Hyper Heuristic Cryptography with Mixed Adversarial Nets

Author: Aly Shmahell
Supervisor: Prof. Giovanni De Gasperis

July 11, 2018

University of L'Aquila

Introduction

Q.: What is this dissertation about? A.: Neural Cryptography

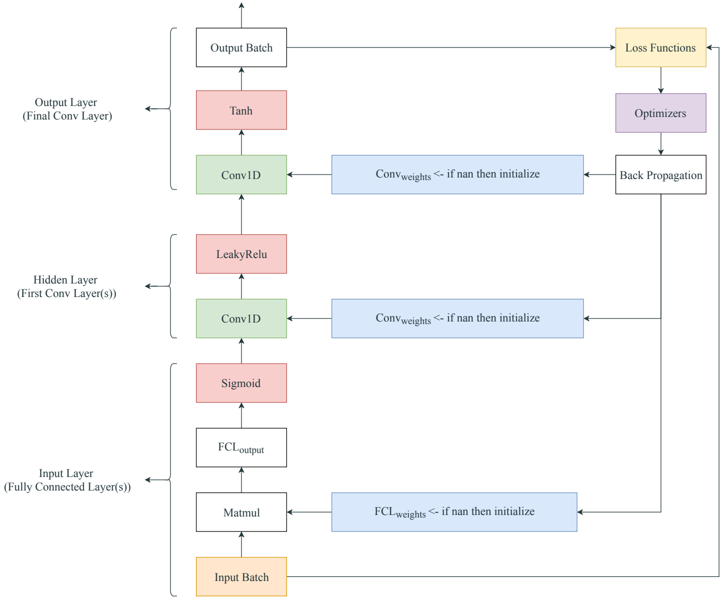
Neural Cryptography: is an interdisciplinary field in Computer Science, combining both artificial intelligence and cryptography, towards the development of stochastic methods, based on artificial neural networks, for use in encryption and cryptanalysis.
Hyper Heuristic Cryptography with Mixed Adversarial Nets adds to the latest experiments in neural cryptography, building upon methodologies presented in a new paper released in 2016 by Google Brain. [1]
These methodologies utilize Adversarial Convolutional Neural Networks.

Current State of Neural Cryptography

- **Neural Cryptography is viable:** The introduction of convolutional networks provides a well tested and understood methodology in reducing problems where local spatial relations in the data matter, which is the case for cryptography.
- **Neural Cryptanalysis is viable:** Having a mixed convolutional net with fully connected layers will teach the network to account for global spatial relations as well, which teaches the net to learn and counter cryptanalysis.
- **Neural Cryptography can be fast:** A result of using convolutions is that the small-sized pattern-finding filter has shared weights (and biases) for all spatial locations which the convolution processes, and this reduces the compute-power required for the whole process compared to other network models.
- **Neural Cryptography is evolved opposite to being patched:** Adversarial computation has been proven to be effective for years in the form of Genetic Algorithms, and adding adversary as a non-supervised generative model provides a better and easier experiment on how to synthesize a new form of cryptography.

