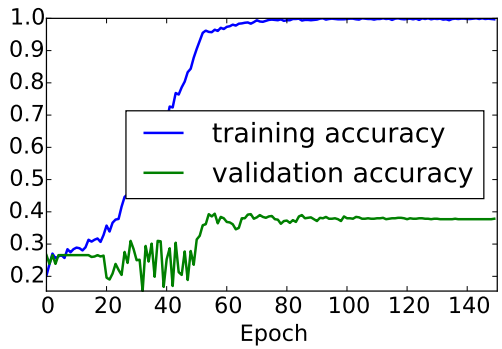
Data augmentation vs overfitting

Metrics

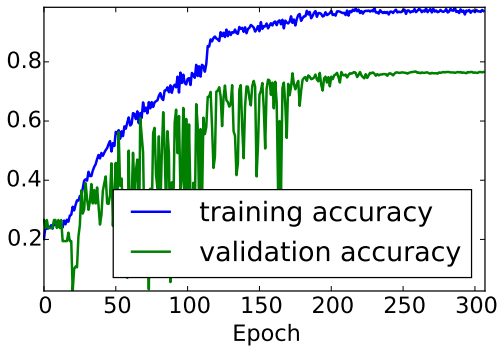
- Test accuracy: classification accuracy over the attack traces
- $N^{\star}$: minimum number of attack traces to make *guessing entropy* of the right key permanently equal to one ($N^{\star}$ estimated over 10 independent attacks)



SH_0



SH_{100}



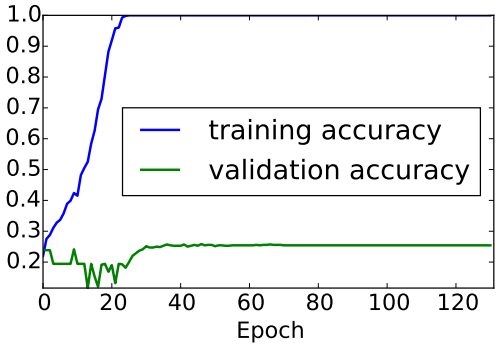
SH_{500}

Random delays

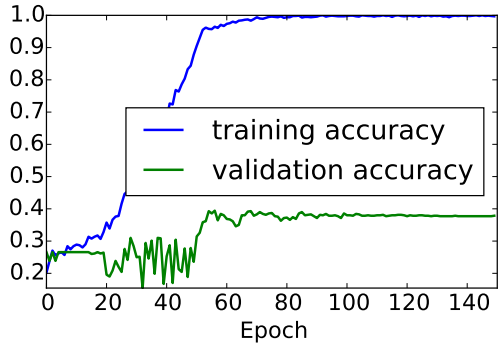
Data augmentation vs overfitting

Metrics

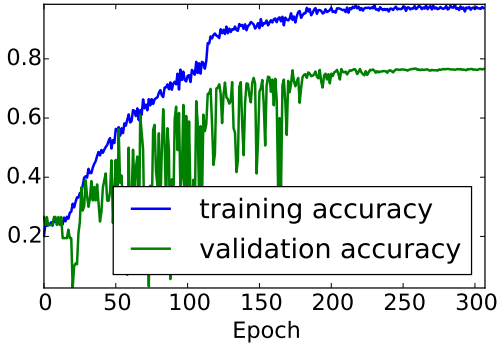
- Test accuracy: classification accuracy over the attack traces
- $N^{\star}$: minimum number of attack traces to make *guessing entropy* of the right key permanently equal to one ($N^{\star}$ estimated over 10 independent attacks)