Neural Mechanisms for Synthesizing Cryptography

Dummy Net Example

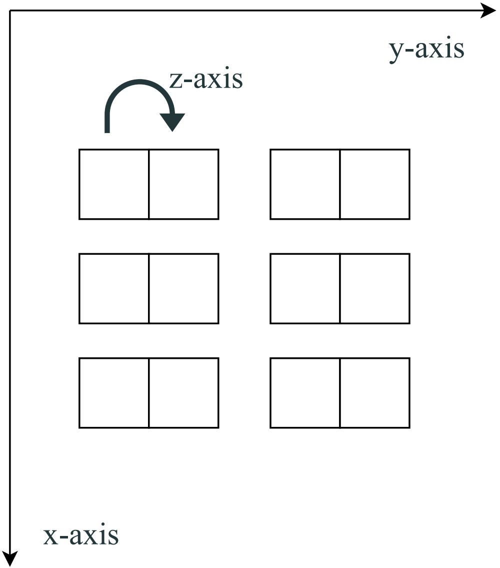


1D Convolution

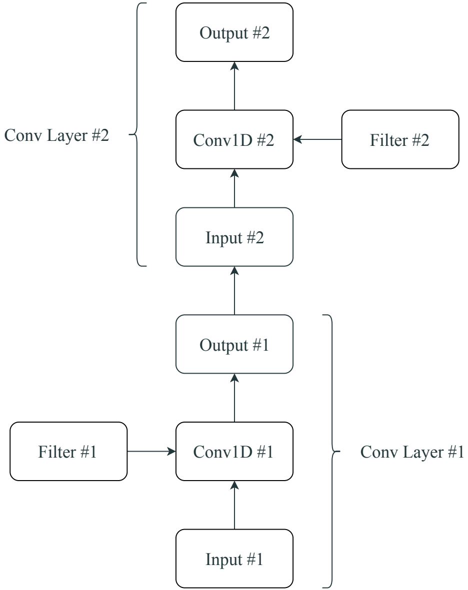
1D Convolution: is the process of using a small window (a filter) to determine local spatial relations over a 1D data sample, regardless of whether the sample is inside an n-D data batch.
In practice, it is an algorithmic utilization of matrix multiplications between the filter and consecutive spatial locations in the sample.

- When 1D Convolution is performed on a batch of samples, the batch should be expanded to 3D (by injecting a z-axis within the y-axis).
- When convoluting over 3D data, the filter should be 3D as well.

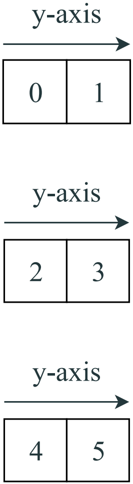
1D Convolution - Continued



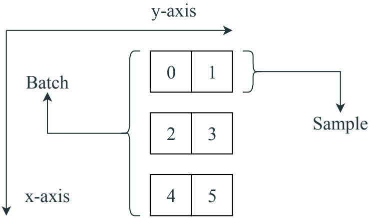
Simple ConvNet Example with 2 Conv Layers

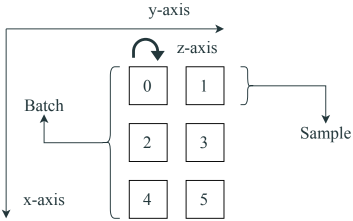


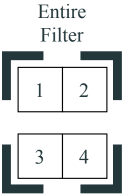
Simple ConvNet Example with 2 Conv Layers - Continued



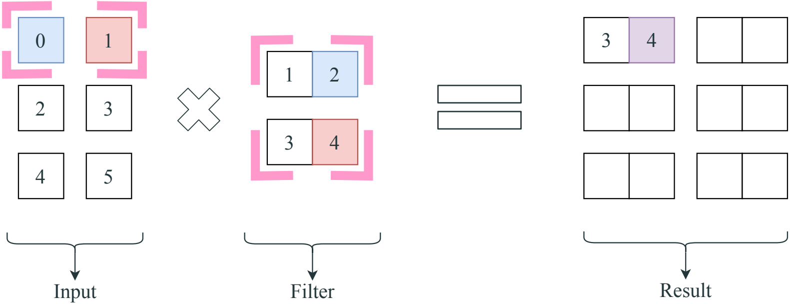
Simple ConvNet Example with 2 Conv Layers - Continued







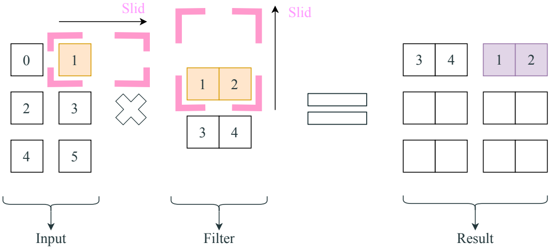
Simple ConvNet Example with 2 Conv Layers - Continued



$$Result[0][0][1] = Input[0][0][0] * Filter[0][0][1] + Input[0][1][0] * Filter[1][0][1]$$

$$4 = 0 * 2 + 1 * 4$$

Simple ConvNet Example with 2 Conv Layers - Continued



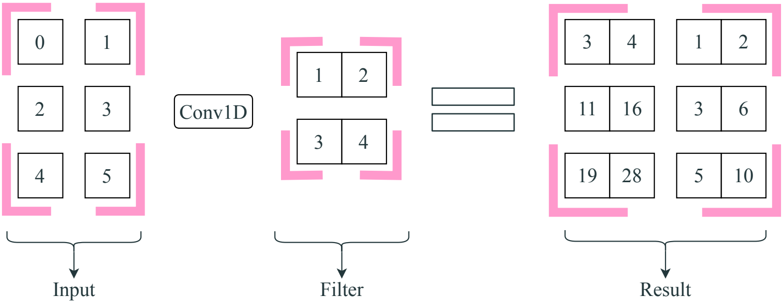
$$Result[0][1][0] = Input[0][1][0] * Filter[0][0][0]$$

$$1 = 1 * 1$$

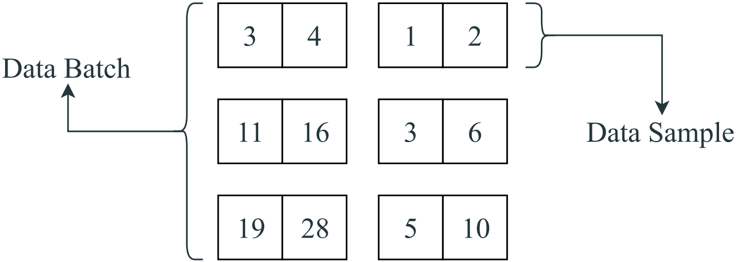
$$Result[0][1][1] = Input[0][1][0] * Filter[0][0][1]$$

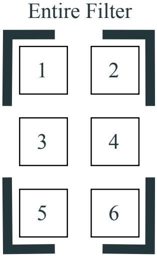
$$2 = 1 * 2$$

Simple ConvNet Example with 2 Conv Layers - Continued

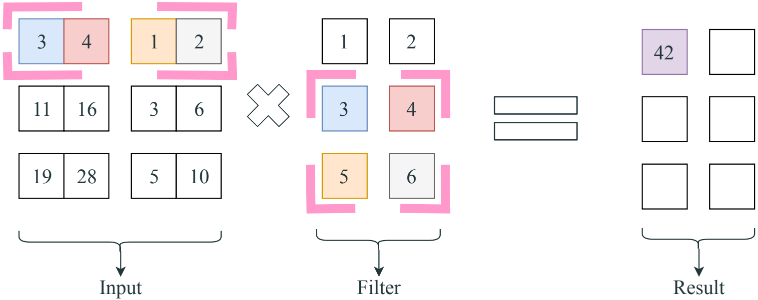


Simple ConvNet Example with 2 Conv Layers - Continued





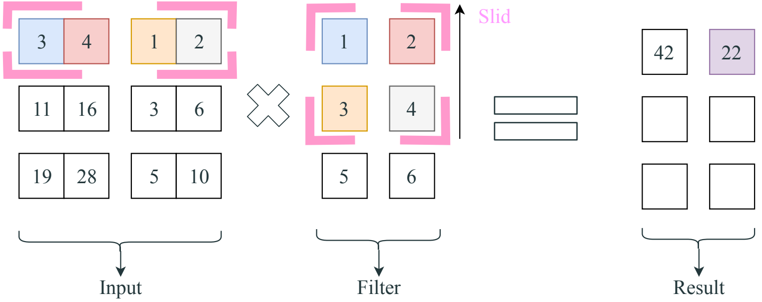
Simple ConvNet Example with 2 Conv Layers - Continued



$$
\begin{aligned}
Result[0][0][0] = {} & Input[0][0][0] * Filter[1][0][0] \\
& + Input[0][0][1] * Filter[1][1][0] \\
& + Input[0][1][0] * Filter[2][0][0] \\
& + Input[0][1][1] * Filter[2][1][0]
\end{aligned}
$$