SH_0



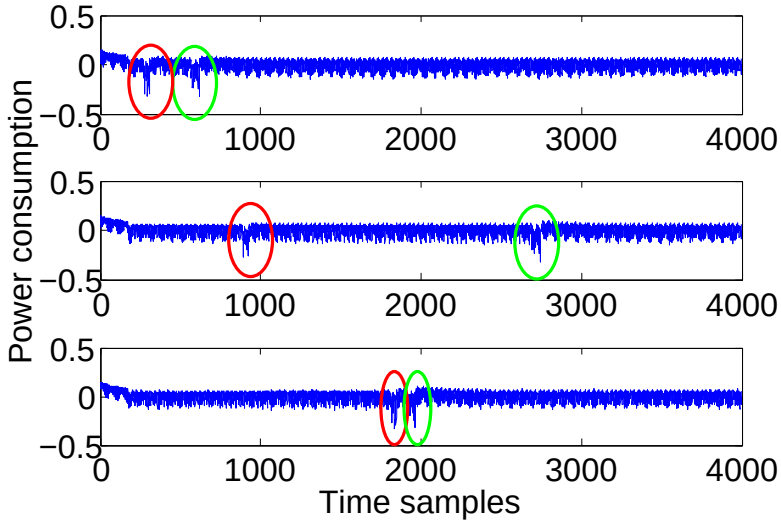
SH_{100}



SH_{500}

		SH_0		SH_{100}		SH_{500}	
Acc	$N^{\star}$	27.0%	$> 1,000$	31.8%	101	**78%**	**7**

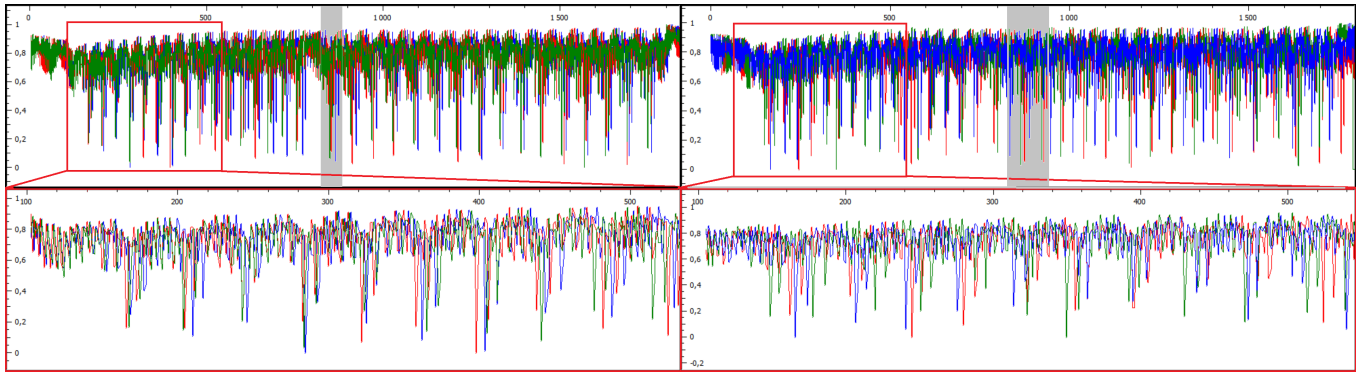
Random Delays - Two Leaking Operations



Two leaking operations

First operation - Test acc: 76.8%, $N^\star = 7$
Second operation - Test acc: 82.5%, $N^\star = 6$

Artificial Jitter



(h) Low artificial jitter (i) High artificial jitter

Target

- Target Variable: $Z = \mathrm{HW}(\mathrm{Sbox}(P \oplus K))$
- $\approx$ 2000 time samples
- Countermeasure: artificial signal treatment simulating clock jitter
- 10000 training traces

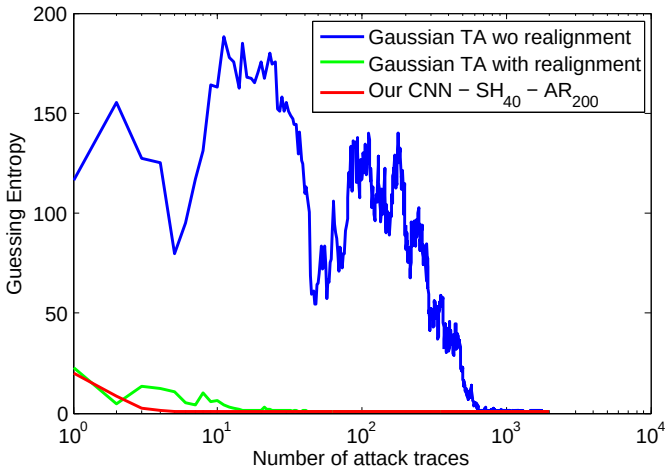
Artificial Jitter (2)