$$42 = 3 * 3 + 4 * 4 + 1 * 5 + 2 * 6$$

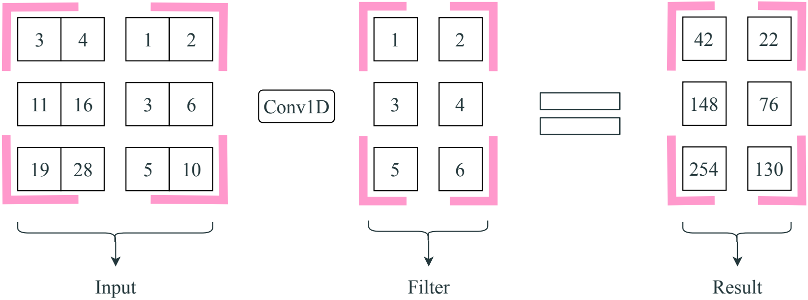
Simple ConvNet Example with 2 Conv Layers - Continued



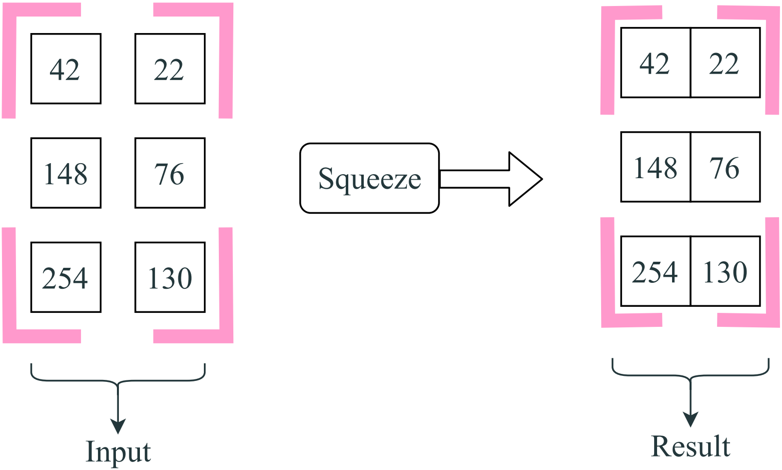
$$
\begin{aligned}
Result[0][1][0] = Input[0][0][0] * Filter[0][0][0] \\
+ Input[0][0][1] * Filter[0][1][0] \\
+ Input[0][1][0] * Filter[1][0][0] \\
+ Input[0][1][1] * Filter[1][1][0]
\end{aligned}
$$

$$42 = 3 * 1 + 4 * 2 + 1 * 3 + 2 * 4$$

Simple ConvNet Example with 2 Conv Layers - Continued



Simple ConvNet Example with 2 Conv Layers - Continued



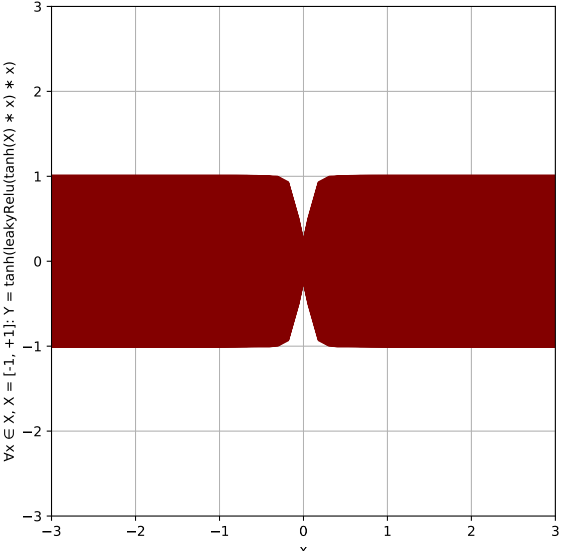
Activation Functions after each Layer

- "*Sigmoid* $\rightarrow$ *LeakyRealu* $\rightarrow$ *Sigmoid*".
- "*Tanh* $\rightarrow$ *LeakyRealu* $\rightarrow$ *Tanh*".
- "*Sigmoid* $\rightarrow$ *LeakyRealu* $\rightarrow$ *Tanh*".

The choice was made to use: "*Sigmoid* $\rightarrow$ *LeakyRealu* $\rightarrow$ *Tanh*".

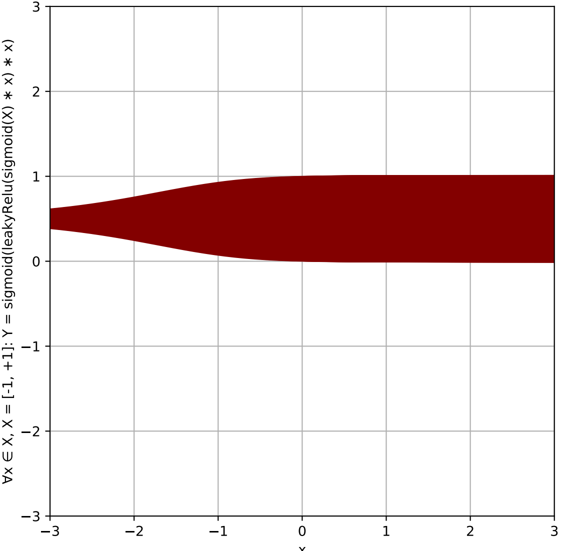
Activation Functions after each Layer - Numerical Analysis

$"Tanh \rightarrow LeakyRealu \rightarrow Tanh"$.



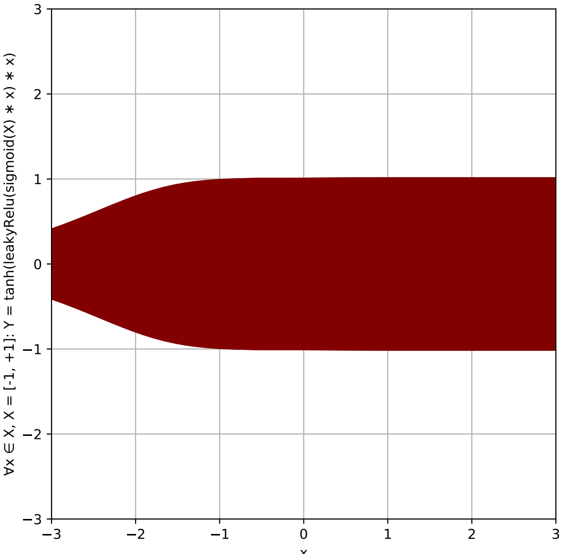
Activation Functions after each Layer - Numerical Analysis

"Sigmoid $\rightarrow$ LeakyRealu $\rightarrow$ Sigmoid".



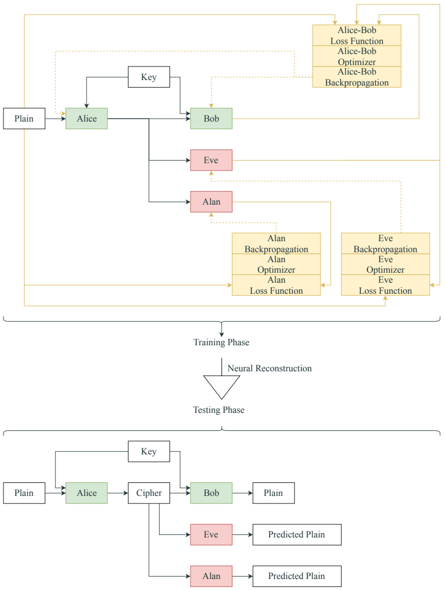
Activation Functions after each Layer - Numerical Analysis

"*Sigmoid* $\rightarrow$ *LeakyRealu* $\rightarrow$ *Tanh*".