Low_jitter

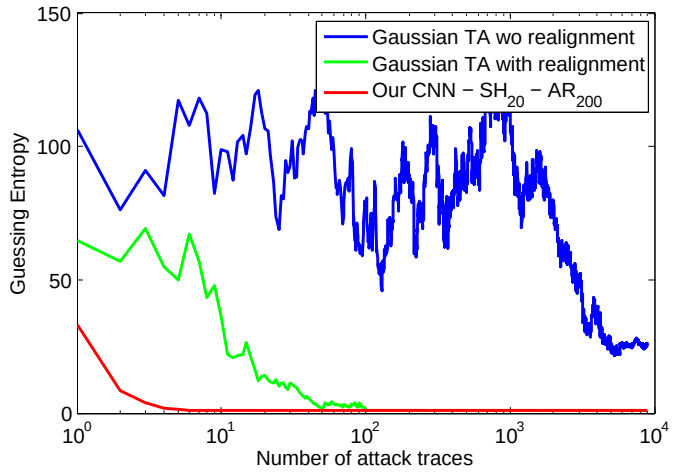
Acc	$N^\star$	SH_0		SH_{20}		SH_{40}	
AR_0		57.4%	14	82.5%	6	83.6%	6
AR_{100}		86.0%	6	87.0%	5	87.5%	6
AR_{200}		86.6%	6	85.7%	6	87.7%	5

High_jitter

Acc	$N^\star$	SH_0		SH_{20}		SH_{40}	
AR_0		40.6%	35	51.1%	9	62.4%	11
AR_{100}		50.2%	15	72.4%	11	73.5%	9
AR_{200}		64.0%	11	75.5%	8	74.4%	8



(j) Low Jitter



(k) High Jitter

Artificial Jitter

DS_low_jitter								
a / *c*	*b* / *d*	SH_0		SH_{20}		SH_{40}		SH_{200}	
AR_0		100.0%	68.7%	99.8%	86.1%	98.9%	84.1%		
		57.4%	14	82.5%	6	83.6%	6		
AR_{100}		87.7%	88.2%	82.4%	88.4%	81.9%	89.6%		
		86.0%	6	87.0%	5	87.5%	6		
AR_{200}		83.2%	88.6%	81.4%	86.9%	80.6%	88.9%		
		86.6%	6	85.7%	6	87.7%	5		
AR_{500}								85.0%	88.6%
								86.2%	5

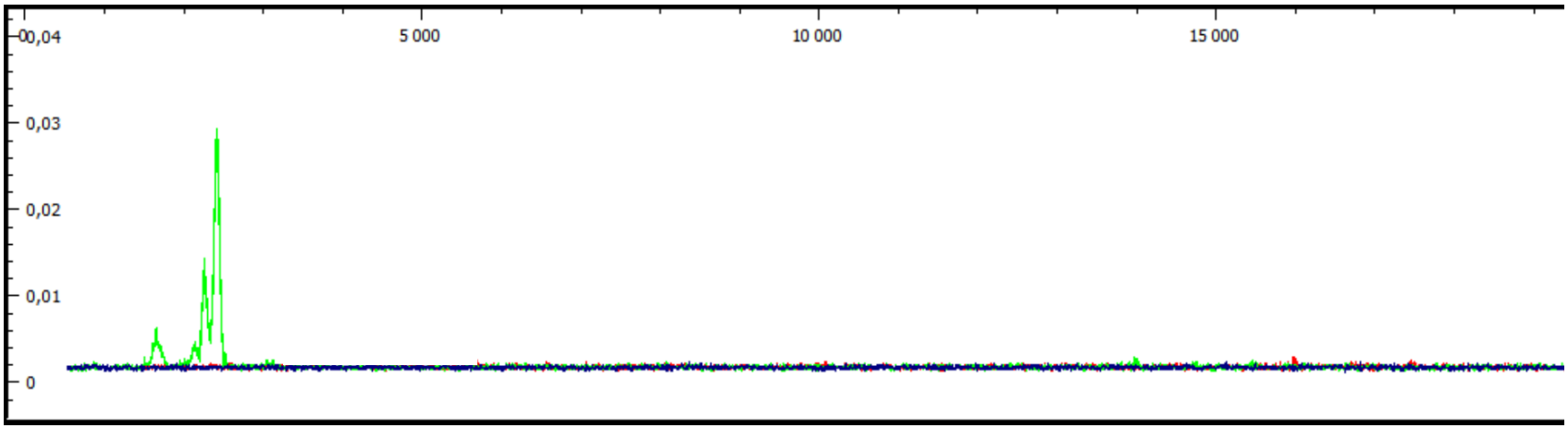
DS_high_jitter								
a / *c*	*b* / *d*	SH_0		SH_{20}		SH_{40}		SH_{200}	
AR_0		100%	45.0%	100%	60.0%	98.5%	67.6%		
		40.6%	35	51.1%	9	62.4%	11		
AR_{100}		90.4%	57.3%	76.6%	73.6%	78.5%	76.4%		
		50.2%	15	72.4%	11	73.5%	9		
AR_{200}		83.1%	67.7%	82.0%	77.1%	82.6%	77.0%		
		64.0%	11	75.5%	8	74.4%	8		
AR_{500}								83.6%	73.4%
								68.2%	11

Real Jitter (1)

Target

- AES hardware implementation
- strong jitter effect
- Target Variable: $Z = \mathrm{Sbox}(P \oplus K)$
- $2,500$ selected time samples
- $99,000$ training traces

SNR first Sbox

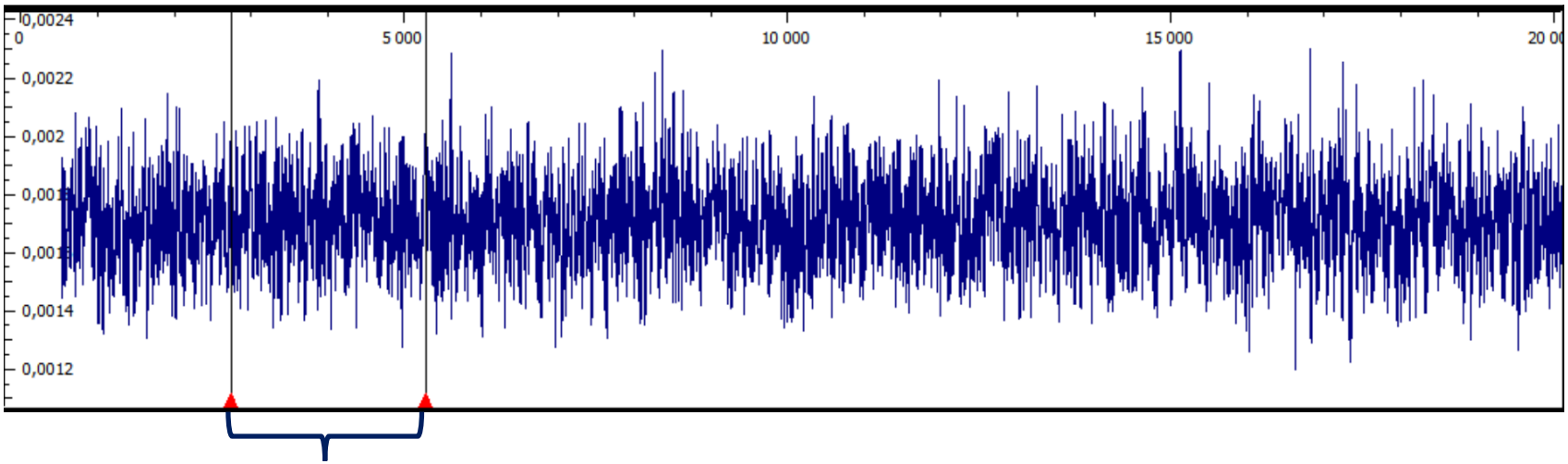


Real Jitter (1)

Target

- AES hardware implementation
- strong jitter effect
- Target Variable: $Z = \mathrm{Sbox}(P \oplus K)$
- $2,500$ selected time samples
- $99,000$ training traces

SNR second Sbox without realignment



Entry region for CNN (2,500 pts)