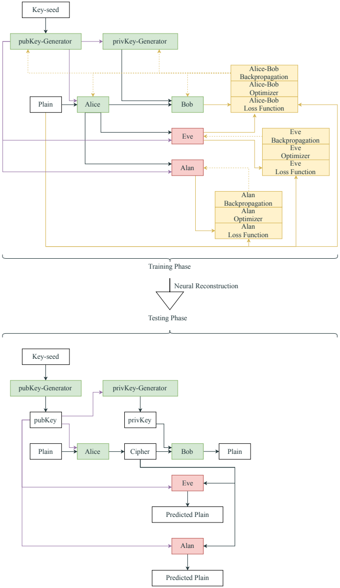


Scheme Structures

Symmetric Scheme



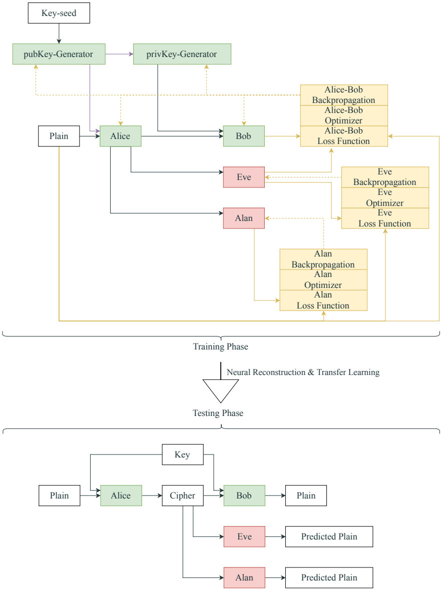
Asymmetric Scheme



Transfer Learning

Transfer Learning: Given a source domain D_S and learning task T_S , a target domain D_T and learning task T_T , transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S , where $D_S \neq D_T$, or $T_S \neq T_T$. [2]

Hybrid Scheme

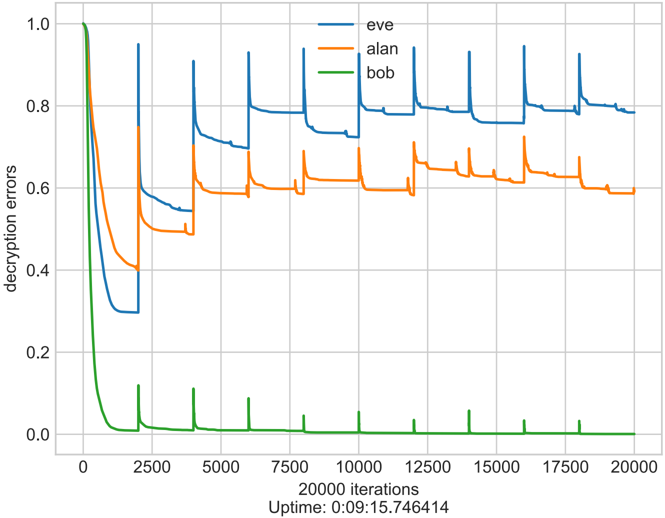


Project Structure

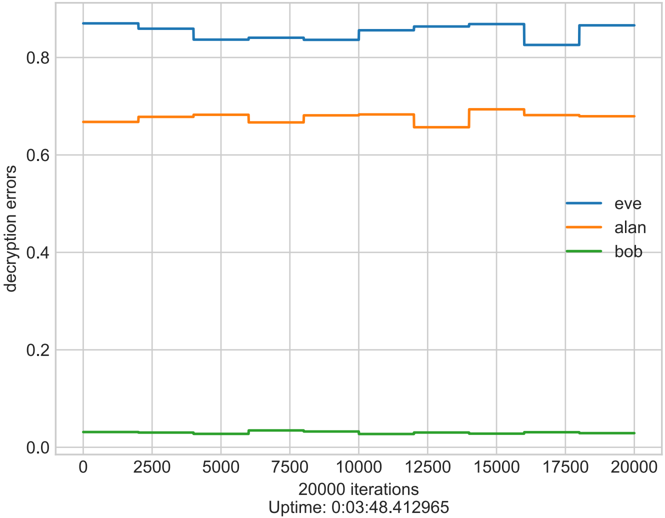
Class Diagram

Results

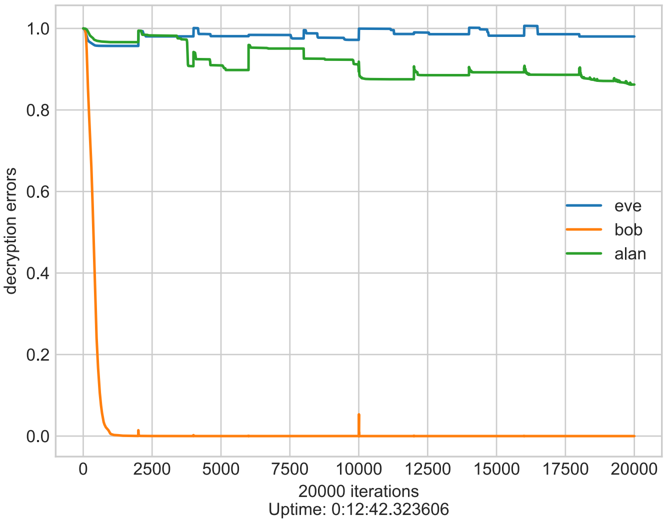
Thesis Results - Symmetric Training



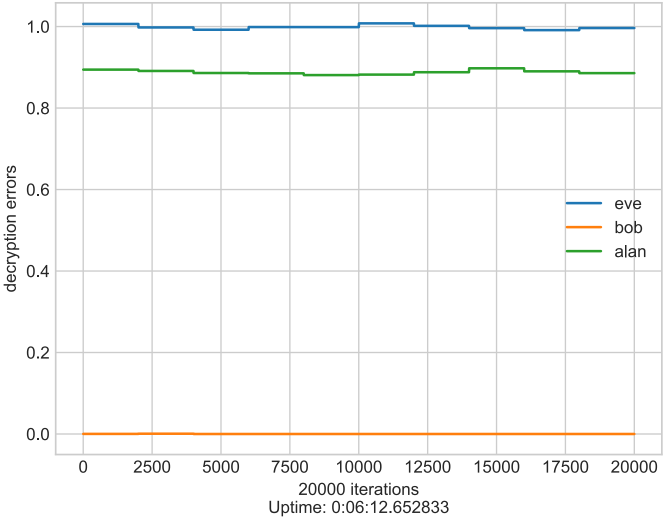
Thesis Results - Symmetric Testing



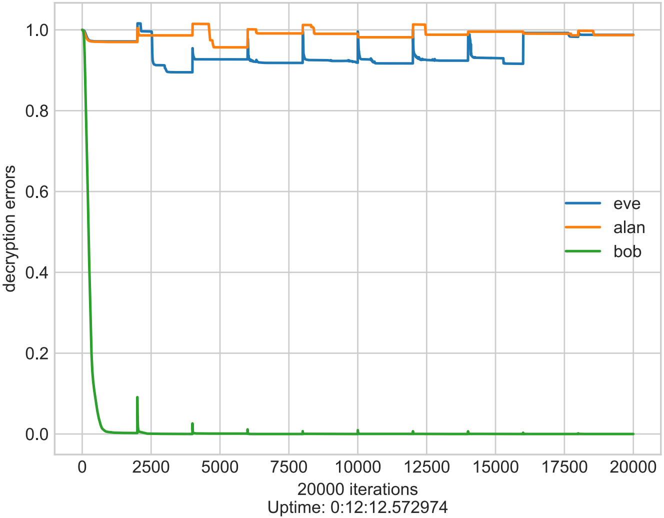
Thesis Results - Asymmetric Training



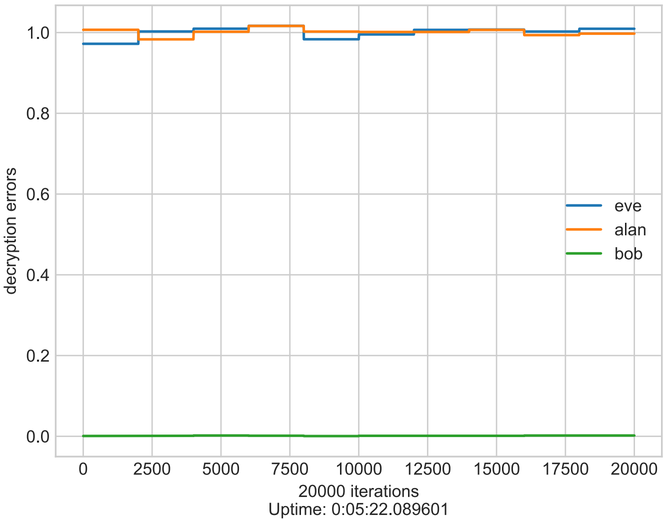
Thesis Results - Asymmetric Testing