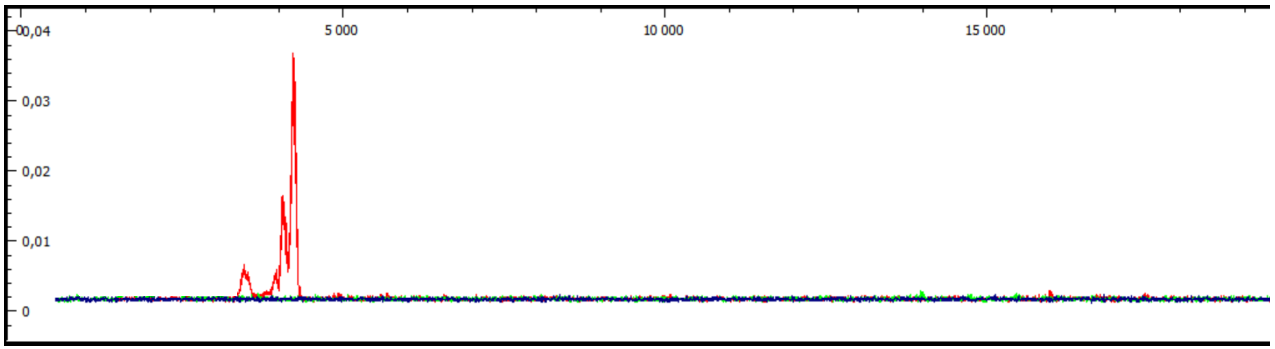
Real Jitter (2)

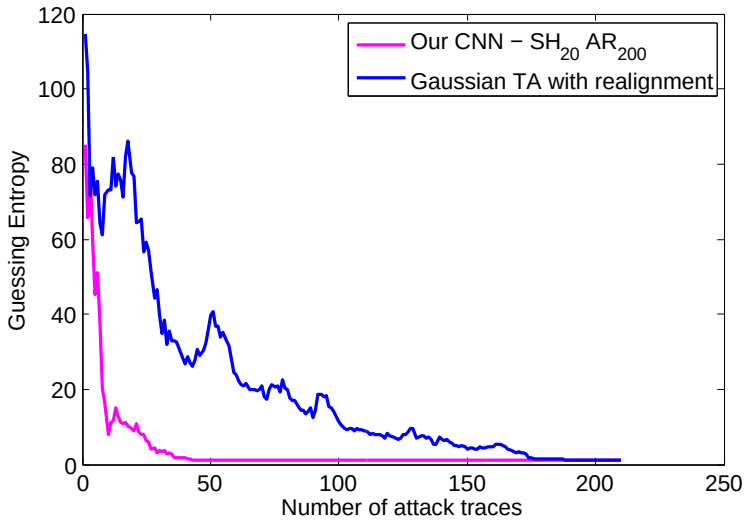
		SH_0AR_0		$SH_{10}AR_{100}$		$SH_{20}AR_{200}$	
Acc	$N^\star$	1.2%	137	1.3%	89	1.8%	54

Real Jitter (2)

		SH_0AR_0		$SH_{10}AR_{100}$		$SH_{20}AR_{200}$	
Acc	$N^\star$	1.2%	137	1.3%	89	1.8%	54

SNR second Sbox with realignment





Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization

Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization
- CNNs provide an integrated approach to directly construct a discriminative model from rough data

Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization
- CNNs provide an integrated approach to directly construct a discriminative model from rough data
- CNN models may have high capacity and require plenty of data to be trained

Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization
- CNNs provide an integrated approach to directly construct a discriminative model from rough data
- CNN models may have high capacity and require plenty of data to be trained
- Data Augmentation provides an answer to the lack of data

Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization
- CNNs provide an integrated approach to directly construct a discriminative model from rough data
- CNN models may have high capacity and require plenty of data to be trained
- Data Augmentation provides an answer to the lack of data
- we proposed two Side-Channel-adapted Data Augmentation techniques (inspired by trace misalignment)