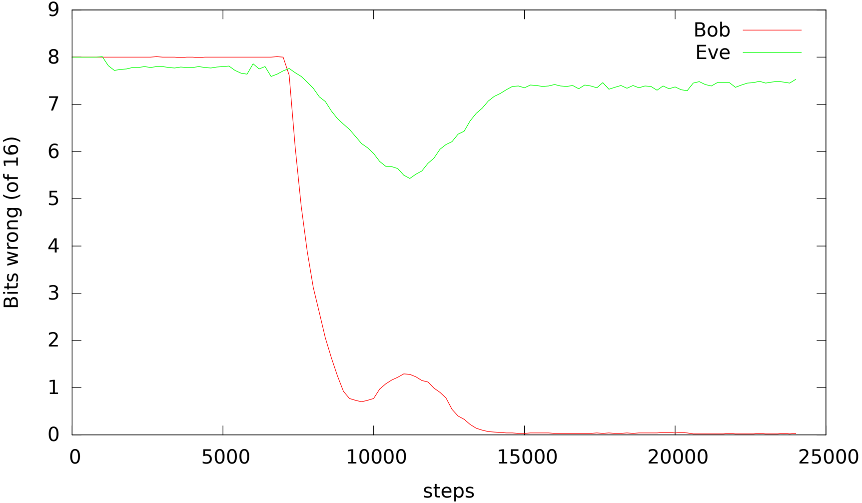
Thesis Results - Hybrid Training



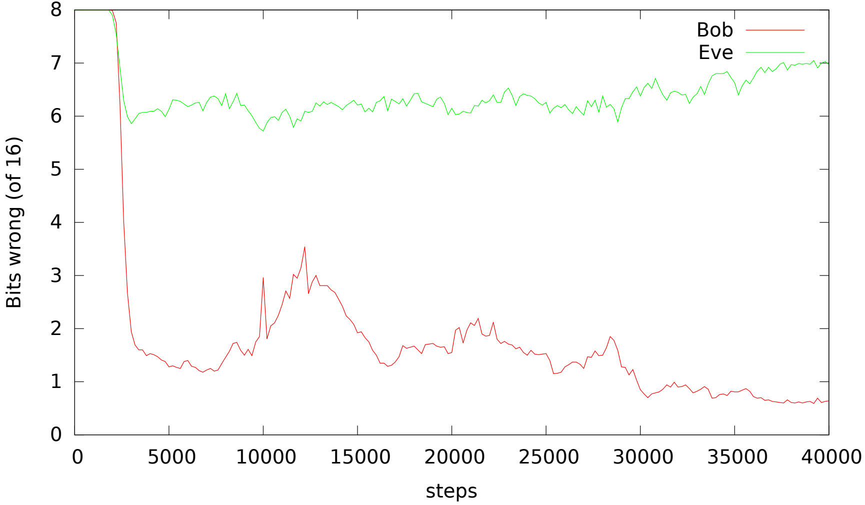
Thesis Results - Hybrid Testing



Symmetric Results from Google Brain - for Comparison



Asymmetric Results from Google Brain - for Comparison



References

[1] Martín Abadi and David G. Andersen. Learning to protect communications with adversarial neural cryptography. *CoRR*, abs/1610.06918, 2016.

[2] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.