Conclusions about CNN

- State-of-the-Art Template Attack routine separates resynchronization/dimensionality reduction from characterization
- CNNs provide an integrated approach to directly construct a discriminative model from rough data
- CNN models may have high capacity and require plenty of data to be trained
- Data Augmentation provides an answer to the lack of data
- we proposed two Side-Channel-adapted Data Augmentation techniques (inspired by trace misalignment)
- we verified the effectiveness/efficiency of the CNN+Data Augmentation approach over different sets of misaligned data

Contents

Conclusions

- Curse of dimensionality affects the potential optimality of profiling attacks
- In many applicative domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Conclusions

- Curse of dimensionality affects the potential optimality of profiling attacks
- In many applicative domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Today and in the future

- ASCAD database and SCA/DL community
- From CNN to PoIs, visualizing techniques
- Advanced-attack-oriented machine learning task (instead of multiple classification)
- Collision attacks with siamese network

Conclusions

- Curse of dimensionality affects the potential optimality of profiling attacks
- In many applicative domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Today and in the future

- ASCAD database and SCA/DL community
- From CNN to PoIs, visualizing techniques
- Advanced-attack-oriented machine learning task (instead of multiple classification)
- Collision attacks with siamese network

References I

[Arc+06] C. Archambeau et al. "Template Attacks in Principal Subspaces". English. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 1–14. ISBN: 978-3-540-46559-1. DOI: 10.1007/11894063_1. URL: http://dx.doi.org/10.1007/11894063_1.

[BDP10] Martin Bär, Hermann Drexler, and Jürgen Pulkus. "Improved template attacks". In: *COSADE2010* (2010).

[BHW12] Lejla Batina, Jip Hogenboom, and Jasper G.J. van Woudenberg. "Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis". English. In: *Topics in Cryptology CT-RSA 2012*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 383–397. ISBN: 978-3-642-27953-9. DOI: 10.1007/978-3-642-27954-6_24. URL: http://dx.doi.org/10.1007/978-3-642-27954-6_24.

References II

[Bha+14] Shivam Bhasin et al. "Side-channel leakage and trace compression using normalized inter-class variance". In: *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. ACM. 2014, p. 7.

[Bru+15] Nicolas Bruneau et al. "Less is more". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2015, pp. 22–41.

[Car+14] Claude Carlet et al. "Achieving side-channel high-order correlation immunity with leakage squeezing". In: *Journal of Cryptographic Engineering* 4.2 (2014), pp. 107–121.

References III

[CRR03] Suresh Chari, JosyulaR. Rao, and Pankaj Rohatgi. "Template Attacks". English. In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, Cetin K. Koc, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 13–28. ISBN: 978-3-540-00409-7. DOI: 10.1007/3-540-36400-5_3. URL: http://dx.doi.org/10.1007/3-540-36400-5_3.

[Cho+15] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[CK14] Omar Choudary and Markus G Kuhn. "Efficient template attacks". In: *Smart Card Research and Advanced Applications*. Springer, 2014, pp. 253–270.

[Dur+15] François Durvaux et al. "Efficient selection of time samples for higher-order DPA with projection pursuits". In: *Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 34–50.

References IV

[GLRP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. "Templates vs. stochastic methods". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, pp. 15–29.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[OC14] Colin O'Flynn and Zhizhang David Chen. "ChipWhisperer: An open-source platform for hardware embedded security research". In: *Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

References V

[Osw+06] Elisabeth Oswald et al. "Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers". English. In: *Topics in Cryptology CT-RSA 2006*. Ed. by David Pointcheval. Vol. 3860. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 192–207. ISBN: 978-3-540-31033-4. DOI: 10.1007/11605805_13. URL: http://dx.doi.org/10.1007/11605805_13.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. "A stochastic model for differential side channel cryptanalysis". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2005, pp. 30–46.

References VI

[SA08] François-Xavier Standaert and Cedric Archambeau. "Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages". English. In: *Cryptographic Hardware and Embedded Systems CHES 2008*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Vol. 5154. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 411–425. ISBN: 978-3-540-85052-6. DOI: 10.1007/978-3-540-85053-3_26. URL: http://dx.doi.org/10.1007/978-3-540-85053-3_26.

[TM97] Mitchell T. M. *Machine Learning*. McGraw-Hill, New York, 1997.